

MooTools API

1.11

Contents

Core.js	3
Class.js	7
Class.Extras.js	9
Array.js	13
String.js	20
Function.js	24
Number.js	28
Element.js	30
Element.Event.js	41
Element.Filters.js	45
Element.Selectors.js	46
Element.Form.js	48
Element.Dimensions.js	49
Window.DomReady.js	52
Window.Size.js	53
Fx.Base.js	55
Fx.CSS.js	57
Fx.Style.js	58
Fx.Styles.js	60
Fx.Elements.js	62
Fx.Scroll.js	63
Fx.Slide.js	65
Fx.Transitions.js	67
Drag.Base.js	71
Drag.Move.js	73
XHR.js	74
Ajax.js	76
Cookie.js	79
Json.js	81
Json.Remote.js	83
Assets.js	84
Hash.js	86
Hash.Cookie.js	89
Color.js	91
Scroller.js	94
Slider.js	95
SmoothScroll.js	96
Sortable.js	97
Tips.js	98
Group.js	100
Accordion.js	101

Core.js

Mootools - MSNS Object Oriented javascript.

License:

MIT-style license.

Summary

Core.js	Mootools - MSNS Object Oriented javascript.
Abstract	Abstract class, to be used as singleton. Will add .extend to any object
window	Some properties are attached to the window object by the browser detection.

MooTools Copyright:

copyright (c) 2007 Valerio Proietti, <<http://mad4milk.net>>

MooTools Credits:

- Class is slightly based on Base.js <<http://dean.edwards.name/weblog/2006/03/base/>> (c) 2006 Dean Edwards, License <<http://creativecommons.org/licenses/LGPL/2.1/>>
- Some functions are inspired by those found in prototype.js <<http://prototype.conio.net/>> (c) 2005 Sam Stephenson sam [at] conio [dot] net, MIT-style license
- Documentation by Aaron Newton (aaron.newton [at] cnet [dot] com) and Valerio Proietti.

Function \$defined

Returns true if the passed in value/object is defined, that means is not null or undefined.

Arguments:

obj object to inspect

Function \$type

Returns the type of object that matches the element passed in.

Arguments:

obj the object to inspect.

Example:

```
var myString = 'hello';
$type(myString); //returns "string"
```

Function \$merge

merges a number of objects recursively without referencing them or their sub-objects.

Arguments:

any number of objects.

Example:

```
var mergedObj = $merge(obj1, obj2, obj3);  
//obj1, obj2, and obj3 are unaltered
```

Function \$extend

Copies all the properties from the second passed object to the first passed Object.

If you do `myWhatever.extend = $extend` the first parameter will become `myWhatever`, and your extend function will only need one parameter.

Example:

```
var firstOb = {  
  'name': 'John',  
  'lastName': 'Doe'  
};  
var secondOb = {  
  'age': '20',  
  'sex': 'male',  
  'lastName': 'Dorian'  
};  
$extend(firstOb, secondOb);  
//firstOb will become:  
{  
  'name': 'John',  
  'lastName': 'Dorian',  
  'age': '20',  
  'sex': 'male'  
};  
(end)
```

Returns:

The first object, extended.

Function \$native

Will add a `.extend` method to the objects passed as a parameter, but the property passed in will be copied to the object's prototype only if non previously existent.

Its handy if you dont want the `.extend` method of an object to overwrite existing methods.

Used automatically in MooTools to implement Array/String/Function/Number methods to browser that dont support them whitout manual checking.

Arguments:

a number of classes/native javascript objects

Function \$chk

Returns true if the passed in value/object exists or is 0, otherwise returns false.
Useful to accept zeroes.

Arguments:

obj object to inspect

Function \$pick

Returns the first object if defined, otherwise returns the second.

Arguments:

obj object to test
picked the default to return

Example:

```
function say(msg){  
  alert($pick(msg, 'no message supplied'));  
}  
(end)
```

Function \$random

Returns a random integer number between the two passed in values.

Arguments:

min integer, the minimum value (inclusive).
max integer, the maximum value (inclusive).

Returns:

a random integer between min and max.

Function \$time

Returns the current timestamp

Returns:

a timestamp integer.

Function \$clear

clears a timeout or an Interval.

Returns:

null

Arguments:

timer the setInterval or setTimeout to clear.

Example:

```
var myTimer = myFunction.delay(5000); //wait 5 seconds and execute my function.  
myTimer = $clear(myTimer); //nevermind
```

Class Abstract

Abstract class, to be used as singleton. Will add .extend to any object

Arguments:

an object

Returns:

the object with an .extend property, equivalent to <\$extend>.

Class window

Some properties are attached to the window object by the browser detection.

Note:

browser detection is entirely object-based. We dont sniff.

Properties

window.ie	will be set to true if the current browser is internet explorer (any).
window.ie6	will be set to true if the current browser is internet explorer 6.
window.ie7	will be set to true if the current browser is internet explorer 7.
window.gecko	will be set to true if the current browser is Mozilla/Gecko.
window.webkit	will be set to true if the current browser is Safari/Konqueror.
window.webkit419	will be set to true if the current browser is Safari2 / webkit till version 419.
window.webkit420	will be set to true if the current browser is Safari3 (Webkit SVN Build) / webkit over version 419.
window.opera	is set to true by opera itself.

Class.js

Contains the Class Function, aims to ease the creation of reusable Classes.

License:

MIT-style license.

Summary

Class.js	Contains the Class Function, aims to ease the creation of reusable Classes.
Class	The base class object of the http://mootools.net framework.
empty	Returns an empty function
extend	Returns the copy of the Class extended with the passed in properties.
implement	Implements the passed in properties to the base Class prototypes, altering the base class, unlike Class.extend.

Class

The base class object of the <http://mootools.net> framework.

Creates a new class, its initialize method will fire upon class instantiation.

Initialize wont fire on instantiation when you pass *null*.

Arguments:

`properties` the collection of properties that apply to the class.

Example:

```
var Cat = new Class({
  initialize: function(name){
    this.name = name;
  }
});
var myCat = new Cat('Micia');
alert(myCat.name); //alerts 'Micia'
(end)
```

Method `empty`

Returns an empty function

Method `extend`

Returns the copy of the Class extended with the passed in properties.

Arguments:

`properties` the properties to add to the base class in this new Class.

Example:

```
var Animal = new Class({
```

```

initialize: function(age){
  this.age = age;
}
});
var Cat = Animal.extend({
  initialize: function(name, age){
    this.parent(age); //will call the previous initialize;
    this.name = name;
  }
});
var myCat = new Cat('Micia', 20);
alert(myCat.name); //alerts 'Micia'
alert(myCat.age); //alerts 20
(end)

```

Method **implement**

Implements the passed in properties to the base Class prototypes, altering the base class, unlike <Class.extend>.

Arguments:

properties the properties to add to the base class.

Example:

```

var Animal = new Class({
  initialize: function(age){
    this.age = age;
  }
});
Animal.implement({
  setName: function(name){
    this.name = name
  }
});
var myAnimal = new Animal(20);
myAnimal.setName('Micia');
alert(myAnimal.name); //alerts 'Micia'
(end)

```

Class.Extras.js

Contains common implementations for custom classes. In Mootools is implemented in <Ajax>, <XHR> and <Fx.Base> and many more.

License:

MIT-style license.

Summary

Class.Extras.js	Contains common implementations for custom classes. In Mootools is implemented in Ajax, XHR and Fx.Base and many more.
Chain	An "Utility" Class. Its methods can be implemented with Class.implement into any Class.
chain	adds a function to the Chain instance stack.
callChain	Executes the first function of the Chain instance stack, then removes it. The first function will then become the second.
clearChain	Clears the stack of a Chain instance.
Events	An "Utility" Class. Its methods can be implemented with Class.implement into any Class.
addEvent	adds an event to the stack of events of the Class instance.
fireEvent	fires all events of the specified type in the Class instance.
removeEvent	removes an event from the stack of events of the Class instance.
Options	An "Utility" Class. Its methods can be implemented with Class.implement into any Class.
setOptions	sets this.options

Class Chain

An "Utility" Class. Its methods can be implemented with <Class.implement> into any <Class>.

Currently implemented in <Fx.Base>, <XHR> and <Ajax>. In <Fx.Base> for example, is used to execute a list of function, one after another, once the effect is completed.

The functions will not be fired all together, but one every completion, to create custom complex animations.

Example:

```
var myFx = new Fx.Style('element', 'opacity');

myFx.start(1,0).chain(function(){
myFx.start(0,1);
}).chain(function(){
myFx.start(1,0);
}).chain(function(){
myFx.start(0,1);
});

//the element will appear and disappear three times
(end)
```

Method chain

adds a function to the Chain instance stack.

Arguments:

fn the function to append.

Method callChain

Executes the first function of the Chain instance stack, then removes it. The first function will then become the second.

Method `clearChain`

Clears the stack of a Chain instance.

Class `Events`

An "Utility" Class. Its methods can be implemented with `<Class.implement>` into any `<Class>`.

In `<Fx.Base>` Class, for example, is used to give the possibility add any number of functions to the Effects events, like `onComplete`, `onStart`, `onCancel`.

Events in a Class that implements `<Events>` can be either added as an option, or with `addEvent`. Never with `.options.onEventName`.

Example:

```
var myFx = new Fx.Style('element', 'opacity').addEvent('onComplete', function(){
  alert('the effect is completed');
}).addEvent('onComplete', function(){
  alert('I told you the effect is completed');
});

myFx.start(0,1);
//upon completion it will display the 2 alerts, in order.
(end)
```

Implementing:

This class can be implemented into other classes to add the functionality to them.

Goes well with the `<Options>` class.

Example:

```
var Widget = new Class({
  initialize: function(){},
  finish: function(){
    this.fireEvent('onComplete');
  }
});

Widget.implement(new Events);

//later...
var myWidget = new Widget();
myWidget.addEvent('onComplete', myfunction);
(end)
```

Method `addEvent`

adds an event to the stack of events of the Class instance.

Arguments:

type string; the event name (e.g. 'onComplete')

fn function to execute

Method fireEvent

fires all events of the specified type in the Class instance.

Arguments:

type string; the event name (e.g. 'onComplete')

args array or single object; arguments to pass to the function; if more than one argument, must be an array

delay (integer) delay (in ms) to wait to execute the event

Example:

```
var Widget = new Class({
  initialize: function(arg1, arg2){
    ...
    this.fireEvent("onInitialize", [arg1, arg2], 50);
  }
});
Widget.implement(new Events);
(end)
```

Method removeEvent

removes an event from the stack of events of the Class instance.

Arguments:

type string; the event name (e.g. 'onComplete')

fn function that was added

Class Options

An "Utility" Class. Its methods can be implemented with <Class.implement> into any <Class>. Used to automate the options settings, also adding Class <Events> when the option begins with on.

Example:

```
var Widget = new Class({
  options: {
    color: '#fff',
    size: {
      width: 100
      height: 100
    }
  },
  initialize: function(options){
```

```
this.setOptions(options);
}
});
Widget.implement(new Options);
//later...
var myWidget = new Widget({
color: '#f00',
size: {
width: 200
}
});
//myWidget.options = {color: #f00, size: {width: 200, height: 100}}
(end)
```

Method `setOptions`

sets this.options

Arguments:

defaults	object; the default set of options
options	object; the user entered options. can be empty too.

Note:

if your Class has <Events> implemented, every option beginning with on, followed by a capital letter (onComplete) becomes an Class instance event.

Array.js

Contains Array prototypes, <\$A>, <\$each>

License:

MIT-style license.

Summary

Array.js	Contains Array prototypes, \$A, \$each
Array	A collection of The Array Object prototype methods.
forEach	Iterates through an array; This method is only available for browsers without native *forEach* support.
filter	This method is provided only for browsers without native *filter* support.
map	This method is provided only for browsers without native *map* support.
every	This method is provided only for browsers without native *every* support.
some	This method is provided only for browsers without native *some* support.
indexOf	This method is provided only for browsers without native *indexOf* support.
each	Same as Array.forEach.
copy	returns a copy of the array.
remove	Removes all occurrences of an item from the array.
contains	Tests an array for the presence of an item.
associate	Creates an object with key-value pairs based on the array of keywords passed in
extend	Extends an array with another one.
merge	merges an array in another array, without duplicates. (case- and type-sensitive)
include	includes the passed in element in the array, only if its not already present. (case- and type-sensitive)
getRandom	returns a random item in the Array
getLast	returns the last item in the Array

Class Array

A collection of The Array Object prototype methods.

Method forEach

Iterates through an array; This method is only available for browsers without native *forEach* support.

For more info see <http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference:Global_Objects:Array:forEach>

forEach executes the provided function (callback) once for each element present in the array. callback is invoked only for indexes of the array which have assigned values; it is not invoked for indexes which have been deleted or which have never been assigned values.

Arguments:

fn function to execute with each item in the array; passed the item and the index of that item in the array

bind the object to bind "this" to (see <Function.bind>)

Example:

```
['apple', 'banana', 'lemon'].each(function(item, index){
  alert(index + " = " + item); //alerts "0 = apple" etc.
}, bindObj); //optional second arg for binding, not used here
```

Method `filter`

This method is provided only for browsers without native `*filter*` support.

For more info see http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference:Objects:Array:filter

`*filter*` calls a provided callback function once for each element in an array, and constructs a new array of all the values for which callback returns a true value. callback is invoked only for indexes of the array which have assigned values; it is not invoked for indexes which have been deleted or which have never been assigned values. Array elements which do not pass the callback test are simply skipped, and are not included in the new array.

Arguments:

`fn` function to execute with each item in the array; passed the item and the index of that item in the array

`bind` the object to bind "this" to (see `<Function.bind>`)

Example:

```
var biggerThanTwenty = [10,3,25,100].filter(function(item, index){
  return item > 20;
});
//biggerThanTwenty = [25,100]
```

Method `map`

This method is provided only for browsers without native `*map*` support.

For more info see http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference:Global_Objects:Array:map

`*map*` calls a provided callback function once for each element in an array, in order, and constructs a new array from the results. callback is invoked only for indexes of the array which have assigned values; it is not invoked for indexes which have been deleted or which have never been assigned values.

Arguments:

`fn` function to execute with each item in the array; passed the item and the index of that item in the array

`bind` the object to bind "this" to (see `<Function.bind>`)

Example:

```
var timesTwo = [1,2,3].map(function(item, index){
  return item*2;
});
//timesTwo = [2,4,6];
```

Method `every`

This method is provided only for browsers without native `*every*` support.

For more info see <http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference:Global_Objects:Array:every>

every executes the provided callback function once for each element present in the array until it finds one where callback returns a false value. If such an element is found, the every method immediately returns false. Otherwise, if callback returned a true value for all elements, every will return true. callback is invoked only for indexes of the array which have assigned values; it is not invoked for indexes which have been deleted or which have never been assigned values.

Arguments:

fn function to execute with each item in the array; passed the item and the index of that item in the array
bind the object to bind "this" to (see <[Function.bind](#)>)

Example:

```
var areAllBigEnough = [10,4,25,100].every(function(item, index){
  return item > 20;
});
//areAllBigEnough = false
```

Method **some**

This method is provided only for browsers without native ***some*** support.

For more info see <http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference:Global_Objects:Array:some>

some executes the callback function once for each element present in the array until it finds one where callback returns a true value. If such an element is found, some immediately returns true. Otherwise, some returns false. callback is invoked only for indexes of the array which have assigned values; it is not invoked for indexes which have been deleted or which have never been assigned values.

Arguments:

fn function to execute with each item in the array; passed the item and the index of that item in the array
bind the object to bind "this" to (see <[Function.bind](#)>)

Example:

```
var isAnyBigEnough = [10,4,25,100].some(function(item, index){
  return item > 20;
});
//isAnyBigEnough = true
```

Method **indexOf**

This method is provided only for browsers without native ***indexOf*** support.

For more info see <http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference:Global_Objects:Array:indexOf>

indexOf compares a search element to elements of the Array using strict equality (the same method used by the ===, or triple-equals, operator).

Arguments:

item any type of object; element to locate in the array

from integer; optional; the index of the array at which to begin the search (defaults to 0)

)

Example:

```
['apple','lemon','banana'].indexOf('lemon'); //returns 1
['apple','lemon'].indexOf('banana'); //returns -1
```

Method each

Same as <Array.forEach>.

Arguments:

fn function to execute with each item in the array; passed the item and the index of that item in the array

bind optional, the object that the "this" of the function will refer to.

Example:

```
var Animals = ['Cat', 'Dog', 'Coala'];
Animals.each(function(animal){
  document.write(animal)
});
```

Method copy

returns a copy of the array.

Returns:

a new array which is a copy of the current one.

Arguments:

start integer; optional; the index where to start the copy, default is 0. If negative, it is taken as the offset from the end of the array.

length integer; optional; the number of elements to copy. By default, copies all elements from start to the end of the array.

Example:

```
var letters = ["a","b","c"];
var copy = letters.copy(); // ["a","b","c"] (new instance)
```

Method remove

Removes all occurrences of an item from the array.

Arguments:

item the item to remove

Returns:

the Array with all occurrences of the item removed.

Example:

```
["1","2","3","2"].remove("2") // ["1","3"];
```

Method contains

Tests an array for the presence of an item.

Arguments:

item the item to search for in the array.
from integer; optional; the index at which to begin the search, default is 0. If negative,
 it is taken as the offset from the end of the array.

Returns:

true - the item was found

false - it wasn't

Example:

```
["a","b","c"].contains("a"); // true  
["a","b","c"].contains("d"); // false
```

Method associate

Creates an object with key-value pairs based on the array of keywords passed in and the current content of the array.

Arguments:

keys the array of keywords.

Example:

```
var Animals = ['Cat', 'Dog', 'Coala', 'Lizard'];  
var Speech = ['Miao', 'Bau', 'Fruuu', 'Mute'];  
var Speeches = Animals.associate(Speech);  
//Speeches['Miao'] is now Cat.  
//Speeches['Bau'] is now Dog.  
//...  
(end)
```

Method `extend`

Extends an array with another one.

Arguments:

array the array to extend ours with

Example:

```
var Animals = ['Cat', 'Dog', 'Coala'];
Animals.extend(['Lizard']);
//Animals is now: ['Cat', 'Dog', 'Coala', 'Lizard'];
```

Method `merge`

merges an array in another array, without duplicates. (case- and type-sensitive)

Arguments:

array the array to merge from.

Example:

```
['Cat', 'Dog'].merge(['Dog', 'Coala']); //returns ['Cat', 'Dog', 'Coala']
```

Method `include`

includes the passed in element in the array, only if its not already present. (case- and type-sensitive)

Arguments:

item item to add to the array (if not present)

Example:

```
['Cat', 'Dog'].include('Dog'); //returns ['Cat', 'Dog']
['Cat', 'Dog'].include('Coala'); //returns ['Cat', 'Dog', 'Coala']
```

Method `getRandom`

returns a random item in the Array

Method `getLast`

returns the last item in the Array

Function `$A()`

Same as <Array.copy>, but as function.

Useful to apply Array prototypes to iterable objects, as a collection of DOM elements or the arguments object.

Example:

```
function myFunction(){
  $A(arguments).each(argument, function(){
    alert(argument);
  });
};
//the above will alert all the arguments passed to the function myFunction.
(end)
```

Function `$each`

Use to iterate through iterables that are not regular arrays, such as builtin `getElementsByTagName` calls, arguments of a function, or an object.

Arguments:

`iterable` an iterable element or an object.

`function` function to apply to the iterable.

`bind` optional, the 'this' of the function will refer to this object.

Function argument:

The function argument will be passed the following arguments.

`item` - the current item in the iterator being processed

`index` - integer; the index of the item, or key in case of an object.

Examples:

```
$each(['Sun','Mon','Tue'], function(day, index){
  alert('name:' + day + ', index: ' + index);
});
//alerts "name: Sun, index: 0", "name: Mon, index: 1", etc.
//over an object
$each({first: "Sunday", second: "Monday", third: "Tuesday"}, function(value, key){
  alert("the " + key + " day of the week is " + value);
});
//alerts "the first day of the week is Sunday",
//"the second day of the week is Monday", etc.
(end)
```

String.js

Contains String prototypes.

License:

MIT-style license.

Summary

String.js	Contains String prototypes.
String	A collection of The String Object prototype methods.
test	Tests a string with a regular expression.
toInt	parses a string to an integer.
toFloat	parses a string to a float.
camelCase	Converts a hiphenated string to a camelcase string.
hyphenate	Converts a camelCased string to a hyphen-ated string.
capitalize	Converts the first letter in each word of a string to Uppercase.
trim	Trims the leading and trailing spaces off a string.
clean	trims (String.trim) a string AND removes all the double spaces in a string.
rgbToHex	Converts an RGB value to hexadecimal. The string must be in the format of "rgb(255,255,255)" or "rgba(255,255,255,1)";
hexToRgb	Converts a hexadecimal color value to RGB. Input string must be the hex color value (with or without the hash). Also accepts triplets ('333');
contains	checks if the passed in string is contained in the String. also accepts an optional second parameter, to check if the string is contained in a list of separated values.
escapeRegExp	Returns string with escaped regular expression characters
rgbToHex	see String.rgbToHex, but as an array method.
hexToRgb	same as String.hexToRgb, but as an array method.

Class String

A collection of The String Object prototype methods.

Method test

Tests a string with a regular expression.

Arguments:

`regex` a string or regular expression object, the regular expression you want to match the string with

`params` optional, if first parameter is a string, any parameters you want to pass to the `regex` ('g' has no effect)

Returns:

true if a match for the regular expression is found in the string, false if not.

See <http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference:Objects:RegExp:test>

Example:

```
"I like cookies".test("cookie"); // returns true
"I like cookies".test("COOKIE", "i") // ignore case, returns true
```

```
"I like cookies".test("cake"); // returns false
```

Method `toInt`

parses a string to an integer.

Returns:

either an int or "NaN" if the string is not a number.

Example:

```
var value = "10px".toInt(); // value is 10
```

Method `toFloat`

parses a string to an float.

Returns:

either a float or "NaN" if the string is not a number.

Example:

```
var value = "10.848".toFloat(); // value is 10.848
```

Method `camelCase`

Converts a hiphenated string to a camelcase string.

Example:

```
"I-like-cookies".camelCase(); //"ILikeCookies"
```

Method `hyphenate`

Converts a camelCased string to a hyphen-ated string.

Example:

```
"ILikeCookies".hyphenate(); //"I-like-cookies"
```

Method `capitalize`

Converts the first letter in each word of a string to Uppercase.

Example:

```
"i like cookies".capitalize(); //"I Like Cookies"
```

Method trim

Trims the leading and trailing spaces off a string.

Example:

```
" i like cookies ".trim() //"i like cookies"
```

Method clean

trims (<String.trim>) a string AND removes all the double spaces in a string.

Returns:

the cleaned string

Example:

```
" i like cookies \n\n".clean() //"i like cookies"
```

Method rgbToHex

Converts an RGB value to hexadecimal. The string must be in the format of "rgb(255,255,255)" or "rgba(255,255,255,1)";

Arguments:

array boolean value, defaults to false. Use true if you want the array ['FF','33','00'] as output instead of "#FF3300"

Returns:

hex string or array. returns "transparent" if the output is set as string and the fourth value of rgba in input string is 0.

Example:

```
"rgb(17,34,51)".rgbToHex(); //"#112233"  
"rgba(17,34,51,0)".rgbToHex(); //"transparent"  
"rgb(17,34,51)".rgbToHex(true); //[ '11', '22', '33' ]
```

Method hexToRgb

Converts a hexadecimal color value to RGB. Input string must be the hex color value (with or without the hash). Also accepts triplets ('333');

Arguments:

array boolean value, defaults to false. Use true if you want the array [255,255,255] as output instead of "rgb(255,255,255)";

Returns:

rgb string or array.

Example:

```
"#112233".hexToRgb(); //"rgb(17,34,51)"  
"#112233".hexToRgb(true); //[17,34,51]
```

Method contains

checks if the passed in string is contained in the String. also accepts an optional second parameter, to check if the string is contained in a list of separated values.

Example:

```
'a b c'.contains('c', ' '); //true  
'a bc'.contains('bc'); //true  
'a bc'.contains('b', ' '); //false
```

Method escapeRegExp

Returns string with escaped regular expression characters

Example:

```
var search = 'animals.sheeps[1]'.escapeRegExp(); // search is now 'animals\.sheeps\[1\]'
```

Method rgbToHex

see <String.rgbToHex>, but as an array method.

Method hexToRgb

same as <String.hexToRgb>, but as an array method.

Function.js

Contains Function prototypes and utility functions .

License:

MIT-style license.

Summary

Function.js	Contains Function prototypes and utility functions .
Function	A collection of The Function Object prototype methods.
create	Main function to create closures.
pass	Shortcut to create closures with arguments and bind.
attempt	Tries to execute the function, returns either the result of the function or false on error.
bind	method to easily create closures with "this" altered.
bindAsEventListener	cross browser method to pass event firer
delay	Delays the execution of a function by a specified duration.
periodical	Executes a function in the specified intervals of time

Credits:

- Some functions are inspired by those found in prototype.js <<http://prototype.conio.net/>> (c) 2005
Sam Stephenson sam [at] conio [dot] net, MIT-style license

Class Function

A collection of The Function Object prototype methods.

Method create

Main function to create closures.

Returns:

a function.

Arguments:

options An Options object.

Options

bind The object that the "this" of the function will refer to. Default is the current function.

event If set to true, the function will act as an event listener and receive an event as first argument.

If set to a class name, the function will receive a new instance of this class (with the event passed as argument's constructor) as first argument.

Default is false.

arguments A single argument or array of arguments that will be passed to the function when called.

If both the event and arguments options are set, the event is passed as first argument and the arguments array will follow.

Default is no custom arguments, the function will receive the standard arguments when called.

delay - Numeric value: if set, the returned function will delay the actual execution by this amount of milliseconds and return a timer handle when called.

Default is no delay.

periodical - Numeric value: if set, the returned function will periodically perform the actual execution with this specified interval and return a timer handle when called.

Default is no periodical execution.

attempt - If set to true, the returned function will try to execute and return either the results or false on error. Default is false.

Method `pass`

Shortcut to create closures with arguments and bind.

Returns:

a function.

Arguments:

args the arguments passed. must be an array if arguments > 1
bind optional, the object that the "this" of the function will refer to.

Example:

```
myFunction.pass([arg1, arg2], myElement);
```

Method `attempt`

Tries to execute the function, returns either the result of the function or false on error.

Arguments:

args the arguments passed. must be an array if arguments > 1
bind optional, the object that the "this" of the function will refer to.

Example:

```
myFunction.attempt([arg1, arg2], myElement);
```

Method `bind`

method to easily create closures with "this" altered.

Arguments:

bind optional, the object that the "this" of the function will refer to.
args optional, the arguments passed. must be an array if arguments > 1

Returns:

a function.

Example:

```
function myFunction(){
  this.setStyle('color', 'red');
  // note that 'this' here refers to myFunction, not an element
  // we'll need to bind this function to the element we want to alter
};
var myBoundFunction = myFunction.bind(myElement);
myBoundFunction(); // this will make the element myElement red.
```

Method `bindAsEventListener`

cross browser method to pass event firer

Arguments:

`bind` optional, the object that the "this" of the function will refer to.
`args` optional, the arguments passed. must be an array if arguments > 1

Returns:

a function with the parameter bind as its "this" and as a pre-passed argument event or window.event, depending on the browser.

Example:

```
function myFunction(event){
  alert(event.clientX) //returns the coordinates of the mouse..
};
myElement.onclick = myFunction.bindAsEventListener(myElement);
```

Method `delay`

Delays the execution of a function by a specified duration.

Arguments:

`delay` the duration to wait in milliseconds.
`bind` optional, the object that the "this" of the function will refer to.
`args` optional, the arguments passed. must be an array if arguments > 1

Example:

```
myFunction.delay(50, myElement) //wait 50 milliseconds, then call myFunction and bind myElement to it
(function(){alert('one second later...')}).delay(1000); //wait a second and alert
```

Method `periodical`

Executes a function in the specified intervals of time

Arguments:

interval	the duration of the intervals between executions.
bind	optional, the object that the "this" of the function will refer to.
args	optional, the arguments passed. must be an array if arguments > 1

Number.js

Contains the Number prototypes.

License:

MIT-style license.

Summary

Number.js	Contains the Number prototypes.
Number	A collection of The Number Object prototype methods.
toInt	Returns this number; useful because toInt must work on both Strings and Numbers.
toFloat	Returns this number as a float; useful because toFloat must work on both Strings and Numbers.
limit	Limits the number.
round	Returns the number rounded to specified precision.
times	Executes a passed in function the specified number of times

Class Number

A collection of The Number Object prototype methods.

Method toInt

Returns this number; useful because toInt must work on both Strings and Numbers.

Method toFloat

Returns this number as a float; useful because toFloat must work on both Strings and Numbers.

Method limit

Limits the number.

Arguments:

min number, minimum value

max number, maximum value

Returns:

the number in the given limits.

Example:

```
(12).limit(2, 6.5) // returns 6.5
(-4).limit(2, 6.5) // returns 2
(4.3).limit(2, 6.5) // returns 4.3
```

Method round

Returns the number rounded to specified precision.

Arguments:

precision integer, number of digits after the decimal point. Can also be negative or zero (default).

Example:

```
12.45.round() // returns 12
12.45.round(1) // returns 12.5
12.45.round(-1) // returns 10
```

Method `times`

Executes a passed in function the specified number of times

Arguments:

function the function to be executed on each iteration of the loop

Example:

```
(4).times(alert);
```

Element.js

Contains useful Element prototypes, to be used with the dollar function `<$>`.

License:

MIT-style license.

Summary

Element.js	Contains useful Element prototypes, to be used with the dollar function \$.
Element	Custom class to allow all of its methods to be used with any DOM element via the dollar function \$.
initialize	Creates a new element of the type passed in.
Elements	- Every dom function such as \$\$, or in general every function that returns a collection of nodes in mootools, returns them as an Elements class.
Element	Custom class to allow all of its methods to be used with any DOM element via the dollar function \$.
set	you can set events, styles and properties with this shortcut. same as calling new Element.
injectBefore	Inserts the Element before the passed element.
injectAfter	Same as Element.injectBefore, but inserts the element after.
injectInside	Same as Element.injectBefore, but inserts the element inside.
injectTop	Same as Element.injectInside, but inserts the element inside, at the top.
adopt	Inserts the passed elements inside the Element.
remove	Removes the Element from the DOM.
clone	Clones the Element and returns the cloned one.
replaceWith	Replaces the Element with an element passed.
appendText	Appends text node to a DOM element.
hasClass	Tests the Element to see if it has the passed in className.
addClass	Adds the passed in class to the Element, if the element doesnt already have it.
removeClass	Works like Element.addClass, but removes the class from the element.
toggleClass	Adds or removes the passed in class name to the element, depending on if it's present or not.
setStyle	Sets a css property to the Element.
setStyles	Applies a collection of styles to the Element.
setOpacity	Sets the opacity of the Element, and sets also visibility == "hidden" if opacity == 0, and visibility = "visible" if opacity 0.
getStyle	Returns the style of the Element given the property passed in.
getStyles	Returns an object of styles of the Element for each argument passed in.
getPrevious	Returns the previousSibling of the Element, excluding text nodes.
getNext	Works as Element.getPrevious, but tries to find the nextSibling.
getFirst	Works as Element.getPrevious, but tries to find the firstChild.
getLast	Works as Element.getPrevious, but tries to find the lastChild.
getParent	returns the \$(element.parentNode)
getChildren	returns all the \$(element.childNodes), excluding text nodes. Returns as Elements.
hasChild	returns true if the passed in element is a child of the \$(element).
getProperty	Gets the an attribute of the Element.
removeProperty	Removes an attribute from the Element
getProperties	same as Element.getStyles, but for properties
setProperty	Sets an attribute for the Element.
setPropertyies	Sets numerous attributes for the Element.
setHTML	Sets the innerHTML of the Element.
setText	Sets the inner text of the Element.
getText	Gets the inner text of the Element.
getTag	Returns the tagName of the element in lower case.
empty	Empties an element of all its children.

Credits:

Class Element

Custom class to allow all of its methods to be used with any DOM element via the dollar function <\$>.

Method initialize

Creates a new element of the type passed in.

Arguments:

`el` string; the tag name for the element you wish to create. you can also pass in an element reference, in which case it will be extended.

`props` object; the properties you want to add to your element.

Accepts the same keys as <Element.setProperties>, but also allows events and styles

Props:

the key styles will be used as setStyles, the key events will be used as addEvents. any other key is used as setProperty.

Example:

```
new Element('a', {
  'styles': {
    'display': 'block',
    'border': '1px solid black'
  },
  'events': {
    'click': function(){
      //aaa
    },
    'mousedown': function(){
      //aaa
    }
  },
  'class': 'myClassSuperClass',
  'href': 'http://mad4milk.net'
});

(end)
```

Class Elements

- Every dom function such as <\$\$>, or in general every function that returns a collection of nodes in mootools, returns them as an Elements class.
- The purpose of the Elements class is to allow <Element> methods to work also on <Elements> array.
- Elements is also an Array, so it accepts all the <Array> methods.
- Every node of the Elements instance is already extended with <\$>.

Example:

```

$$('myselector').each(function(el){
  //...
});
$$('myselector').setStyle('color', 'red');

```

Section Utility Functions

Function \$

returns the element passed in with all the Element prototypes applied.

Arguments:

el a reference to an actual element or a string representing the id of an element

Example:

```

$('myElement') // gets a DOM element by id with all the Element prototypes applied.
var div = document.getElementById('myElement');
$(div) //returns an Element also with all the mootools extensions applied.

```

Function \$\$

Selects, and extends DOM elements. Elements arrays returned with \$\$ will also accept all the <Element> methods. The return type of element methods run through \$\$ is always an array. If the return array is only made by elements, \$\$ will be applied automatically.

Arguments:

HTML Collections, arrays of elements, arrays of strings as element ids, elements, strings as selectors. Any number of the above as arguments are accepted.

Note:

if you load <Element.Selectors.js>, \$\$ will also accept CSS Selectors, otherwise the only selectors supported are tag names.

Example:

```

$$('a') //an array of all anchor tags on the page
$$('a', 'b') //an array of all anchor and bold tags on the page
$$('#myElement') //array containing only the element with id = myElement. (only with
<Element.Selectors.js>)
$$('#myElement a.myClass') //an array of all anchor tags with the class "myClass"
//within the DOM element with id "myElement" (only with <Element.Selectors.js>)
$$(myelement, myelement2, 'a', ['myid', myid2, 'myid3'], document.getElementsByTagName('div')) //an
array containing:
// the element referenced as myelement if existing,
// the element referenced as myelement2 if existing,
// all the elements with a as tag in the page,

```

```
// the element with id = myid if existing
// the element with id = myid2 if existing
// the element with id = myid3 if existing
// all the elements with div as tag in the page
```

Class Element

Custom class to allow all of its methods to be used with any DOM element via the dollar function `<$>`.

Method set

you can set events, styles and properties with this shortcut. same as calling new Element.

Method injectBefore

Inserts the Element before the passed element.

Arguments:

`el` an element reference or the id of the element to be injected in.

Example:

```
html:
<div id="myElement"></div>
<div id="mySecondElement"></div>
js:
$('mySecondElement').injectBefore('myElement');
resulting html:
<div id="mySecondElement"></div>
<div id="myElement"></div>
```

Method injectAfter

Same as `<Element.injectBefore>`, but inserts the element after.

Method injectInside

Same as `<Element.injectBefore>`, but inserts the element inside.

Method injectTop

Same as `<Element.injectInside>`, but inserts the element inside, at the top.

Method adopt

Inserts the passed elements inside the Element.

Arguments:

accepts elements references, element ids as string, selectors (`$$('stuff')`) / array of elements, array of ids as strings and collections.

Method `remove`

Removes the Element from the DOM.

Example:

```
$('#myElement').remove() //bye bye
```

Method `clone`

Clones the Element and returns the cloned one.

Arguments:

`contents` `boolean`, when true the Element is cloned with `childNodes`, default true

Returns:

the cloned element

Example:

```
var clone = $('#myElement').clone().injectAfter('myElement');  
//clones the Element and append the clone after the Element.
```

Method `replaceWith`

Replaces the Element with an element passed.

Arguments:

`el` a string representing the element to be injected in (`myElementId`, or `div`), or an element reference.

If you pass `div` or another tag, the element will be created.

Returns:

the passed in element

Example:

```
$('#myOldElement').replaceWith($('#myNewElement')); //$('#myOldElement') is gone, and  
$('#myNewElement') is in its place.
```

Method `appendText`

Appends text node to a DOM element.

Arguments:

text the text to append.

Example:

```
<div id="myElement">hey</div>
$('myElement').appendText(' howdy'); //myElement innerHTML is now "hey howdy"
```

Method `hasClass`

Tests the Element to see if it has the passed in className.

Returns:

true - the Element has the class

false - it doesn't

Arguments:

className string; the class name to test.

Example:

```
<div id="myElement" class="testClass"></div>
$('myElement').hasClass('testClass'); //returns true
```

Method `addClass`

Adds the passed in class to the Element, if the element doesnt already have it.

Arguments:

className string; the class name to add

Example:

```
<div id="myElement" class="testClass"></div>
$('myElement').addClass('newClass'); //<div id="myElement" class="testClass newClass"></div>
```

Method `removeClass`

Works like `<Element.addClass>`, but removes the class from the element.

Method `toggleClass`

Adds or removes the passed in class name to the element, depending on if it's present or not.

Arguments:

className the class to add or remove

Example:

```
<div id="myElement" class="myClass"></div>
$('myElement').toggleClass('myClass');
<div id="myElement" class=""></div>
$('myElement').toggleClass('myClass');
<div id="myElement" class="myClass"></div>
```

Method `setStyle`

Sets a css property to the Element.

Arguments:

property the property to set
value the value to which to set it; for numeric values that require "px" you can pass an integer

Example:

```
$('myElement').setStyle('width', '300px'); //the width is now 300px
$('myElement').setStyle('width', 300); //the width is now 300px
```

Method `setStyles`

Applies a collection of styles to the Element.

Arguments:

source an object or string containing all the styles to apply. When its a string it overrides old style.

Examples:

```
$('myElement').setStyles({
  border: '1px solid #000',
  width: 300,
  height: 400
});
$('myElement').setStyles('border: 1px solid #000; width: 300px; height: 400px;');
```

Method `setOpacity`

Sets the opacity of the Element, and sets also visibility == "hidden" if opacity == 0, and visibility = "visible" if opacity > 0.

Arguments:

opacity float; Accepts values from 0 to 1.

Example:

```
$('#myElement').setOpacity(0.5) //make it 50% transparent
```

Method `getStyle`

Returns the style of the Element given the property passed in.

Arguments:

property the css style property you want to retrieve

Example:

```
$('#myElement').getStyle('width'); //returns "400px"  
//but you can also use  
$('#myElement').getStyle('width').toInt(); //returns 400
```

Method `getStyles`

Returns an object of styles of the Element for each argument passed in.

Arguments:

properties strings; any number of style properties

Example:

```
$('#myElement').getStyles('width', 'height', 'padding');  
//returns an object like:  
{width: "10px", height: "10px", padding: "10px 0px 10px 0px"}
```

Method `getPrevious`

Returns the previousSibling of the Element, excluding text nodes.

Example:

```
$('#myElement').getPrevious(); //get the previous DOM element from myElement
```

Method `getNext`

Works as Element.getPrevious, but tries to find the nextSibling.

Method `getFirst`

Works as <Element.getPrevious>, but tries to find the firstChild.

Method `getLast`

Works as `<Element.getPrevious>`, but tries to find the `lastChild`.

Method `getParent`

returns the `$(element.parentNode)`

Method `getChildren`

returns all the `$(element.childNodes)`, excluding text nodes. Returns as `<Elements>`.

Method `hasChild`

returns true if the passed in element is a child of the `$(element)`.

Method `getMethod`

Gets the an attribute of the Element.

Arguments:

`property` `string`; the attribute to retrieve

Example:

```
$('#myImage').getProperty('src') // returns whatever.gif
```

Method `removeMethod`

Removes an attribute from the Element

Arguments:

`property` `string`; the attribute to remove

Method `getProperties`

same as `<Element.getStyles>`, but for properties

Method `setMethod`

Sets an attribute for the Element.

Arguments:

`property` `string`; the property to assign the value passed in
`value` the value to assign to the property passed in

Example:

```
$('#myImage').setProperty('src', 'whatever.gif'); //myImage now points to whatever.gif for its source
```

Method `setProperty`

Sets numerous attributes for the Element.

Arguments:

source an object with key/value pairs.

Example:

```
$('#myElement').setProperty({
src: 'whatever.gif',
alt: 'whatever dude'
});

(end)
```

Method `setHTML`

Sets the innerHTML of the Element.

Arguments:

html string; the new innerHTML for the element.

Example:

```
$('#myElement').setHTML(newHTML) //the innerHTML of myElement is now = newHTML
```

Method `setText`

Sets the inner text of the Element.

Arguments:

text string; the new text content for the element.

Example:

```
$('#myElement').setText('some text') //the text of myElement is now = 'some text'
```

Method `getText`

Gets the inner text of the Element.

Method `getTag`

Returns the tagName of the element in lower case.

Example:

```
$('#myImage').getTag() // returns 'img'
```

Method `empty`

Empties an element of all its children.

Example:

```
$('#myDiv').empty() // empties the Div and returns it
```

Element.Event.js

Contains the Event Class, Element methods to deal with Element events, custom Events, and the Function prototype `bindWithEvent`.

License:

MIT-style license.

Summary

Element.Event.js	Contains the Event Class, Element methods to deal with Element events, custom Events, and the Function prototype <code>bindWithEvent</code> .
Event	Cross browser methods to manage events.
stop	cross browser method to stop an event
stopPropagation	cross browser method to stop the propagation of an event
preventDefault	cross browser method to prevent the default action of the event
keys	you can add additional Event keys codes this way:
Element	Custom class to allow all of its methods to be used with any DOM element via the dollar function <code>\$</code> .
addEvent	Attaches an event listener to a DOM element.
removeEvent	Works as <code>Element.addEvent</code> , but instead removes the previously added event listener.
addEvents	As <code>addEvent</code> , but accepts an object and add multiple events at once.
removeEvents	removes all events of a certain type from an element. if no argument is passed in, removes all events.
fireEvent	executes all events of the specified type present in the element.
cloneEvents	Clones all events from an element to this element.
Function	A collection of The Function Object prototype methods.
bindWithEvent	automatically passes MooTools Event Class.

Class Event

Cross browser methods to manage events.

Arguments:

`event` the event

Properties

<code>shift</code>	true if the user pressed the shift
<code>control</code>	true if the user pressed the control
<code>alt</code>	true if the user pressed the alt
<code>meta</code>	true if the user pressed the meta key
<code>wheel</code>	the amount of third button scrolling
<code>code</code>	the keycode of the key pressed
<code>page.x</code>	the x position of the mouse, relative to the full window
<code>page.y</code>	the y position of the mouse, relative to the full window
<code>client.x</code>	the x position of the mouse, relative to the viewport
<code>client.y</code>	the y position of the mouse, relative to the viewport
<code>key</code>	the key pressed as a lowercase string. <code>key</code> also returns 'enter', 'up', 'down', 'left', 'right', 'space', 'backspace', 'delete', 'esc'. Handy for these special keys.
<code>target</code>	the event target
<code>relatedTarget</code>	the event related target

Example:

```
$('#myLink').onkeydown = function(event){  
  var event = new Event(event);  
  //event is now the Event class.  
  alert(event.key); //returns the lowercase letter pressed  
  alert(event.shift); //returns true if the key pressed is shift  
  if (event.key == 's' && event.control) alert('document saved');  
};  
(end)
```

Method **stop**

cross browser method to stop an event

Method **stopPropagation**

cross browser method to stop the propagation of an event

Method **preventDefault**

cross browser method to prevent the default action of the event

Method **keys**

you can add additional Event keys codes this way:

Example:

```
Event.keys.whatever = 80;  
$(myelement).addEvent(keydown, function(event){  
  event = new Event(event);  
  if (event.key == 'whatever') console.log(whatever key clicked).  
});  
(end)
```

Class **Element**

Custom class to allow all of its methods to be used with any DOM element via the dollar function <\$>.

Method **addEvent**

Attaches an event listener to a DOM element.

Arguments:

type the event to monitor ('click', 'load', etc) without the prefix 'on'.
fn the function to execute

Example:

```
$('#myElement').addEvent('click', function(){alert('clicked!')});
```

Method `removeEvent`

Works as `Element.addEvent`, but instead removes the previously added event listener.

Method `addEvents`

As `<addEvent>`, but accepts an object and add multiple events at once.

Method `removeEvents`

removes all events of a certain type from an element. if no argument is passed in, removes all events.

Arguments:

type string; the event name (e.g. 'click')

Method `fireEvent`

executes all events of the specified type present in the element.

Arguments:

type string; the event name (e.g. 'click')

args array or single object; arguments to pass to the function; if more than one argument, must be an array

delay (integer) delay (in ms) to wait to execute the event

Method `cloneEvents`

Clones all events from an element to this element.

Arguments:

from element, copy all events from this element

type optional, copies only events of this type

Event: `mouseenter`

In addition to the standard javascript events (load, mouseover, mouseout, click, etc.) `<Event.js>` contains two custom events this event fires when the mouse enters the area of the dom element; will not be fired again if the mouse crosses over children of the element (unlike `mouseover`)

Example:

```
$(myElement).addEvent('mouseenter', myFunction);
```

Event: `mouseleave`

this event fires when the mouse exits the area of the dom element; will not be fired again if the mouse crosses over children of the element (unlike mouseout)

Example:

```
$(myElement).addEvent('mouseleave', myFunction);
```

Class Function

A collection of The Function Object prototype methods.

Method bindWithEvent

automatically passes MooTools Event Class.

Arguments:

bind optional, the object that the "this" of the function will refer to.
args optional, an argument to pass to the function; if more than one argument, it must be an array of arguments.

Returns:

a function with the parameter bind as its "this" and as a pre-passed argument event or window.event, depending on the browser.

Example:

```
function myFunction(event){  
  alert(event.client.x) //returns the coordinates of the mouse..  
};  
myElement.addEvent('click', myFunction.bindWithEvent(myElement));
```

Element.Filters.js

add Filters capability to <Elements>.

License:

MIT-style license.

Summary

Element.Filters.js	add Filters capability to Elements.
Elements	A collection of methods to be used with \$\$ elements collections.
filterByTag	Filters the collection by a specified tag name.
filterByClass	Filters the collection by a specified class name.
filterById	Filters the collection by a specified ID.
filterByAttribute	Filters the collection by a specified attribute.

Class Elements

A collection of methods to be used with <\$\$> elements collections.

Method filterByTag

Filters the collection by a specified tag name.

Returns a new Elements collection, while the original remains untouched.

Method filterByClass

Filters the collection by a specified class name.

Returns a new Elements collection, while the original remains untouched.

Method filterById

Filters the collection by a specified ID.

Returns a new Elements collection, while the original remains untouched.

Method filterByAttribute

Filters the collection by a specified attribute.

Returns a new Elements collection, while the original remains untouched.

Arguments:

name the attribute name.

operator optional, the attribute operator.

value optional, the attribute value, only valid if the operator is specified.

Element.Selectors.js

Css Query related functions and <Element> extensions

License:

MIT-style license.

Summary

Element.Selectors.js	Css Query related functions and Element extensions
Element	Custom class to allow all of its methods to be used with any DOM element via the dollar function \$.
getElements	Gets all the elements within an element that match the given (single) selector.
getElement	Same as Element.getElements, but returns only the first. Alternate syntax for \$E, where filter is the Element.
getElementsBySelector	Same as Element.getElements, but allows for comma separated selectors, as in css. Alternate syntax for \$\$, where filter is the Element.
getElementById	Targets an element with the specified id found inside the Element. Does not overwrite document.getElementById.

Function `$E`

Selects a single (i.e. the first found) Element based on the selector passed in and an optional filter element.

Returns as <Element>.

Arguments:

`selector` `string`; the css selector to match

`filter` `optional`; a DOM element to limit the scope of the selector match; defaults to document.

Example:

```
$E('a', 'myElement') //find the first anchor tag inside the DOM element with id 'myElement'
```

Function `$$ES`

Returns a collection of Elements that match the selector passed in limited to the scope of the optional filter.

See Also: `<Element.getElements>` for an alternate syntax.

Returns as <Elements>.

Returns:

an array of dom elements that match the selector within the filter

Arguments:

`selector` `string`; css selector to match

`filter` `optional`; a DOM element to limit the scope of the selector match; defaults to document.

Examples:

```
$$ES("a") //gets all the anchor tags; synonymous with $$("a")
```

```
$ES('a','myElement') //get all the anchor tags within $('myElement')
```

Class Element

Custom class to allow all of its methods to be used with any DOM element via the dollar function `<$>`.

Method `getElements`

Gets all the elements within an element that match the given (single) selector.
Returns as `<Elements>`.

Arguments:

`selector` `string`; the css selector to match

Examples:

```
$('myElement').getElements('a'); // get all anchors within myElement
$('myElement').getElements('input[name=dialog]') //get all input tags with name 'dialog'
$('myElement').getElements('input[name$=log]') //get all input tags with names ending with 'log'
```

Method `getElement`

Same as `<Element.getElements>`, but returns only the first. Alternate syntax for `<$E>`, where `filter` is the `Element`.
Returns as `<Element>`.

Arguments:

`selector` `string`; css selector

Method `getElementsBySelector`

Same as `<Element.getElements>`, but allows for comma separated selectors, as in css. Alternate syntax for `<$$>`, where `filter` is the `Element`.
Returns as `<Elements>`.

Arguments:

`selector` `string`; css selector

Method `getElementById`

Targets an element with the specified id found inside the `Element`. Does not overwrite `document.getElementById`.

Arguments:

`id` `string`; the id of the element to find.

Element.Form.js

Contains Element prototypes to deal with Forms and their elements.

License:

MIT-style license.

Summary

Element.Form.js	Contains Element prototypes to deal with Forms and their elements.
Element	Custom class to allow all of its methods to be used with any DOM element via the dollar function \$.
getValue	Returns the value of the Element, if its tag is textarea, select or input. getValue called on a multiple select will return an array.
toQueryString	Reads the children inputs of the Element and generates a query string, based on their values. Used internally in Ajax

Class Element

Custom class to allow all of its methods to be used with any DOM element via the dollar function <\$>.

Method getValue

Returns the value of the Element, if its tag is textarea, select or input. getValue called on a multiple select will return an array.

Method toQueryString

Reads the children inputs of the Element and generates a query string, based on their values. Used internally in <Ajax>

Example:

```
<form id="myForm" action="submit.php">
<input name="email" value="bob@bob.com">
<input name="zipCode" value="90210">
</form>

<script>
$('myForm').toQueryString()
</script>
(end)
```

Returns:

email=bob@bob.com&zipCode=90210

Element.Dimensions.js

Contains Element prototypes to deal with Element size and position in space.

Note:

The functions in this script require n XHTML doctype.

License:

MIT-style license.

Summary

Element.Dimensions.js	Contains Element prototypes to deal with Element size and position in space.
Element	Custom class to allow all of its methods to be used with any DOM element via the dollar function \$.
scrollTo	Scrolls the element to the specified coordinated (if the element has an overflow)
getSize	Return an Object representing the size/scroll values of the element.
getPosition	Returns the real offsets of the element.
getTop	Returns the distance from the top of the window to the Element.
getLeft	Returns the distance from the left of the window to the Element.
getCoordinates	Returns an object with width, height, left, right, top, and bottom, representing the values of the Element

Class Element

Custom class to allow all of its methods to be used with any DOM element via the dollar function <\$>.

Method scrollTo

Scrolls the element to the specified coordinated (if the element has an overflow)

Arguments:

- x the x coordinate
- y the y coordinate

Example:

```
$( 'myElement' ).scrollTo( 0, 100 )
```

Method getSize

Return an Object representing the size/scroll values of the element.

Example:

```
$( 'myElement' ).getSize();  
(end)
```

Returns:

(start code)

```
{
```

```
'scroll': {'x': 100, 'y': 100},
'size': {'x': 200, 'y': 400},
'scrollSize': {'x': 300, 'y': 500}
}
(end)
```

Method `getPosition`

Returns the real offsets of the element.

Arguments:

`overflown` optional, an array of nested scrolling containers for scroll offset calculation, use this if your element is inside any element containing scrollbars

Example:

```
$('#element').getPosition();
{x: 100, y:500};
```

Method `getTop`

Returns the distance from the top of the window to the Element.

Arguments:

`overflown` optional, an array of nested scrolling containers, see `Element::getPosition`

Method `getLeft`

Returns the distance from the left of the window to the Element.

Arguments:

`overflown` optional, an array of nested scrolling containers, see `Element::getPosition`

Method `getCoordinates`

Returns an object with width, height, left, right, top, and bottom, representing the values of the Element

Arguments:

`overflown` optional, an array of nested scrolling containers, see `Element::getPosition`

Example:

```
var myValues = $('#myElement').getCoordinates();
(end)
```

Returns:

```
(start code)
{
width: 200,
height: 300,
left: 100,
top: 50,
right: 300,
bottom: 350
}
(end)
```

Window.DomReady.js

Contains the custom event domready, for window.

License:

MIT-style license.

Summary

Window.DomReady.js Contains the custom event domready, for window.

Event: domready

executes a function when the dom tree is loaded, without waiting for images. Only works when called from window.

Credits:

(c) Dean Edwards/Matthias Miller/John Resig, remastered for MooTools.

Arguments:

fn the function to execute when the DOM is ready

Example:

```
window.addEvent('domready', function(){
  alert('the dom is ready');
});
```

Window.Size.js

Window cross-browser dimensions methods.

Note:

The Functions in this script require an XHTML doctype.

License:

MIT-style license.

Summary

Window.Size.js	Window cross-browser dimensions methods.
window	Cross browser methods to get various window dimensions.
getWidth	Returns an integer representing the width of the browser window (without the scrollbar).
getHeight	Returns an integer representing the height of the browser window (without the scrollbar).
getScrollWidth	Returns an integer representing the scrollWidth of the window.
getScrollHeight	Returns an integer representing the scrollHeight of the window.
getScrollLeft	Returns an integer representing the scrollLeft of the window (the number of pixels the window has scrolled from the left).
getScrollTop	Returns an integer representing the scrollTop of the window (the number of pixels the window has scrolled from the top).
getSize	Same as Element.getSize

Class `window`

Cross browser methods to get various window dimensions.

Warning: All these methods require that the browser operates in strict mode, not quirks mode.

Method `getWidth`

Returns an integer representing the width of the browser window (without the scrollbar).

Method `getHeight`

Returns an integer representing the height of the browser window (without the scrollbar).

Method `getScrollWidth`

Returns an integer representing the scrollWidth of the window.

This value is equal to or bigger than `<getWidth>`.

See Also:

<http://developer.mozilla.org/en/docs/DOM:element.scrollWidth>

Method `getScrollHeight`

Returns an integer representing the scrollHeight of the window.

This value is equal to or bigger than `<getHeight>`.

See Also:

<http://developer.mozilla.org/en/docs/DOM:element.scrollHeight>

Method `getScrollLeft`

Returns an integer representing the `scrollLeft` of the window (the number of pixels the window has scrolled from the left).

See Also:

[<http://developer.mozilla.org/en/docs/DOM:element.scrollLeft>](http://developer.mozilla.org/en/docs/DOM:element.scrollLeft)

Method `scrollTop`

Returns an integer representing the `scrollTop` of the window (the number of pixels the window has scrolled from the top).

See Also:

[<http://developer.mozilla.org/en/docs/DOM:element.scrollTop>](http://developer.mozilla.org/en/docs/DOM:element.scrollTop)

Method `getSize`

Same as `<Element.getSize>`

Fx.Base.js

Contains <Fx.Base>, the fundamentals of the MooTools Effects.

License:

MIT-style license.

Summary

Fx.Base.js	Contains Fx.Base, the fundamentals of the MooTools Effects.
Fx.Base	Base class for the Effects.
set	Immediately sets the value with no transition.
start	Executes an effect from one position to the other.
stop	Stops the transition.

Class Fx.Base

Base class for the Effects.

Options

transition	the equation to use for the effect see <Fx.Transitions>; default is <Fx.Transitions.Sine.easeInOut>
duration	the duration of the effect in ms; 500 is the default.
unit	the unit is 'px' by default (other values include things like 'em' for fonts or '%').
wait	boolean: to wait or not to wait for a current transition to end before running another of the same instance. defaults to true.
fps	the frames per second for the transition; default is 50

Events:

onStart	the function to execute as the effect begins; nothing (<Class.empty>) by default.
onComplete	the function to execute after the effect has processed; nothing (<Class.empty>) by default.
onCancel	the function to execute when you manually stop the effect.

Method set

Immediately sets the value with no transition.

Arguments:

to the point to jump to

Example:

```
var myFx = new Fx.Style('myElement', 'opacity').set(0); //will make it immediately transparent
```

Method **start**

Executes an effect from one position to the other.

Arguments:

from integer: starting value

to integer: the ending value

Examples:

```
var myFx = new Fx.Style('myElement', 'opacity').start(0,1); //display a transition from transparent to opaque.
```

Method **stop**

Stops the transition.

Fx.CSS.js

Css parsing class for effects. Required by <Fx.Style>, <Fx.Styles>, <Fx.Elements>. No documentation needed, as its used internally.

License:

MIT-style license.

Summary

Fx.CSS.js | Css parsing class for effects. Required by Fx.Style, Fx.Styles, Fx.Elements. No documentation needed, as its used internally.

Fx.Style.js

Contains <Fx.Style>

License:

MIT-style license.

Summary

Fx.Style.js	Contains Fx.Style
Fx.Style	The Style effect, used to transition any css property from one value to another. Includes colors.
hide	Same as Fx.Base.set (0); hides the element immediately without transition.
set	Sets the element's css property (specified at instantiation) to the specified value immediately.
start	Displays the transition to the value/values passed in
Element	Custom class to allow all of its methods to be used with any DOM element via the dollar function \$.
effect	Applies an Fx.Style to the Element; This a shortcut for Fx.Style.

Class Fx.Style

The Style effect, used to transition any css property from one value to another. Includes colors.

Colors must be in hex format.

Inherits methods, properties, options and events from <Fx.Base>.

Arguments:

el the \$(element) to apply the style transition to
property the property to transition
options the Fx.Base options (see: <Fx.Base>)

Example:

```
var marginChange = new Fx.Style('myElement', 'margin-top', {duration:500});  
marginChange.start(10, 100);
```

Method hide

Same as <Fx.Base.set> (0); hides the element immediately without transition.

Method set

Sets the element's css property (specified at instantiation) to the specified value immediately.

Example:

```
var marginChange = new Fx.Style('myElement', 'margin-top', {duration:500});  
marginChange.set(10); //margin-top is set to 10px immediately  
(end)
```

Method start

Displays the transition to the value/values passed in

Arguments:

from (integer; optional) the starting position for the transition
to (integer) the ending position for the transition

Note:

If you provide only one argument, the transition will use the current css value for its starting value.

Example:

```
var marginChange = new Fx.Style('myElement', 'margin-top', {duration:500});  
marginChange.start(10); //tries to read current margin top value and goes from current to 10  
(end)
```

Class Element

Custom class to allow all of its methods to be used with any DOM element via the dollar function <\$>.

Method effect

Applies an <Fx.Style> to the Element; This a shortcut for <Fx.Style>.

Arguments:

property (string) the css property to alter
options (object; optional) key/value set of options (see <Fx.Style>)

Example:

```
var myEffect = $('myElement').effect('height', {duration: 1000, transition:  
Fx.Transitions.linear});  
myEffect.start(10, 100);  
//OR  
$('myElement').effect('height', {duration: 1000, transition: Fx.Transitions.linear}).start(10,100);
```

Fx.Styles.js

Contains <Fx.Styles>

License:

MIT-style license.

Summary

Fx.Styles.js	Contains Fx.Styles
Fx.Styles	Allows you to animate multiple css properties at once;
start	Executes a transition for any number of css properties in tandem.
Element	Custom class to allow all of its methods to be used with any DOM element via the dollar function \$.
effects	Applies an Fx.Styles to the Element; This a shortcut for Fx.Styles.

Class Fx.Styles

Allows you to animate multiple css properties at once;

Colors must be in hex format.

Inherits methods, properties, options and events from <Fx.Base>.

Arguments:

el the \$(element) to apply the styles transition to

options the fx options (see: <Fx.Base>)

Example:

```
var myEffects = new Fx.Styles('myElement', {duration: 1000, transition: Fx.Transitions.linear});

//height from 10 to 100 and width from 900 to 300
myEffects.start({
  'height': [10, 100],
  'width': [900, 300]
});

//or height from current height to 100 and width from current width to 300
myEffects.start({
  'height': 100,
  'width': 300
});
(end)
```

Method start

Executes a transition for any number of css properties in tandem.

Arguments:

obj an object containing keys that specify css properties to alter and values that specify either the from/to values (as an array) or just the end value (an integer).

Example:



Class Element

Custom class to allow all of its methods to be used with any DOM element via the dollar function <\$>.

Method effects

Applies an <Fx.Styles> to the Element; This a shortcut for <Fx.Styles>.

Example:

```
var myEffects = $(myElement).effects({duration: 1000, transition: Fx.Transitions.Sine.easeInOut});
myEffects.start({'height': [10, 100], 'width': [900, 300]});
```

Fx.Elements.js

Contains <Fx.Elements>

License:

MIT-style license.

Summary

Fx.Elements.js	Contains Fx.Elements
Fx.Elements	Fx.Elements allows you to apply any number of styles transitions to a selection of elements. Includes colors (must be in hex format).
start	Applies the passed in style transitions to each object named (see example). Each item in the collection is referred to as a numerical string ("1" for instance). The first item is "0", the second "1", etc.

Class Fx.Elements

Fx.Elements allows you to apply any number of styles transitions to a selection of elements. Includes colors (must be in hex format). Inherits methods, properties, options and events from <Fx.Base>.

Arguments:

elements a collection of elements the effects will be applied to.
options same as <Fx.Base> options.

Method start

Applies the passed in style transitions to each object named (see example). Each item in the collection is referred to as a numerical string ("1" for instance). The first item is "0", the second "1", etc.

Example:

```
var myElementsEffects = new Fx.Elements($$('a'));
myElementsEffects.start({
  '0': { //let's change the first element's opacity and width
    'opacity': [0,1],
    'width': [100,200]
  },
  '4': { //and the fifth one's opacity
    'opacity': [0.2, 0.5]
  }
});
(end)
```

Fx.Scroll.js

Contains <Fx.Scroll>

License:

MIT-style license.

Summary

Fx.Scroll.js	Contains Fx.Scroll
Fx.Scroll	Scroll any element with an overflow, including the window element.
scrollTo	Scrolls the chosen element to the x/y coordinates.
toTop	Scrolls the chosen element to its maximum top.
toBottom	Scrolls the chosen element to its maximum bottom.
toLeft	Scrolls the chosen element to its maximum left.
toRight	Scrolls the chosen element to its maximum right.
toElement	Scrolls the specified element to the position the passed in element is found.

Class Fx.Scroll

Scroll any element with an overflow, including the window element.

Inherits methods, properties, options and events from <Fx.Base>.

Note:

Fx.Scroll requires an XHTML doctype.

Arguments:

element the element to scroll
options optional, see Options below.

Options

all the Fx.Base options and events, plus:

offset the distance for the scrollTo point/element. an Object with x/y properties.
overflown an array of nested scrolling containers, see <Element.getPosition>

Method scrollTo

Scrolls the chosen element to the x/y coordinates.

Arguments:

x the x coordinate to scroll the element to
y the y coordinate to scroll the element to

Method toTop

Scrolls the chosen element to its maximum top.

Method `toBottom`

Scrolls the chosen element to its maximum bottom.

Method `ToLeft`

Scrolls the chosen element to its maximum left.

Method `toRight`

Scrolls the chosen element to its maximum right.

Method `toElement`

Scrolls the specified element to the position the passed in element is found.

Arguments:

`e1` the `$(element)` to scroll the window to

Fx.Slide.js

Contains <Fx.Slide>

License:

MIT-style license.

Summary

Fx.Slide.js	Contains Fx.Slide
Fx.Slide	The slide effect; slides an element in horizontally or vertically, the contents will fold inside.
slideIn	Slides the elements in view horizontally or vertically.
slideOut	Sides the elements out of view horizontally or vertically.
hide	Hides the element without a transition.
show	Shows the element without a transition.
toggle	Slides in or Out the element, depending on its state

Class Fx.Slide

The slide effect; slides an element in horizontally or vertically, the contents will fold inside.

Inherits methods, properties, options and events from <Fx.Base>.

Note:

Fx.Slide requires an XHTML doctype.

Options

mode set it to vertical or horizontal. Defaults to vertical.
options all the <Fx.Base> options

Example:

```
var mySlider = new Fx.Slide('myElement', {duration: 500});  
mySlider.toggle() //toggle the slider up and down.  
(end)
```

Method slideIn

Slides the elements in view horizontally or vertically.

Arguments:

mode (optional, string) 'horizontal' or 'vertical'; defaults to options.mode.

Method slideOut

Sides the elements out of view horizontally or vertically.

Arguments:

mode (optional, string) 'horizontal' or 'vertical'; defaults to options.mode.

Method hide

Hides the element without a transition.

Arguments:

mode (optional, string) 'horizontal' or 'vertical'; defaults to options.mode.

Method show

Shows the element without a transition.

Arguments:

mode (optional, string) 'horizontal' or 'vertical'; defaults to options.mode.

Method toggle

Slides in or Out the element, depending on its state

Arguments:

mode (optional, string) 'horizontal' or 'vertical'; defaults to options.mode.

Fx.Transitions.js

Effects transitions, to be used with all the effects.

License:

MIT-style license.

Summary

Fx.Transitions.js	Effects transitions, to be used with all the effects.
Fx.Transitions	A collection of tweening transitions for use with the Fx.Base classes.
linear	displays a linear transition.
Quad	displays a quadratic transition. Must be used as Quad.easeIn or Quad.easeOut or Quad.easeInOut
Cubic	displays a cubicular transition. Must be used as Cubic.easeIn or Cubic.easeOut or Cubic.easeInOut
Quart	displays a quartetic transition. Must be used as Quart.easeIn or Quart.easeOut or Quart.easeInOut
Quint	displays a quintic transition. Must be used as Quint.easeIn or Quint.easeOut or Quint.easeInOut
Pow	Used to generate Quad, Cubic, Quart and Quint.
Expo	displays a exponential transition. Must be used as Expo.easeIn or Expo.easeOut or Expo.easeInOut
Circ	displays a circular transition. Must be used as Circ.easeIn or Circ.easeOut or Circ.easeInOut
Sine	displays a sineoidal transition. Must be used as Sine.easeIn or Sine.easeOut or Sine.easeInOut
Back	makes the transition go back, then all forth. Must be used as Back.easeIn or Back.easeOut or Back.easeInOut
Bounce	makes the transition bouncy. Must be used as Bounce.easeIn or Bounce.easeOut or Bounce.easeInOut
Elastic	Elastic curve. Must be used as Elastic.easeIn or Elastic.easeOut or Elastic.easeInOut

Credits:

Easing Equations by Robert Penner, <<http://www.robertpenner.com/easing/>>, modified & optimized to be used with mootools.

Class Fx.Transitions

A collection of tweening transitions for use with the <Fx.Base> classes.

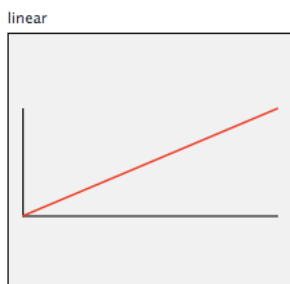
Example:

```
//Elastic.easeOut with default values:  
new Fx.Style('margin', {transition: Fx.Transitions.Elastic.easeOut});  
//Elastic.easeOut with user-defined value for elasticity.  
var myTransition = new Fx.Transition(Fx.Transitions.Elastic, 3);  
new Fx.Style('margin', {transition: myTransition.easeOut});
```

Method linear

displays a linear transition.

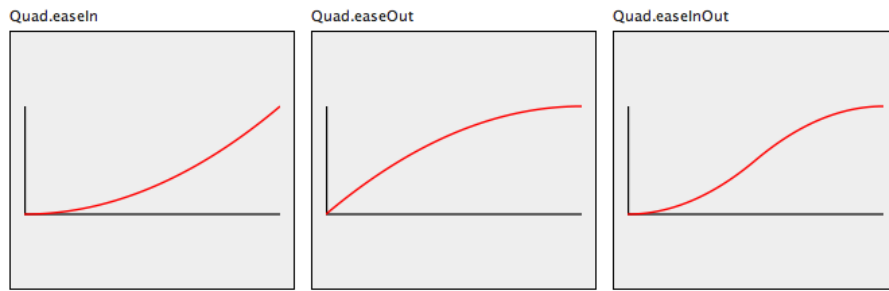
Graph:



Method Quad

displays a quadratic transition. Must be used as `Quad.easeIn` or `Quad.easeOut` or `Quad.easeInOut`

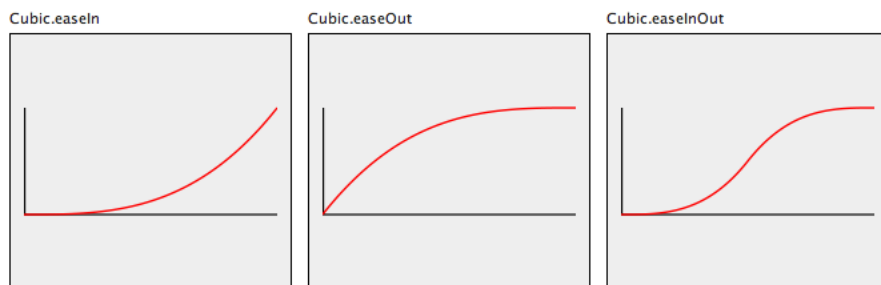
Graph:



Method Cubic

displays a cubicular transition. Must be used as `Cubic.easeIn` or `Cubic.easeOut` or `Cubic.easeInOut`

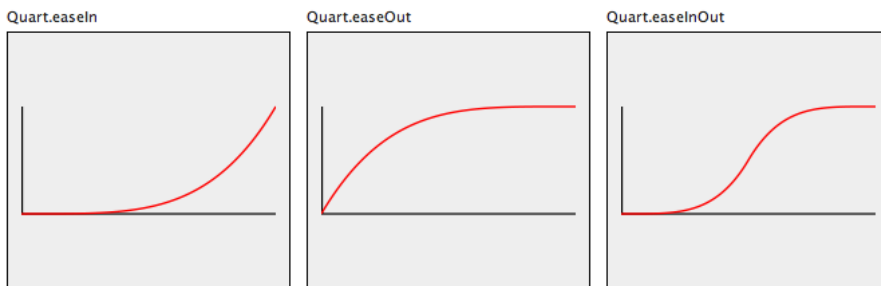
Graph:



Method Quart

displays a quartetic transition. Must be used as `Quart.easeIn` or `Quart.easeOut` or `Quart.easeInOut`

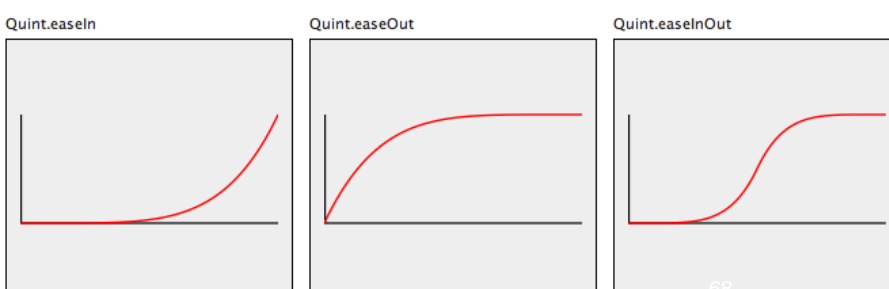
Graph:



Method Quint

displays a quintic transition. Must be used as `Quint.easeIn` or `Quint.easeOut` or `Quint.easeInOut`

Graph:



Method **Pow**

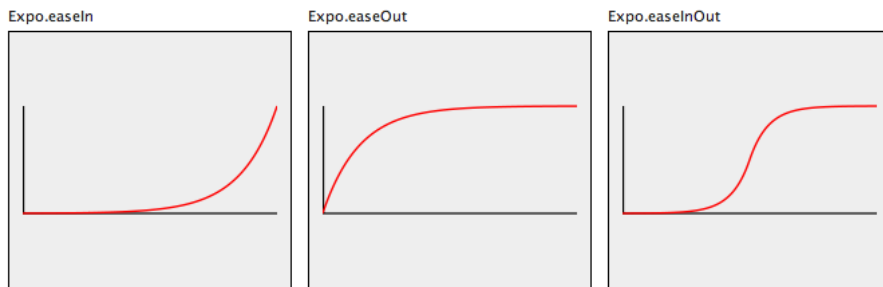
Used to generate Quad, Cubic, Quart and Quint.
By default is p^6 .

Graph:

Method **Expo**

displays a exponential transition. Must be used as Expo.easeIn or Expo.easeOut or Expo.easeInOut

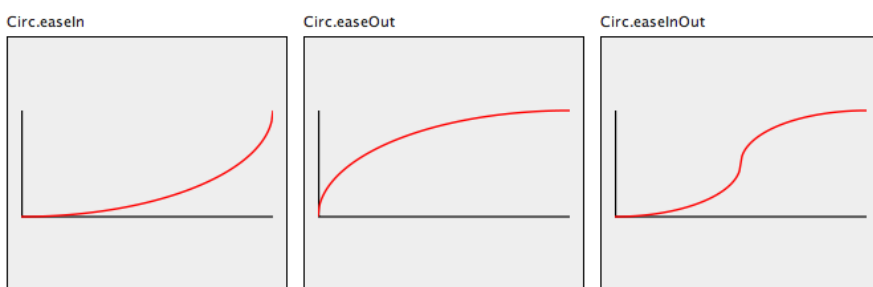
Graph:



Method **Circ**

displays a circular transition. Must be used as Circ.easeIn or Circ.easeOut or Circ.easeInOut

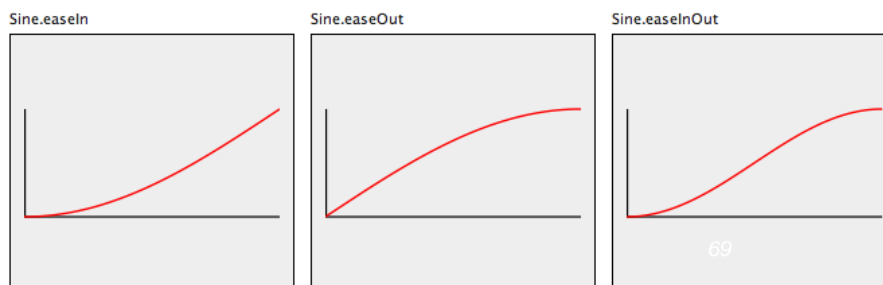
Graph:



Method **Sine**

displays a sineoidal transition. Must be used as Sine.easeIn or Sine.easeOut or Sine.easeInOut

Graph:



Method **Back**

makes the transition go back, then all forth. Must be used as `Back.easeIn` or `Back.easeOut` or `Back.easeInOut`

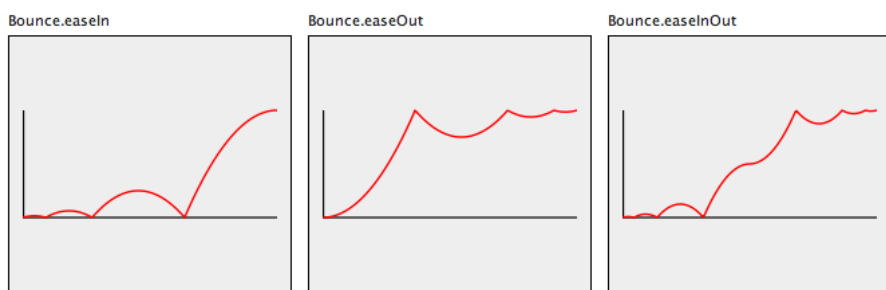
Graph:



Method **Bounce**

makes the transition bouncy. Must be used as `Bounce.easeIn` or `Bounce.easeOut` or `Bounce.easeInOut`

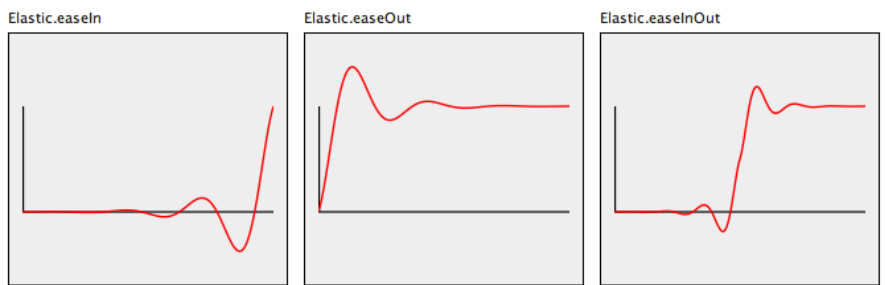
Graph:



Method **Elastic**

Elastic curve. Must be used as `Elastic.easeIn` or `Elastic.easeOut` or `Elastic.easeInOut`

Graph:



Drag.Base.js

Contains <Drag.Base>, <Element.makeResizable>

License:

MIT-style license.

Summary

Drag.Base.js	Contains Drag.Base, Element.makeResizable
Drag.Base	Modify two css properties of an element based on the position of the mouse.
Element	Custom class to allow all of its methods to be used with any DOM element via the dollar function \$.
makeResizable	Makes an element resizable (by dragging) with the supplied options.

Class Drag.Base

Modify two css properties of an element based on the position of the mouse.

Note:

Drag.Base requires an XHTML doctype.

Arguments:

el the \$(element) to apply the transformations to.
options optional. The options object.

Options

handle the \$(element) to act as the handle for the draggable element. defaults to the \$(element) itself.
modifiers an object. see Modifiers Below.
limit an object, see Limit below.
grid optional, distance in px for snap-to-grid dragging
snap optional, the distance you have to drag before the element starts to respond to the drag. defaults to false

modifiers:

x string, the style you want to modify when the mouse moves in an horizontal direction. defaults to 'left'
y string, the style you want to modify when the mouse moves in a vertical direction. defaults to 'top'

limit:

x array with start and end limit relative to modifiers.x
y array with start and end limit relative to modifiers.y

Events:

<code>onStart</code>	optional, function to execute when the user starts to drag (on mousedown);
<code>onComplete</code>	optional, function to execute when the user completes the drag.
<code>onDrag</code>	optional, function to execute at every step of the drag

Class `Element`

Custom class to allow all of its methods to be used with any DOM element via the dollar function `<$>`.

Method `makeResizable`

Makes an element resizable (by dragging) with the supplied options.

Arguments:

`options` see `<Drag.Base>` for acceptable options.

Drag.Move.js

Contains <Drag.Move>, <Element.makeDraggable>

License:

MIT-style license.

Summary

Drag.Move.js	Contains Drag.Move, Element.makeDraggable
Drag.Move	Extends Drag.Base, has additional functionality for dragging an element, support snapping and droppables.
Element	Custom class to allow all of its methods to be used with any DOM element via the dollar function \$.
makeDraggable	Makes an element draggable with the supplied options.

Class Drag.Move

Extends <Drag.Base>, has additional functionality for dragging an element, support snapping and droppables.

Drag.move supports either position absolute or relative. If no position is found, absolute will be set.

Inherits methods, properties, options and events from <Drag.Base>.

Note:

Drag.Move requires an XHTML doctype.

Arguments:

el the \$(element) to apply the drag to.

options optional. see Options below.

Options

all the drag.Base options, plus:

container an element, will fill automatically limiting options based on the \$(element) size and position. defaults to false (no limiting)

droppables an array of elements you can drop your draggable to.

overflown an array of nested scrolling containers, see Element::getPosition

Class Element

Custom class to allow all of its methods to be used with any DOM element via the dollar function <\$>.

Method makeDraggable

Makes an element draggable with the supplied options.

Arguments:

options see <Drag.Move> and <Drag.Base> for acceptable options.

XHR.js

Contains the basic XMLHttpRequest Class Wrapper.

License:

MIT-style license.

Summary

XHR.js	Contains the basic XMLHttpRequest Class Wrapper.
XHR	Basic XMLHttpRequest Wrapper.
setHeader	Add/modify an header for the request. It will not override headers from the options.
send	Opens the XHR connection and sends the data. Data has to be null or a string.
cancel	Cancels the running request. No effect if the request is not running.

Class XHR

Basic XMLHttpRequest Wrapper.

Arguments:

`options` an object with options names as keys. See options below.

Options

`method` 'post' or 'get' - the protocol for the request; optional, defaults to 'post'.
`async` boolean: asynchronous option; true uses asynchronous requests. Defaults to true.
`encoding` the encoding, defaults to utf-8.
`autoCancel` cancels the already running request if another one is sent. defaults to false.
`headers` accepts an object, that will be set to request headers.

Events:

`onRequest` function to execute when the XHR request is fired.
`onSuccess` function to execute when the XHR request completes.
`onStateChange` function to execute when the state of the XMLHttpRequest changes.
`onFailure` function to execute when the state of the XMLHttpRequest changes.

Properties

`running` true if the request is running.
`response` object, text and xml as keys. You can access this property in the onSuccess event.

Example:

```
var myXHR = new XHR({method: 'get'}).send('http://site.com/requestHandler.php',  
'name=john&lastname=dorian');
```

Method `setHeader`

Add/modify a header for the request. It will not override headers from the options.

Example:

```
var myXhr = new XHR(url, {method: 'get', headers: {'X-Request': 'JSON'}});
myXhr.setHeader('Last-Modified', 'Sat, 1 Jan 2005 05:00:00 GMT');
```

Method `send`

Opens the XHR connection and sends the data. Data has to be null or a string.

Example:

```
var myXhr = new XHR({method: 'post'});
myXhr.send(url, querystring);

var syncXhr = new XHR({async: false, method: 'post'});
syncXhr.send(url, null);
```

Method `cancel`

Cancels the running request. No effect if the request is not running.

Example:

```
var myXhr = new XHR({method: 'get'}).send(url);
myXhr.cancel();
```

Ajax.js

Contains the <Ajax> class. Also contains methods to generate querystrings from forms and Objects.

Credits:

Loosely based on the version from prototype.js <<http://prototype.conio.net>>

License:

MIT-style license.

Summary

Ajax.js	Contains the Ajax class. Also contains methods to generate querystrings from forms and Objects.
Ajax	An Ajax class, For all your asynchronous needs.
request	Executes the ajax request.
evalScripts	Executes scripts in the response text
getHeader	Returns the given response header or null
Element	Custom class to allow all of its methods to be used with any DOM element via the dollar function \$.
send	Sends a form with an ajax post request

Class Ajax

An Ajax class, For all your asynchronous needs.

Inherits methods, properties, options and events from <XHR>.

Arguments:

url the url pointing to the server-side script.
options optional, an object containing options.

Options

data you can write parameters here. Can be a querystring, an object or a Form element.
update \$(element) to insert the response text of the XHR into, upon completion of the request.
evalScripts boolean; default is false. Execute scripts in the response text onComplete. When the response is javascript the whole response is evaluated.
evalResponse boolean; default is false. Force global evaluation of the whole response, no matter what content-type it is.

Events:

onComplete function to execute when the ajax request completes.

Example:

```
var myAjax = new Ajax(url, {method: 'get'}).request();
```

Method request

Executes the ajax request.

Example:

```
var myAjax = new Ajax(url, {method: 'get'});  
myAjax.request();  
new Ajax(url, {method: 'get'}).request();
```

Method evalScripts

Executes scripts in the response text

Method getHeader

Returns the given response header or null

Function Object.toQueryString

Generates a querystring from key/pair values in an object

Arguments:

source the object to generate the querystring from.

Returns:

the query string.

Example:

```
Object.toQueryString({apple: "red", lemon: "yellow"}); //returns "apple=red&lemon=yellow"
```

Class Element

Custom class to allow all of its methods to be used with any DOM element via the dollar function <\$>.

Method send

Sends a form with an ajax post request

Arguments:

options option collection for ajax request. See <Ajax> for the options list.

Returns:

The Ajax Class Instance

Example:

```
<form id="myForm" action="submit.php">
<input name="email" value="bob@bob.com">
<input name="zipCode" value="90210">
</form>
<script>
$( 'myForm' ).send()
</script>
(end)
```

Cookie.js

A cookie reader/creator

Credits:

based on the functions by Peter-Paul Koch (<http://quirksmode.org>)

Class Cookie

Class for creating, getting, and removing cookies.

Method set

Sets a cookie in the browser.

Arguments:

key	the key (name) for the cookie
value	the value to set, cannot contain semicolons
options	an object representing the Cookie options. See Options below. Default values are stored in Cookie.options.

Options

domain

the domain the Cookie belongs to. If you want to share the cookie with pages located on a different domain, you have to set this value. Defaults to the current domain.

path

the path the Cookie belongs to. If you want to share the cookie with pages located in a different path, you have to set this value, for example to "/" to share the cookie with all pages

duration

the duration of the Cookie before it expires, in days.

If set to false or 0, the cookie will be a session cookie that expires when the browser is closed. This is default.

secure

Stored cookie information can be accessed only from a secure environment.

Returns:

An object with the options, the key and the value. You can give it as first parameter to Cookie.remove.

Example:

```
Cookie.set('username', 'Harald'); // session cookie (duration is false), or ...
Cookie.set('username', 'JackBauer', {duration: 1}); // save this for 1 day
```

Method get

Gets the value of a cookie.

Arguments:

key	the name of the cookie you wish to retrieve.
-----	--

Returns:

The cookie string value, or false if not found.

Example:

```
Cookie.get("username") //returns JackBauer
```

Method **remove**

Removes a cookie from the browser.

Arguments:

cookie the name of the cookie to remove or a previous cookie (for domains)
options optional. you can also pass the domain and path here. Same as options in <Cookie.set>

Examples:

```
Cookie.remove('username') //bye-bye JackBauer, cya in 24 hours

var myCookie = Cookie.set('username', 'Aaron', {domain: 'mootools.net'}); // Cookie.set returns an
object with all values need to remove the cookie
Cookie.remove(myCookie);
```

Json.js

Simple Json parser and Stringifier, See: <<http://www.json.org/>>

License:

MIT-style license.

Summary

Json.js	Simple Json parser and Stringifier, See: http://www.json.org/
Json	Simple Json parser and Stringifier, See: http://www.json.org/
toString	Converts an object to a string, to be passed in server-side scripts as a parameter. Although its not normal usage for this class, this method can also be used to convert functions and arrays to strings.
evaluate	converts a json string to an javascript Object.

Class **Json**

Simple Json parser and Stringifier, See: <<http://www.json.org/>>

Method **toString**

Converts an object to a string, to be passed in server-side scripts as a parameter. Although its not normal usage for this class, this method can also be used to convert functions and arrays to strings.

Arguments:

obj the object to convert to string

Returns:

A json string

Example:

```
Json.toString({apple: 'red', lemon: 'yellow'}); '{"apple":"red","lemon":"yellow"}'  
(end)
```

Method **evaluate**

converts a json string to an javascript Object.

Arguments:

str the string to evaluate. if its not a string, it returns false.
secure optionally, performs syntax check on json string. Defaults to false.

Credits:

Json test regexp is by Douglas Crockford <<http://crockford.org/>>.

Example:

```
var myObject = Json.evaluate('{"apple":"red","lemon":"yellow"}');
```

```
//myObject will become {apple: 'red', lemon: 'yellow'}
```

Json.Remote.js

Contains <Json.Remote>.

License:

MIT-style license.

Summary

Json.Remote.js	Contains Json.Remote.
Json.Remote	Wrapped XHR with automated sending and receiving of Javascript Objects in Json Format.

Class **Json.Remote**

Wrapped XHR with automated sending and receiving of Javascript Objects in Json Format.

Inherits methods, properties, options and events from <XHR>.

Arguments:

url the url you want to send your object to.

options see <XHR> options

Example:

```
var jSonRequest = new Json.Remote("http://site.com/tellMeAge.php", {onComplete: function(person){
alert(person.age); //is 25 years
alert(person.height); //is 170 cm
alert(person.weight); //is 120 kg
}}).send({'name': 'John', 'lastName': 'Doe'});
(end)
```

Assets.js

provides dynamic loading for images, css and javascript files.

License:

MIT-style license.

Summary

Assets.js	provides dynamic loading for images, css and javascript files.
javascript	Injects a javascript file in the page.
css	Injects a css file in the page.
image	Preloads an image and returns the img element. does not inject it to the page.
images	Preloads an array of images (as strings) and returns an array of img elements. does not inject them to the page.

Method javascript

Injects a javascript file in the page.

Arguments:

source the path of the javascript file
properties some additional attributes you might want to add to the script element

Example:

```
new Asset.javascript('/scripts/myScript.js', {id: 'myScript'});
```

Method CSS

Injects a css file in the page.

Arguments:

source the path of the css file
properties some additional attributes you might want to add to the link element

Example:

```
new Asset.css('/css/myStyle.css', {id: 'myStyle', title: 'myStyle'});
```

Method image

Preloads an image and returns the img element. does not inject it to the page.

Arguments:

source the path of the image file
properties some additional attributes you might want to add to the img element

Example:

```
new Asset.image('/images/myImage.png', {id: 'myImage', title: 'myImage', onload: myFunction});
```

Method images

Preloads an array of images (as strings) and returns an array of img elements. does not inject them to the page.

Arguments:

sources array, the paths of the image files
options object, see below

Options

onComplete a function to execute when all image files are loaded in the browser's cache
onProgress a function to execute when one image file is loaded in the browser's cache

Example:

```
new Asset.images(['/images/myImage.png', '/images/myImage2.gif'], {  
  onComplete: function(){  
    alert('all images loaded!');  
  }  
});  
(end)
```

Returns:

the img elements as \$\$\$. you can inject them anywhere you want with
<Element.injectInside>/<Element.injectAfter>/<Element.injectBefore>

Hash.js

Contains the class Hash.

License:

MIT-style license.

Summary

Hash.js	Contains the class Hash.
Hash	It wraps an object that it uses internally as a map. The user must use set(), get(), and remove() to add/change, retrieve and remove values, it must not access the internal object directly. null/undefined values are allowed.
get	Retrieves a value from the hash.
hasKey	Check the presence of a specified key-value pair in the hash.
set	Adds a key-value pair to the hash or replaces a previous value associated with the key.
remove	Removes a key-value pair from the hash.
each	Calls a function for each key-value pair. The first argument passed to the function will be the value, the second one will be the key, like \$each.
extend	Extends the current hash with an object containing key-value pairs. Values for duplicate keys will be replaced by the new ones.
merge	Merges the current hash with multiple objects.
empty	Empties all hash values properties and values.
keys	Returns an array containing all the keys, in the same order as the values returned by Hash.values.
values	Returns an array containing all the values, in the same order as the keys returned by Hash.keys.

Class Hash

It wraps an object that it uses internally as a map. The user must use set(), get(), and remove() to add/change, retrieve and remove values, it must not access the internal object directly. null/undefined values are allowed.

Note:

Each hash instance has the length property.

Arguments:

obj an object to convert into a Hash instance.

Example:

```
var hash = new Hash({a: 'hi', b: 'world', c: 'howdy'});
hash.remove('b'); // b is removed.
hash.set('c', 'hello');
hash.get('c'); // returns 'hello'
hash.length // returns 2 (a and c)
(end)
```

Method get

Retrieves a value from the hash.

Arguments:

key The key

Returns:

The value

Method `hasKey`

Check the presence of a specified key-value pair in the hash.

Arguments:

key The key

Returns:

True if the Hash contains a value for the specified key, otherwise false

Method `set`

Adds a key-value pair to the hash or replaces a previous value associated with the key.

Arguments:

key The key

value The value

Method `remove`

Removes a key-value pair from the hash.

Arguments:

key The key

Method `each`

Calls a function for each key-value pair. The first argument passed to the function will be the value, the second one will be the key, like `$each`.

Arguments:

fn The function to call for each key-value pair

bind Optional, the object that will be referred to as "this" in the function

Method `extend`

Extends the current hash with an object containing key-value pairs. Values for duplicate keys will be replaced by the new ones.

Arguments:

obj An object containing key-value pairs

Method `merge`

Merges the current hash with multiple objects.

Method `empty`

Empties all hash values properties and values.

Method `keys`

Returns an array containing all the keys, in the same order as the values returned by `<Hash.values>`.

Returns:

An array containing all the keys of the hash

Method `values`

Returns an array containing all the values, in the same order as the keys returned by `<Hash.keys>`.

Returns:

An array containing all the values of the hash

Function `$H`

Shortcut to create a Hash from an Object.

Hash.Cookie.js

Stores and loads an Hash as a cookie using Json format.

Class Hash.Cookie

Inherits all the methods from <Hash>, additional methods are save and load.

Hash json string has a limit of 4kb (4096byte), so be careful with your Hash size.

Creating a new instance automatically loads the data from the Cookie into the Hash.

If the Hash is emptied, the cookie is also removed.

Arguments:

name the key (name) for the cookie

options options are identical to <Cookie> and are simply passed along to it.

In addition, it has the autoSave option, to save the cookie at every operation. defaults to true.

Example:

```
var fruits = new Hash.Cookie('myCookieName', {duration: 3600});
fruits.extend({
  'lemon': 'yellow',
  'apple': 'red'
});
fruits.set('melon', 'green');
fruits.get('lemon'); // yellow

// ... on another page ... values load automatically

var fruits = new Hash.Cookie('myCookieName', {duration: 365});
fruits.get('melon'); // green

fruits.erase(); // delete cookie
(end)
```

Method save

Saves the Hash to the cookie. If the hash is empty, removes the cookie.

Returns:

Returns false when the JSON string cookie is too long (4kb), otherwise true.

Example:

```
var login = new Hash.Cookie('userstatus', {autoSave: false});

login.extend({
  'username': 'John',
  'credentials': [4, 7, 9]
});
```

```
login.set('last_message', 'User logged in!');  
  
login.save(); // finally save the Hash  
(end)
```

Method **load**

Loads the cookie and assigns it to the Hash.

Color.js

Contains the Color class.

License:

MIT-style license.

Summary

Color.js	Contains the Color class.
Color	Creates a new Color Object, which is an array with some color specific methods.
mix	Mixes two or more colors with the Color.
invert	Inverts the Color.
setHue	Modifies the hue of the Color, and returns a new one.
setSaturation	Changes the saturation of the Color, and returns a new one.
setBrightness	Changes the brightness of the Color, and returns a new one.
Array	A collection of The Array Object prototype methods.
rgbToHsb	Converts a RGB array to an HSB array.
hsbToRgb	Converts an HSB array to an RGB array.

Class Color

Creates a new Color Object, which is an array with some color specific methods.

Arguments:

- color** the hex, the RGB array or the HSB array of the color to create. For HSB colors, you need to specify the second argument.
- type** a string representing the type of the color to create. needs to be specified if you intend to create the color with HSB values, or an array of HEX values. Can be 'rgb', 'hsb' or 'hex'.

Example:

```
var black = new Color('#000');
var purple = new Color([255,0,255]);
// mix black with white and purple, each time at 10% of the new color
var darkpurple = black.mix('#fff', purple, 10);
$('myDiv').setStyle('background-color', darkpurple);
(end)
```

Method mix

Mixes two or more colors with the Color.

Arguments:

- color** a color to mix. you can use as arguments how many colors as you want to mix with the original one.
- alpha** if you use a number as the last argument, it will be treated as the amount of the color to mix.

Method invert

Inverts the Color.

Method `setHue`

Modifies the hue of the Color, and returns a new one.

Arguments:

value the hue to set

Method `setSaturation`

Changes the saturation of the Color, and returns a new one.

Arguments:

percent the percentage of the saturation to set

Method `setBrightness`

Changes the brightness of the Color, and returns a new one.

Arguments:

percent the percentage of the brightness to set

Function `$RGB`

Shortcut to create a new color, based on red, green, blue values.

Arguments:

r (integer) red value (0-255)
g (integer) green value (0-255)
b (integer) blue value (0-255)

Function `$HSB`

Shortcut to create a new color, based on hue, saturation, brightness values.

Arguments:

h (integer) hue value (0-100)
s (integer) saturation value (0-100)
b (integer) brightness value (0-100)

Class `Array`

A collection of The Array Object prototype methods.

Method `rgbToHsb`

Converts a RGB array to an HSB array.

Returns:

the HSB array.

Method `hsbToRgb`

Converts an HSB array to an RGB array.

Returns:

the RGB array.

Scroller.js

Contains the <Scroller>.

License:

MIT-style license.

Summary

Scroller.js	Contains the Scroller.
Scroller	The Scroller is a class to scroll any element with an overflow (including the window) when the mouse cursor reaches certain boundaries of that element.
start	The scroller starts listening to mouse movements.
stop	The scroller stops listening to mouse movements.

Class Scroller

The Scroller is a class to scroll any element with an overflow (including the window) when the mouse cursor reaches certain boundaries of that element.

You must call its start method to start listening to mouse movements.

Note:

The Scroller requires an XHTML doctype.

Arguments:

element required, the element to scroll.
options optional, see options below, and <Fx.Base> options.

Options

area integer, the necessary boundaries to make the element scroll.
velocity integer, velocity ratio, the modifier for the window scrolling speed.

Events:

onChange optionally, when the mouse reaches some boundaries, you can choose to alter some other values, instead of the scrolling offsets.

Automatically passes as parameters x and y values.

Method start

The scroller starts listening to mouse movements.

Method stop

The scroller stops listening to mouse movements.

Slider.js

Contains <Slider>

License:

MIT-style license.

Summary

Slider.js	Contains Slider
Slider	Creates a slider with two elements: a knob and a container. Returns the values.
set	The slider will get the step you pass.

Class Slider

Creates a slider with two elements: a knob and a container. Returns the values.

Note:

The Slider requires an XHTML doctype.

Arguments:

element	the knob container
knob	the handle
options	see Options below

Options

steps	the number of steps for your slider.
mode	either 'horizontal' or 'vertical'. defaults to horizontal.
offset	relative offset for knob position. default to 0.

Events:

onChange	a function to fire when the value changes.
onComplete	a function to fire when you're done dragging.
onTick	optionally, you can alter the onTick behavior, for example displaying an effect of the knob moving to the desired position.

Passes as parameter the new position.

Method set

The slider will get the step you pass.

Arguments:

step	one integer
------	-------------

SmoothScroll.js

Contains `<SmoothScroll>`

License:

MIT-style license.

Summary

SmoothScroll.js	Contains SmoothScroll
SmoothScroll	Auto targets all the anchors in a page and display a smooth scrolling effect upon clicking them.

Class SmoothScroll

Auto targets all the anchors in a page and display a smooth scrolling effect upon clicking them.

Inherits methods, properties, options and events from `<Fx.Scroll>`.

Note:

SmoothScroll requires an XHTML doctype.

Arguments:

`options` the `Fx.Scroll` options (see: `<Fx.Scroll>`) plus `links`, a collection of elements you want your smoothscroll on. Defaults to `document.links`.

Example:

```
new SmoothScroll();
```

Sortable.js

Contains <Sortable> Class.

License:

MIT-style license.

Summary

Sortable.js	Contains Sortables Class.
Sortables	Creates an interface for Drag.Base and drop, resorting of a list.

Class `Sortables`

Creates an interface for <Drag.Base> and drop, resorting of a list.

Note:

The Sortables require an XHTML doctype.

Arguments:

`list` required, the list that will become sortable.
`options` an Object, see options below.

Options

`handles` a collection of elements to be used for drag handles. defaults to the elements.

Events:

`onStart` function executed when the item starts dragging
`onComplete` function executed when the item ends dragging

Tips.js

Tooltips, BubbleTips, whatever they are, they will appear on mouseover

License:

MIT-style license.

Summary

Tips.js	Tooltips, BubbleTips, whatever they are, they will appear on mouseover
Tips	Display a tip on any element with a title and/or href.

Credits:

The idea behind Tips.js is based on Bubble Tooltips (<<http://web-graphics.com/mtarchive/001717.php>>) by Alessandro Fulcitiniti <<http://web-graphics.com>>

Class Tips

Display a tip on any element with a title and/or href.

Note:

Tips requires an XHTML doctype.

Arguments:

elements a collection of elements to apply the tooltips to on mouseover.
options an object. See options Below.

Options

maxTitleChars the maximum number of characters to display in the title of the tip. defaults to 30.
showDelay the delay the onShow method is called. (defaults to 100 ms)
hideDelay the delay the onHide method is called. (defaults to 100 ms)

className - the prefix for your tooltip classNames. defaults to 'tool'.

the whole tooltip will have as classname: tool-tip

the title will have as classname: tool-title

the text will have as classname: tool-text

offsets - the distance of your tooltip from the mouse. an Object with x/y properties.

fixed - if set to true, the tooltip will not follow the mouse.

Events:

onShow optionally you can alter the default onShow behaviour with this option (like displaying a fade in effect);

onHide optionally you can alter the default onHide behaviour with this option (like displaying a fade out effect);

```
ng a fade out effect);
```

Example:

```

<script>
var myTips = new Tips($$('.toolTipImg'), {
maxTitleChars: 50 //I like my captions a little long
});
</script>
(end)
```

Note:

The title of the element will always be used as the tooltip body. If you put :: on your title, the text before :: will become the tooltip title.

Group.js

For Grouping Classes or Elements Events. The Event added to the Group will fire when all of the events of the items of the group are fired.

License:

MIT-style license.

Summary

Group.js	For Grouping Classes or Elements Events. The Event added to the Group will fire when all of the events of the items of the group are fired.
Group	An "Utility" Class.
addEvent	adds an event to the stack of events of the Class instances.

Class Group

An "Utility" Class.

Arguments:

List of Class instances

Example:

```
xhr1 = new Ajax('data.js', {evalScript: true});
xhr2 = new Ajax('abstraction.js', {evalScript: true});
xhr3 = new Ajax('template.js', {evalScript: true});

var group = new Group(xhr1, xhr2, xhr3);
group.addEvent('onComplete', function(){
alert('All Scripts loaded');
});

xhr1.request();
xhr2.request();
xhr3.request();
(end)
```

Method addEvent

adds an event to the stack of events of the Class instances.

Arguments:

type string; the event name (e.g. 'onComplete')

fn function to execute when all instances fired this event

Accordion.js

Contains <Accordion>

License:

MIT-style license.

Summary

Accordion.js	Contains Accordion
Accordion	The Accordion class creates a group of elements that are toggled when their handles are clicked. When one elements toggles in, the others toggles back.
addSection	Dynamically adds a new section into the accordion at the specified position.
display	Shows a specific section and hides all others. Useful when triggering an accordion from outside.

Class Accordion

The Accordion class creates a group of elements that are toggled when their handles are clicked. When one elements toggles in, the others toggles back.

Inherits methods, properties, options and events from <Fx.Elements>.

Note:

The Accordion requires an XHTML doctype.

Arguments:

togglers	required, a collection of elements, the elements handlers that will be clickable.
elements	required, a collection of elements the transitions will be applied to.
options	optional, see options below, and <Fx.Base> options and events.

Options

show	integer, the Index of the element to show at start.
display	integer, the Index of the element to show at start (with a transition). defaults to 0 .
fixedHeight	integer, if you want the elements to have a fixed height. defaults to false.
fixedWidth	integer, if you want the elements to have a fixed width. defaults to false.
height	boolean, will add a height transition to the accordion if true. defaults to true.
opacity	boolean, will add an opacity transition to the accordion if true. defaults to true.
width	boolean, will add a width transition to the accordion if true. defaults to false, css mastery is required to make this work!
alwaysHide	boolean, will allow to hide all elements if true, instead of always keeping one element shown. defaults to false.

Events:

onActive	function to execute when an element starts to show
onBackground	function to execute when an element starts to hide

Method `addSection`

Dynamically adds a new section into the accordion at the specified position.

Arguments:

`toggler` (dom element) the element that toggles the accordion section open.
`element` (dom element) the element that stretches open when the toggler is clicked.
`pos` (integer) the index where these objects are to be inserted within the accordion.

Method `display`

Shows a specific section and hides all others. Useful when triggering an accordion from outside.

Arguments:

`index` integer, the index of the item to show, or the actual element to show.