# Developer Notes for prototype.js

**covers version 1.4.0**

*by Sergio Pereira*
*last update: March 10th 2006*

## What is that?

In case you haven't already used it, prototype.js (http://prototype.conio.net/) is a JavaScript library written by Sam Stephenson (http://www.conio.net). This amazingly well thought and well written piece of **standards-compliant** code takes a lot of the burden associated with creating rich, highly interactive web pages that characterize the Web 2.0 off your back.

If you tried to use this library recently, you probably noticed that documentation is not one of its strongest points. As many other developers before me, I got my head around prototype.js by reading the source code and experimenting with it. I thought it would be nice to take notes while I learned and share with everybody else.

I'm also offering an un-official reference (see p. 14) for the objects, classes, functions, and extensions provided by this library.

As you read the examples and the reference, developers familiar with the Ruby programming language will notice an intentional similarity between Ruby's built-in classes and many of the extensions implemented by this library.

## Related article

Advanced JavaScript guide (http://www.sergiopereira.com/articles/advjs.html).

## The utility functions

The library comes with many predefined objects and utility functions. The obvious goal of these functions is to save you a lot of repeated typing and idioms.

### Using the `$()` function

The **`$()`** function is a handy shortcut to the all-too-frequent **`document.getElementById()`** function of the DOM. Like the DOM function, this one returns the element that has the id passed as an argument.

Unlike the DOM function, though, this one goes further. You can pass more than one id and **`$()`** will return an **`Array`** object with all the requested elements. The example below should illustrate this.

```
<HTML>
<HEAD>
<TITLE> Test Page </TITLE>
<script src="prototype-1.4.0.js"></script>

<script>
        function test1()
        {
                var d = $('myDiv');
                alert(d.innerHTML);
        }

        function test2()
        {
                var divs = $('myDiv','myOtherDiv');
                for(i=0; i<divs.length; i++)
                {
                        alert(divs[i].innerHTML);
                }
        }
</script>
</HEAD>

<BODY>
        <div id="myDiv">
                <p>This is a paragraph</p>
        </div>
        <div id="myOtherDiv">
                <p>This is another paragraph</p>
        </div>

        <input type="button" value=Test1 onclick="test1();"><br>
        <input type="button" value=Test2 onclick="test2();"><br>

</BODY>
</HTML>
```

Another nice thing about this function is that you can pass either the id string or the element object itself, which makes this function very useful when creating other functions that can also take either form of argument.

## Using the `$F()` function

The **`$F()`** function is a another welcome shortcut. It returns the value of any field input control, like text boxes or drop-down lists. The function can take as argument either the element id or the element object itself.

```
<script>
	function test3()
	{
		alert(  $F('userName')  );
	}
</script>

<input type="text" id="userName" value="Joe Doe"><br>
<input type="button" value=Test3 onclick="test3();"><br>
```

## Using the `$A()` function

The **`$A()`** function converts the single argument it receives into an **`Array`** object.

This function, combined with the extensions for the Array class, makes it easier to convert or copy any enumerable list into an **`Array`** object. One suggested use is to convert DOM **`NodeLists`** into regular arrays, which can be traversed more efficiently. See example below.

```
<script>
function showOptions() {
  var someNodeList = $('lstEmployees').getElementsByTagName('option');
  var nodes = $A(someNodeList);
  nodes.each(function(node) {
	alert(node.nodeName + ': ' + node.innerHTML);
  });
}
</script>

<select id="lstEmployees" size="10" >
	<option value="5">Buchanan, Steven</option>
	<option value="8">Callahan, Laura</option>
	<option value="1">Davolio, Nancy</option>
</select>

<input type="button" value="Show the options" onclick="showOptions();">
```

## Using the `$H()` function

The **`$H()`** function converts objects into enumerable Hash objects that resemble associative arrays.

```
<script>
function testHash()
{
  //let's create the object
  var a = {
        first: 10,
        second: 20,
        third: 30
  };

  //now transform it into a hash
  var h = $H(a);
  alert(h.toQueryString()); //displays: first=10&second=20&third=30
}
</script>
```

## Using the `$R()` function

The **`$R()`** function is simply a short hand to writing **`new ObjectRange(lowerBound, upperBound, excludeBounds)`**.

Jump to the ObjectRange class documentation for a complete explanation of this class. In the meantime, let's take a look at a simple example that also shows the usage of iterators through the **`each`** method. More on that method will be found in the Enumerable object documentation.

```
<script>
        function demoDollar_R(){
                var range = $R(10, 20, false);
                range.each(function(value, index){
                        alert(value);
                });
        }

</script>

<input type="button" value="Sample Count" onclick="demoDollar_R();" >
```

### Using the `Try.these()` function

The **`Try.these()`** function makes it easy when you want to, ahem, try different function calls until one of them works. It takes a number of functions as arguments and calls them one by one, in sequence, until one of them works, returning the result of that successful function call.

In the example below, the function **`xmlNode.text`** works in some browsers, and **`xmlNode.textContent`** works in the other browsers. Using the **`Try.these()`** function we can return the one that works.

```
<script>
function getXmlNodeValue(xmlNode){
  return Try.these(
    function() {return xmlNode.text;},
    function() {return xmlNode.textContent;)
  );
}
</script>
```

## The `Ajax` object

The utility functions mentioned above are nice but, let's face it, they are not the most advanced type of thing, now are they? You could probably have done it yourself and you may even have similar functions in you own scripts. But those functions are just the tip of the iceberg.

I'm sure that your interest in prototype.js is driven mostly by its AJAX capabilities. So let's explain how the library makes your life easier when you need to perform AJAX logic.

The **`Ajax`** object is a pre-defined object, created by the library to wrap and simplify the tricky code that is involved when writing AJAX functionality. This object contains a number of classes that provide encapsulated AJAX logic. Let's take a look at some of them.

### Using the `Ajax.Request` class

If you are not using any helper library, you are probably writing a whole lot of code to create a **`XMLHttpRequest`** object and then track its progress asynchronously, then extract the response and process it. And consider yourself lucky if you do not have to support more than one type of browser.

To assist with AJAX functionality, the library defines the **`Ajax.Request`** class.

Let's say you have an application that can communicate with the server via the url *http://yoursever/app/get_sales?empID=1234&year=1998*, which returns an XML response like the following.

```
<?xml version="1.0" encoding="utf-8" ?>
<ajax-response>
	<response type="object" id="productDetails">
		<monthly-sales>
			<employee-sales>
				<employee-id>1234</employee-id>
				<year-month>1998-01</year-month>
				<sales>$8,115.36</sales>
			</employee-sales>
			<employee-sales>
				<employee-id>1234</employee-id>
				<year-month>1998-02</year-month>
				<sales>$11,147.51</sales>
			</employee-sales>
		</monthly-sales>
	</response>
</ajax-response>
```

Talking to the server to retrieve this XML is pretty simple using an **`Ajax.Request`** object. The sample below shows how it can be done.

```
<script>
  function searchSales()
  {
    var empID = $F('lstEmployees');
    var y = $F('lstYears');
    var url = 'http://yoursever/app/get_sales';
    var pars = 'empID=' + empID + '&year=' + y;

    var myAjax = new Ajax.Request(url,
    {
      method: 'get',
      parameters: pars,
      onComplete: showResponse
    });

  }

  function showResponse(originalRequest)
  {
    //put returned XML in the textarea
    $('result').value = originalRequest.responseText;
}
</script>

<select id="lstEmployees" size="10" onchange="searchSales()">
  <option value="5">Buchanan, Steven</option>
  <option value="8">Callahan, Laura</option>
  <option value="1">Davolio, Nancy</option>
</select>
```

```
<select id="lstYears" size="3" onchange="searchSales()">
  <option selected="selected" value="1996">1996</option>
  <option value="1997">1997</option>
  <option value="1998">1998</option>
</select>
<br><textarea id=result cols=60 rows=10 ></textarea>
```

Can you see the second parameter passed to the constructor of the **`Ajax.Request`** object? The parameter **`{method: 'get', parameters: pars, onComplete: showResponse}`** represents an anonymous object in literal notation (a.k.a. JSON). What it means is that we are passing an object that has a property named **`method`** that contains the string **`'get'`**, another property named **`parameters`** that contains the querystring of the HTTP request, and an **`onComplete`** property/method containing the function **`showResponse`**.

There are a few other properties that you can define and populate in this object, like **`asynchronous`**, which can be **`true`** or **`false`** and determines if the AJAX call to the server will be made asynchronously (the default value is **`true`**.)

This parameter defines the options for the AJAX call. In our sample, we are calling the url in the first argument via a HTTP GET command, passing the querystring contained in the variable **`pars`**, and the **`Ajax.Request`** object will call the **`showResponse`** function when it finishes retrieving the response.

As you may know, the **`XMLHttpRequest`** reports progress during the HTTP call. This progress can inform four different stages: *Loading*, *Loaded*, *Interactive*, or *Complete*. You can make the **`Ajax.Request`** object call a custom function in any of these stages, the *Complete* being the most common one. To inform the function to the object, simply provide property/methods named **`onXXXXX`** in the request options, just like the **`onComplete`** from our example. The function you pass in will be called by the object with two arguments, the first one will be the **`XMLHttpRequest`** (a.k.a. XHR) object itself, the second one will be the evaluated X-JSON response HTTP header (if one is present). You can then use the XHR to get the returned data and maybe check the **`status`** property, which will contain the HTTP result code of the call. The X-JSON header is useful if you want to return some script or JSON-formatted data.

Two other interesting options can be used to process the results. We can specify the **`onSuccess`** option as a function to be called when the AJAX call executes without errors and, conversily, the **`onFailure`** option can be a function to be called when a server error happens. Just like the **`onXXXXX`** option functions, these two will also be called passing the XHR that carried the AJAX call and the evaluated X-JSON header.

Our sample did not process the XML response in any interesting way. We just dumped the XML in the textarea. A typical usage of the response would probably find the desired information inside the XML and update some page elements, or maybe even some sort of XSLT transformation to produce HTML in the page.

In version 1.4.0, a new form of event callback handling is introduced. If you have code that should always be executed for a particular event, regardless of which AJAX call caused it to happen, then you can use the new Ajax.Responders object.

Let's suppose you want to show some visual indication that an AJAX call is in progress, like a spinning icon or something of that nature. You can use two global event handlers to help you, one to show the icon when the first call starts and another one to hide the icon when the last one finishes. See example below.

```
<script>
        var myGlobalHandlers = {
                onCreate: function(){
                        Element.show('systemWorking');
                },

                onComplete: function() {
                        if(Ajax.activeRequestCount == 0){
                                Element.hide('systemWorking');
                        }
                }
        };

        Ajax.Responders.register(myGlobalHandlers);
</script>

<div id='systemWorking'><img src='spinner.gif'>Loading...</div>
```

For more complete explanations, see the Ajax.Request reference and the options reference.

## Using the `Ajax.Updater` class

If you have a server endpoint that can return information already formatted in HTML, the library makes life even easier for you with the **`Ajax.Updater`** class. With it you just inform which element should be filled with the HTML returned from the AJAX call. An example speaks better than I can write.

```
<script>
        function getHTML()
        {
                var url = 'http://yourserver/app/getSomeHTML';
                var pars = 'someParameter=ABC';

                var myAjax = new Ajax.Updater(
                        'placeholder',
                        url,
                        {
                                method: 'get',
                                parameters: pars
                        });

        }
</script>

<input type=button value=GetHtml onclick="getHTML()">
```

```
<div id="placeholder"></div>
```

As you can see, the code is very similar to the previous example, with the exclusion of the **`onComplete`** function and the element id being passed in the constructor. Let's change the code a little bit to illustrate how it is possible to handle server errors on the client.

We will add more options to the call, specifying a function to capture error conditions. This is done using the **`onFailure`** option. We will also specify that the **`placeholder`** only gets populated in case of a successful operation. To achieve this we will change the first parameter from a simple element id to an object with two properties, **`success`** (to be used when everything goes OK) and **`failure`** (to be used when things go bad.) We will not be using the **`failure`** property in our example, just the **`reportError`** function in the **`onFailure`** option.

```
<script>
	function getHTML()
	{
		var url = 'http://yourserver/app/getSomeHTML';
		var pars = 'someParameter=ABC';

		var myAjax = new Ajax.Updater(
					{success: 'placeholder'},
					url,
					{
						method: 'get',
						parameters: pars,
						onFailure: reportError
					});

	}

	function reportError(request)
	{
		alert('Sorry. There was an error.');
	}
</script>

<input type=button value=GetHtml onclick="getHTML()">
<div id="placeholder"></div>
```

If your server logic returns JavaScript code along with HTML markup, the **`Ajax.Updater`** object can evaluate that JavaScript code. To get the object to treat the response as JavaScript, you simply add **`evalScripts: true;`** to the list of properties in the last argument of the object constructor. But there's a caveat. Those script blocks will not be added to the page's script. As the option name **`evalScripts`** suggests, the scripts will be **evaluated**. What's the difference, you may ask? Lets assume the requested URL returns something like this:

```
<script language="javascript" type="text/javascript">
	function sayHi(){
		alert('Hi');
	}
</script>

<input type=button value="Click Me" onclick="sayHi()">
```

In case you've tried it before, you know it doesn't work. The reason is that the script block will be evaluated, and evaluating a script like the above will not create a function named **`sayHi`**. It will do nothing. To create this function we need to change our script to **create** the function. See below.

```
<script language="javascript" type="text/javascript">

sayHi = function(){
		alert('Hi');
	};

</script>

<input type=button value="Click Me" onclick="sayHi()">
```

Note that in the previous example we did not use the **`var`** keyword to declare the variable. Doing so would have created a function object that would be local to the script block (at least in IE). Without the **`var`** keyword the function object is scoped to the window, which is our intent.

For more complete explanations, see the Ajax.Updater reference and the options reference.

## Enumerating... Wow! Damn! Wahoo!

We are all familar with for loops. You know, create yourself an array, populate it with elements of the same kind, create a loop control structure (for, foreach, while, repeat, etc,) access each element sequentially, by its numeric index, and do something with the element.

When you come to think about it, almost every time you have an array in your code it means that you'll be using that array in a loop sooner or later. Wouldn't it be nice if the array objects had more functionality to deal with these iterations? Yes, it would, and many programming languages provide such functionality in their arrays or equivalent structures (like collections and lists.)

Well, it turns out that prototype.js gives us the **`Enumerable`** object, which implements a plethora of tricks for us to use when dealing with iterable data. The prototype.js library goes one step further and extends the **`Array`** class with all the methods of **`Enumerable`**.

## Loops, Ruby-style

In standard javascript, if you wanted to sequentially display the elements of an array, you could very well write something like this.

```
<script>
	function showList(){
		var simpsons = ['Homer', 'Marge', 'Lisa', 'Bart', 
'Meg'];
		for(i=0;i<simpsons.length;i++){
			alert(simpsons[i]);
		}

	}

</script>

<input type="button" value="Show List" onclick="showList();" >
```

With our new best friend, prototype.js, we can rewrite this loop like this.

```
	function showList(){
		var simpsons = ['Homer', 'Marge', 'Lisa', 'Bart', 
'Meg'];
		simpsons.each( function(familyMember){
			alert(familyMember);
		});

	}
```

You are probably thinking "big freaking deal...just a weird syntax for the same old thing." Well, in the above example, yes, there's nothing too earth shattering going on. Afterall, there's not much to be changed in such a drop-dead-simple example. But keep reading, nonetheless.

Before we move on. Do you see this function that is being passed as an argument to the **each** method? Let's start referring to it as an **iterator** function.

## Your arrays on steroids

Like we mentioned above, it's very common for all the elements in your array to be of the same kind, with the same properties and methods. Let's see how we can take advantage of iterator functions with our new souped-up arrays.

Finding an element according to a criteria.

```
<script>
        function findEmployeeById(emp_id){
                var listBox = $('lstEmployees')
                var options = listBox.getElementsByTagName('option');
                options = $A(options);
                var opt = options.find( function(employee){
                        return (employee.value == emp_id);
                });
                alert(opt.innerHTML); //displays the employee name
        }
</script>

<select id="lstEmployees" size="10" >
        <option value="5">Buchanan, Steven</option>
        <option value="8">Callahan, Laura</option>
        <option value="1">Davolio, Nancy</option>
</select>

<input type="button" value="Find Laura" onclick="findEmployeeById(8);"
>
```

Now let's kick it up another notch. See how we can filter out items in arrays, then retrieve just a desired member from each element.

```
<script>
        function showLocalLinks(paragraph){
                paragraph = $(paragraph);
                var links = $A(paragraph.getElementsByTagName('a'));
                //find links that do not start with 'http'
                var localLinks = links.findAll( function(link){
                        var start = link.href.substring(0,4);
                        return start !='http';
                });
                //now the link texts
                var texts = localLinks.pluck('innerHTML');
                //get them in a single string
                var result = texts.inspect();
                alert(result);
        }

</script>
<p id="someText">
        This <a href="http://othersite.com/page.html">text</a> has
        a <a href="#localAnchor">lot</a> of
        <a href="#otherAnchor">links</a>. Some are
        <a href="http://wherever.com/page.html">external</a>
        and some are <a href="#someAnchor">local</a>
</p>
<input type=button value="Find Local Links"
onclick="showLocalLinks('someText')">
```

It takes just a little bit of practice to get completely addicted to this syntax. Take a look at the **`Enumerable`** and **`Array`** references for all the available functions.

## Reference for prototype.js

### Extensions to the JavaScript classes

One of the ways the prototype.js library adds functionality is by extending the existing JavaScript classes.

### Extensions for the `Object` class

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `extend(destination, source)` | static | destination: any object, source: any object | Provides a way to implement inheritance by copying all properties and methods from **`source`** to **`destination`**. |
| `inspect(targetObj)` | static | targetObj: any object | Returns a human-readable string representation of targetObj. It defaults to the return value of **`toString`** if the given object does not define an **`inspect`** instance method. |

### Extensions for the `Number` class

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `toColorPart()` | instance | (none) | Returns the hexadecimal representation of the number. Useful when converting the RGB components of a color into its HTML representation. |
| `succ()` | instance | (none) | Returns the next number. This function is used in scenarios that involve iteration. |
| `times(iterator)` | instance | iterator: a function object conforming to Function(index) | Calls the **`iterator`** function repeatedly passing the current index in the **`index`** argument. |

The following sample will display alert message boxes from 0 to 9.

```
<script>
       function demoTimes(){
              var n = 10;
              n.times(function(index){
                     alert(index);
              });
              /***************************
               * you could have also used:
               *           (10).times( .... );
               ***************************/
       }

</script>

<input type=button value="Test Number.times()" onclick="demoTimes()">
```

## Extensions for the `Function` class

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `bind(object)` | instance | `object`: the object that owns the method | Returns an instance of the function pre-bound to the function(=method) owner object. The returned function will have the same arguments as the original one. |
| `bindAsEventListener(object)` | instance | `object`: the object that owns the method | Returns an instance of the function pre-bound to the function(=method) owner object.The returned function will have the current **event** object as its argument. |

Let's see one of these extensions in action.

```
<input type=checkbox id=myChk value=1> Test?
<script>
        //declaring the class
        var CheckboxWatcher = Class.create();

        //defining the rest of the class implementation
        CheckboxWatcher.prototype = {

           initialize: function(chkBox, message) {
                        this.chkBox = $(chkBox);
                        this.message = message;
                        //assigning our method to the event

                        this.chkBox.onclick =
                           this.showMessage.bindAsEventListener(this);

           },

           showMessage: function(evt) {
                 alert(this.message + ' (' + evt.type + ')');
           }
        };


        var watcher = new CheckboxWatcher('myChk', 'Changed');
</script>
```

## Extensions for the `String` class

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `stripTags()` | instance | (none) | Returns the string with any HTML or XML tags removed |
| `stripScripts()` | instance | (none) | Returns the string with any **`<script />`** blocks removed |
| `escapeHTML()` | instance | (none) | Returns the string with any HTML markup characters properly escaped |
| `unescapeHTML()` | instance | (none) | The reverse of **`escapeHTML()`** |
| `extractScripts()` | instance | (none) | Returns an **`Array`** object containing all the **`<script />`** blocks found in the string. |
| `evalScripts()` | instance | (none) | Evaluates each **`<script />`** block found in the string. |
| `toQueryParams()` | instance | (none) | Splits a querystring into an associative **`Array`** indexed by parameter name (more like a hash). |
| `parseQuery()` | instance | (none) | Same as **`toQueryParams()`**. |
| `toArray()` | instance | (none) | Splits the string into an **`Array`** of its characters. |
| `camelize()` | instance | (none) | Converts a hyphen-delimited-string into a camelCaseString. This function is useful when writing code that deals with style properties, for example. |

## Extensions for the `Array` class

To start off, **`Array`** extends **`Enumerable`**, so all the handy methods defined in the **`Enumerable`** object are available. Besides that, the methods listed below are also implemented.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `clear()` | instance | (none) | Empties the array and returns itself. |
| `compact()` | instance | (none) | Returns the array without the elements that are **`null`** or **`undefined`**. This method does ot change the array itself |
| `first()` | instance | (none) | Returns the first element of the array. |
| `flatten()` | instance | (none) | Returns a flat, one-dimensional version of the array. This flattening happens by finding each of the array's elements that are also arrays and including their elements in the returned array, recursively. |
| `indexOf(value)` | instance | value: what you are looking for. | Returns the zero-based position of the given **`value`** if it is found in the array. Returns -1 if **`value`** is not found. |
| `inspect()` | instance | (none) | Overriden to return a nicely formatted string representation of the array with its elements. |
| `last()` | instance | (none) | Returns the last element of the array. |
| `reverse([applyToSelf])` | instance | applyToSelf: indicates if the array itself should also be reversed. | Returns the array in reverse sequence. If no argument is given or if the argument is **`true`** the array itself will also be reversed. Otherwise it remains unchanged. |
| `shift()` | instance | (none) | Returns the first element and removes it from the array, reducing the array's length by 1. |
| `without(value1 [, value2 [, .. valueN]])` | instance | value1 ... valueN: values to be excluded if present in the array. | Returns the array excluding the elements that are included in the list of arguments. |

## Extensions for the `document` DOM object

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `getElementsByClassName(className [, parentElement])` | instance | `className: name of a CSS class associated with the elements, parentElement: object or id of the element that contains the elements being retrieved.` | Returns all the elements that are associated with the given CSS class name. If no `parentElement` id given, the entire document body will be searched. |

## Extensions for the `Event` object

| Property | Type | Description |
|---|---|---|
| KEY_BACKSPACE | Number | 8: Constant. Code for the Backspace key. |
| KEY_TAB | Number | 9: Constant. Code for the Tab key. |
| KEY_RETURN | Number | 13: Constant. Code for the Return key. |
| KEY_ESC | Number | 27: Constant. Code for the Esc key. |
| KEY_LEFT | Number | 37: Constant. Code for the Left arrow key. |
| KEY_UP | Number | 38: Constant. Code for the Up arrow key. |
| KEY_RIGHT | Number | 39: Constant. Code for the Right arrow key. |
| KEY_DOWN | Number | 40: Constant. Code for the Down arrow key. |
| KEY_DELETE | Number | 46: Constant. Code for the Delete key. |
| observers: | Array | List of cached observers. Part of the internal implementation details of the object. |

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `element(event)` | static | `event: an Event object` | Returns element that originated the event. |
| `isLeftClick(event)` | static | `event: an Event object` | Returns `true` if the left mouse button was clicked. |
| `pointerX(event)` | static | `event: an Event object` | Returns the x coordinate of the mouse pointer on the page. |
| `pointerY(event)` | static | `event: an Event object` | Returns the y coordinate of the mouse pointer on the page. |
| `stop(event)` | static | `event: an Event object` | Use this function to abort the default behavior of an event and to suspend its propagation. |
| `findElement(event, tagName)` | static | `event: an Event object, tagName: name of the desired tag.` | Traverses the DOM tree upwards, searching for the first element with the given tag name, starting from the element that originated the event. |
| `observe(element, name, observer, useCapture)` | static | `element: object or id, name: event name (like 'click', 'load', etc), observer: function to handle the event, useCapture: if` **`true`**`, handles the event in the` *`capture`* `phase and if` **`false`** `in the` *`bubbling`* `phase.` | Adds an event handler function to an event. |
| `stopObserving(element, name, observer, useCapture)` | static | `element: object or id, name: event name (like 'click'), observer: function that is handling the event, useCapture: if true handles the event in the` *`capture`* `phase and if false in the` *`bubbling`* `phase.` | Removes an event handler from the event. |
| `_observeAndCache(element,` | static | | Private method, do not |

|  |  |  |  |
|---|---|---|---|
| name, observer, useCapture) |  |  | worry about it. |
| unloadCache() | static | (none) | Private method, do not worry about it. Clears all cached observers from memory. |

Let's see how to use this object to add an event handler to the load event of the **window** object.

```
<script>
	Event.observe(window, 'load', showMessage, false);

	function showMessage() {
	  alert('Page loaded.');
	}
</script>
```

## New objects and classes defined by prototype.js

Another way the library helps you is by providing many objects that implement both support for object oriented designs and common functionality in general.

## The PeriodicalExecuter object

This object provides the logic for calling a given function repeatedly, at a given interval.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| [ctor](callback, interval) | constructor | callback: a parameterless function, interval: number of seconds | Creates one instance of this object that will call the function repeatedly. |

| Property | Type | Description |
|---|---|---|
| callback | Function() | The function to be called. No parameters will be passed to it. |
| frequency | Number | This is actually the interval in seconds |
| currentlyExecuting | Boolean | Indicates if the function call is in progress |

## The Prototype object

The **Prototype** object does not have any important role, other than declaring the version of the library being used.

| Property | Type | Description |
|---|---|---|
| Version | String | The version of the library |
| emptyFunction | Function() | An empty function object |
| K | Function(obj) | A function object that just echoes back the given parameter. |
| ScriptFragment | String | A regular expression to identify scripts |

## The Enumerable object

The **Enumerable** object allows one to write more elegant code to iterate items in a list-like structure.

Many other objects extend the **Enumerable** object to leverage its useful interface.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| each(iterator) | instance | iterator: a function object conforming to Function(value, index) | Calls the given iterator function passing each element in the list in the first argument and the index of the element in the second argument |
| all([iterator]) | instance | iterator: a function object conforming to Function(value, index) | This function is a way to test the entire collection of values using a given function. **all** will return **false** if the iterator function returns **false** or **null** for any of the elements. It will return **true** otherwise. If no iterator is given, then the test will be if the element itself is different than **false** or **null**. You can simply read it as "check if all elements are not-false." |
| any(iterator) | instance | iterator: a function object conforming to Function(value, index), optional. | This function is a way to test the entire collection of values using a given function. **any** will return **true** if the iterator function does not returns **false** or **null** for any of the elements. It will return **false** otherwise. If no iterator is given, then the test will be if the element itself is different than **false** or **null**.You can simply read it as "check if any element is not-false." |
| collect(iterator) | instance | iterator: a function object conforming to Function(value, index) | Calls the iterator function for each element in the collection and returns each result in an **Array**, one result element for each element in the collection, in the same sequence. |
| detect(iterator) | instance | iterator: a function object conforming to Function(value, index) | Calls the iterator function for each element in the collection and returns the first element that caused the iterator function to return true (or, more precisely, not-false.) If no element returns true, then **detect** returns null. |
| entries() | instance | (none) | Same as **toArray()**. |
| find(iterator) | instance | iterator: a function object conforming to Function(value, index) | Same as **detect()**. |
| findAll(iterator) | instance | iterator: a function object conforming to Function(value, index) | Calls the iterator function for each element in the collection and returns an **Array** with all the elements that caused the iterator function to return a value that resolves to **true**. This function is the opposite of **reject()**. |
| grep(pattern [, iterator]) | instance | pattern: a RegExp object used to match the elements, iterator: a function object conforming to Function(value, index) | Tests the string value of each element in the collection against the **pattern** regular expression . The function will return an **Array** containing all the elements that matched the regular expression. If the iterator function is given, then the **Array** will contain the result of calling the iterator with each element that was a match. |

| | | | |
|---|---|---|---|
| include(obj) | instance | obj: any object | Tries to find the given object in the collection. Returns **true** if the object is found, **false** otherwise. |
| inject(initialValue, iterator) | instance | initialValue: any object to be used as the initial value, iterator: a function object conforming to Function(accumulator, value, index) | Combines all the elements of the collection using the iterator function. The iterator is called passing the result of the previous iteration in the accumulator argument. The first iteration gets **initialValue** in the **accumulator** argument. The last result is the final return value. |
| invoke(methodName [, arg1 [, arg2 [...]]]) | instance | methodName: name of the method that will be called in each element, arg1..argN: arguments that will be passed in the method invocation. | Calls the method specified by methodName in each element of the collection, passing any given arguments (arg1 to argN), and returns the results in an **Array** object. |
| map(iterator) | instance | iterator: a function object conforming to Function(value, index) | Same as **collect()**. |
| max([iterator]) | instance | iterator: a function object conforming to Function(value, index) | Returns the element with the greatest value in the collection or the greatest result of calling the iterator for each element in the collection, if an iterator is given. |
| member(obj) | instance | obj: any object | Same as **include()**. |
| min([iterator]) | instance | iterator: a function object conforming to Function(value, index) | Returns the element with the lowest value in the collection or the lowest result of calling the iterator for each element in the collection, if an iterator is given. |
| partition([iterator]) | instance | iterator: a function object conforming to Function(value, index) | Returns an **Array** containing two other arrays. The first array will contain all the elements that caused the iterator function to return **true** and the second array will contain the remaining elements. If the iterator is not given, then the first array will contain the elements that resolve to **true** and the other array will contain the remaining elements. |
| pluck(propertyName) | instance | propertyName name of the property that will be read from each element. This can also contain the index of the element | Retrieves the value to the property specified by propertyName in each element of the collection and returns the results in an **Array** object. |
| reject(iterator) | instance | iterator: a function object conforming to Function(value, index) | Calls the iterator function for each element in the collection and returns an **Array** with all the elements that caused the iterator function to return a value that resolves to **false**. This function is the opposite of **findAll()**. |
| select(iterator) | instance | iterator: a function object conforming to Function(value, index) | Same as **findAll()**. |
| sortBy(iterator) | instance | iterator: a function object conforming to Function(value, index) | Returns an **Array** with all the elements sorted according to the result the iterator function call. |
| toArray() | instance | (none) | Returns an **Array** with all the elements of the collection. |
| zip(collection1[, | instance | collection1 .. | Merges each given collection with the |

|  |  |  |  |
|---|---|---|---|
| collection2 [, ... collectionN [,transform]]]) |  | collectionN: enumerations that will be merged, transform: a function object conforming to Function(value, index) | current collection. The merge operation returns a new array with the same number of elements as the current collection and each element is an array (let's call them sub-arrays) of the elements with the same index from each of the merged collections. If the transform function is given, then each sub-array will be transformed by this function before being returned. Quick example: [1,2,3].zip([4,5,6], [7,8,9]).inspect() returns "[ [1,4,7],[2,5,8],[3,6,9] ]" |

## The `Hash` object

The `Hash` object implements a hash structure, i.e. a collection of Key:Value pairs.

Each item in a `Hash` object is an array with two elements: first the key then the value. Each item also has two properties: `key` and `value`, which are pretty self-explanatory.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| keys() | instance | (none) | Returns an `Array` with the keys of all items. |
| values() | instance | (none) | Returns an `Array` with the values of all items. |
| merge(otherHash) | instance | otherHash: Hash object | Combines the hash with the other hash passed in and returns the new resulting hash. |
| toQueryString() | instance | (none) | Returns all the items of the hash in a string formatted like a query string, e.g. `'key1=value1&key2=value2&key3=value3'` |
| inspect() | instance | (none) | Overriden to return a nicely formatted string representation of the hash with its key:value pairs. |

## The `ObjectRange` class

*Inherits from* `Enumerable`

Represents a range of values, with upper and lower bounds.

| Property | Type | Kind | Description |
|---|---|---|---|
| start | (any) | instance | The lower bound of the range |
| end | (any) | instance | The upper bound of the range |
| exclusive | Boolean | instance | Determines if the boundaries themselves are part of the range. |

| Method | Kind | Arguments | Description |
|---|---|---|---|
| [ctor](start, end, exclusive) | constructor | start: the lower bound, end: the upper bound, exclusive: include the bounds in the range? | Creates one range object, spanning from `start` to `end`. It is important to note that `start` and `end` have to be objects of the same type and they must have a `succ()` method. |
| include(searchedValue) | instance | searchedValue: value that we are looking for | Checks if the given value is part of the range. Returns `true` or `false`. |

## The `Class` object

The **`Class`** object is used when declaring the other classes in the library. Using this object when declaring a class causes the to new class to support an **`initialize()`** method, which serves as the constructor.

See the sample below.

```
//declaring the class
var MySampleClass = Class.create();

//defining the rest of the class implmentation
MySampleClass.prototype = {

   initialize: function(message) {
                this.message = message;
   },

   showMessage: function(ajaxResponse) {
      alert(this.message);
   }
};

//now, let's instantiate and use one object
var myTalker = new MySampleClass('hi there.');
myTalker.showMessage(); //displays alert
```

| Method | Kind | Arguments | Description |
|---|---|---|---|
| create(*) | instance | (any) | Defines a constructor for a new class |

## The `Ajax` object

This object serves as the root and namespace for many other classes that provide AJAX functionality.

| Property | Type | Kind | Description |
|---|---|---|---|
| activeRequestCount | Number | instance | The number of AJAX requests in progress. |

| Method | Kind | Arguments | Description |
|---|---|---|---|
| getTransport() | instance | (none) | Returns a new **`XMLHttpRequest`** object |

## The `Ajax.Responders` object

*Inherits from* ***`Enumerable`***

This object maintains a list of objects that will be called when Ajax-related events occur. You can use this object, for example, if you want to hook up a global exception handler for AJAX operations.

| Property | Type | Kind | Description |
|---|---|---|---|
| responders | Array | instance | The list of objects registered for AJAX events notifications. |

| Method | Kind | Arguments | Description |
|---|---|---|---|
| register(responderToAdd) | instance | responderToAdd: object with methods that will be called. | The object passed in the **`responderToAdd`** argument should contain methods named like the AJAX events (e.g. **`onCreate`**, **`onComplete`**, **`onException`**, etc.) When the corresponding event occurs all the registered objects that contain a method with the appropriate name will have that method called. |
| unregister(responderToRemove) | instance | responderToRemove: object to be removed from the list. | The object passed in the **`responderToRemove`** argument will be removed from the list of registered objects. |
| dispatch(callback, request, transport, json) | instance | callback: name of the AJAX event being reported, request: the Ajax.Request object responsible for the event, transport: the XMLHttpRequest object that carried (or is carrying) the AJAX call, json: the X-JSON header of the response (if present) | Runs through the list of registered objects looking for the ones that have the method determined in the **`callback`** argument. Then each of these methods is called passing the other 3 arguments. If the AJAX response contains a **`X-JSON`** HTTP header with some JSON content, then it will be evaluated and passed in the **`json`** argument. If the event is **`onException`**, the transport argument will have the exception instead and **`json`** will not be passed. |

## The `Ajax.Base` class

This class is used as the base class for most of the other classes defined in the **`Ajax`** object.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `setOptions(options)` | instance | options: AJAX options | Sets the desired options for the AJAX operation |
| `responseIsSuccess()` | instance | (none) | Returns **`true`** if the AJAX operation succeeded, **`false`** otherwise |
| `responseIsFailure()` | instance | (none) | The opposite of **`responseIsSuccess()`**. |

## The `Ajax.Request` class

*Inherits from **`Ajax.Base`***

Encapsulates AJAX operations

| Property | Type | Kind | Description |
|---|---|---|---|
| Events | Array | static | List of possible events/statuses reported during an AJAX operation. The list contains: 'Uninitialized', 'Loading', 'Loaded', 'Interactive', and 'Complete.' |
| transport | XMLHttpRequest | instance | The **`XMLHttpRequest`** object that carries the AJAX operation |
| url | String | instance | The URL targeted by the request. |

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `[ctor](url, options)` | constructor | url: the url to be fetched, options: AJAX options | Creates one instance of this object that will call the given **`url`** using the given **`options`**. The onCreate event will be raised during the constructor call. **Important:** It is worth noting that the chosen url is subject to the browser's security settings. In many cases the browser will not fetch the url if it is not from the same host (domain) as the current page. You should ideally use only local urls to avoid having to configure or restrict the user's browser. (Thanks Clay). |
| `evalJSON()` | instance | (none) | This method is typically not called externally. It is called internally to evaluate the content of an eventual **`X-JSON`** HTTP header present in the AJAX response. |
| `evalReponse()` | instance | (none) | This method is typically not called externally. If the AJAX response has a **`Content-type`** header of **`text/javascript`** then the response body will be evaluated and this method will be used. |
| `header(name)` | instance | name: HTTP header name | This method is typically not called externally. It is called internally to retrieve the contents of any HTTP header of the AJAX response. |
| `onStateChange()` | instance | (none) | This method is typically not called externally. It is called by the object itself when the AJAX call status changes. |
| `request(url)` | instance | url: url for the AJAX | This method is typically not called externally. It is already called during the |

| | | | |
|---|---|---|---|
| | | call | constructor call. |
| respondToReadyState(readyState) | instance | readyState: state number (1 to 4) | This method is typically not called externally. It is called by the object itself when the AJAX call status changes. |
| setRequestHeaders() | instance | (none) | This method is typically not called externally. It is called by the object itself to assemble the HTTP header that will be sent during the HTTP request. |

## The `options` argument object

An important part of the AJAX operations is the **`options`** argument. There's no **`options`** class per se. Any object can be passed, as long as it has the expected properties. It is common to create anonymous objects just for the AJAX calls.

| **Property** | **Type** | **Default** | **Description** |
|---|---|---|---|
| method | String | 'post' | Method of the HTTP request |
| parameters | String | '' | The url-formatted list of values passed to the request |
| asynchronous | Boolean | true | Indicates if the AJAX call will be made asynchronously |
| postBody | String | undefined | Content passed to in the request's body in case of a HTTP POST |
| requestHeaders | Array | undefined | List of HTTP headers to be passed with the request. This list must have an even number of items, any odd item is the name of a custom header, and the following even item is the string value of that header. Example: **`['my-header1', 'this is the value', 'my-other-header', 'another value']`** |
| onXXXXXXXX | Function(XMLHttpRequest, Object) | undefined | Custom function to be called when the respective event/status is reached during the AJAX call. Example **`var myOpts = {onComplete: showResponse, onLoaded: registerLoaded};`**. The function used will receive one argument, containing the **`XMLHttpRequest`** object that is carrying the AJAX operation and another argument containing the evaluated X-JSON response HTTP header. |
| onSuccess | Function(XMLHttpRequest, Object) | undefined | Custom function to be called when the AJAX call completes successfully. The function used will receive one argument, containing the **`XMLHttpRequest`** object that is carrying the AJAX operation and another argument containing the evaluated X-JSON response HTTP header. |
| onFailure | Function(XMLHttpRequest, Object) | undefined | Custom function to be called when the AJAX call completes with error. The function used will receive one argument, containing the **`XMLHttpRequest`** object that is carrying the AJAX operation and another argument containing the evaluated X-JSON response HTTP header. |
| onException | Function(Ajax.Request, exception) | undefined | Custom function to be called when an exceptional condition happens on the client side of the AJAX call, like an invalid response or invalid arguments. The function used will receive two arguments, |

| | | | |
|---|---|---|---|
| | | | containing the `Ajax.Request` object that wraps the AJAX operation and the exception object. |
| insertion | an Insertion class | undefined | A class that will determine how the new content will be inserted. It can be `Insertion.Before`, `Insertion.Top`, `Insertion.Bottom`, or `Insertion.After`. Applies only to `Ajax.Updater` objects. |
| evalScripts | Boolean | undefined, false | Determines if script blocks will be evaluated when the response arrives. Applies only to `Ajax.Updater` objects. |
| decay | Number | undefined, 1 | Determines the progressive slowdown in a `Ajax.PeriodicalUpdater` object refresh rate when the received response is the same as the last one. For example, if you use 2, after one of the refreshes produces the same result as the previous one, the object will wait twice as much time for the next refresh. If it repeats again, the object will wait four times as much, and so on. Leave it undefined or use 1 to avoid the slowdown. |

## The `Ajax.Updater` class

*Inherits from `Ajax.Request`*

Used when the requested url returns HTML that you want to inject directly in a specific element of your page. You can also use this object when the url returns `<script>` blocks that will be evaluated upon arrival. Use the `evalScripts` option to work with scripts.

| Property | Type | Kind | Description |
|---|---|---|---|
| containers | Object | instance | This object contains two properties: `containers.success` will be used when the AJAX call succeeds, and `containers.failure` will be used otherwise. |

| Method | Kind | Arguments | Description |
|---|---|---|---|
| [ctor](container, url, options) | constructor | container:this can be the id of an element, the element object itself, or an object with two properties - **object.success** element (or id) that will be used when the AJAX call succeeds, and **object.failure** element (or id) that will be used otherwise. url: the url to be fetched, options: `AJAX options` | Creates one instance of this object that will call the given `url` using the given `options`. |
| updateContent() | instance | (none) | This method is typically not called externally. It is called by the object itself when the response is received. It will update the appropriate element with the HTML or call the function passed in the `insertion` option. The function will be called with two arguments, the element to be updated and the response text. |

## The `Ajax.PeriodicalUpdater` class

*Inherits from* **`Ajax.Base`**

This class repeatedly instantiates and uses an **`Ajax.Updater`** object to refresh an element on the page, or to perform any of the other tasks the **`Ajax.Updater`** can perform. Check the Ajax.Updater reference for more information.

| Property | Type | Kind | Description |
|---|---|---|---|
| container | Object | instance | This value will be passed straight to the **Ajax.Updater**'s constructor. |
| url | String | instance | This value will be passed straight to the **Ajax.Updater**'s constructor. |
| frequency | Number | instance | Interval (not frequency) between refreshes, in seconds. Defaults to 2 seconds. This number will be multiplied by the current **decay** when invoking the**Ajax.Updater** object |
| decay | Number | instance | Keeps the current decay level applied when re-executing the task |
| updater | Ajax.Updater | instance | The most recently used **Ajax.Updater** object |
| timer | Object | instance | The JavaScript timer being used to notify the object when it is time for the next refresh. |

| Method | Kind | Arguments | Description |
|---|---|---|---|
| [ctor](container, url, options) | constructor | container:this can be the id of an element, the element object itself, or an object with two properties - **object.success** element (or id) that will be used when the AJAX call succeeds, and **object.failure** element (or id) that will be used otherwise. url: the url to be fetched, options: **AJAX options** | Creates one instance of this object that will call the given **url** using the given **options**. |
| start() | instance | (none) | This method is typically not called externally. It is called by the object itself to start performing its periodical tasks. |
| stop() | instance | (none) | This method is typically not called externally. It is called by the object itself to stop performing its periodical tasks. |
| updateComplete() | instance | (none) | This method is typically not called externally. It is called by the currently used **Ajax.Updater** after it completes the request. It is used to schedule the next refresh. |
| onTimerEvent() | instance | (none) | This method is typically not called externally. It is called internally when it is time for the next update. |

## The `Element` object

This object provides some utility functions for manipulating elements in the DOM.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| addClassName(element, className) | instance | element: element object or id, className: name of a CSS class | Adds the given class name to the element's class names. |
| classNames(element) | instance | element: element object or id | Returns an `Element.ClassNames` object representing the CSS class names associated with the given element. |
| cleanWhitespace(element) | instance | element: element object or id | Removes any white space text node children of the element |
| empty(element) | instance | element: element object or id | Returns a `Boolean` value indicating if the element tag is empty (or has only whitespaces) |
| getDimensions(element) | instance | element: element object or id | |
| getHeight(element) | instance | element: element object or id | Returns the `offsetHeight` of the element |
| getStyle(element, cssProperty) | instance | element: element object or id, cssProperty name of a CSS property (either format 'prop-name' or 'propName' works). | Returns the value of the CSS property in the given element or `null` if not present. |
| hasClassName(element, className) | instance | element: element object or id, className: name of a CSS class | Returns `true` if the element has the given class name as one of its class names. |
| hide(elem1 [, elem2 [, elem3 [...]]]) | instance | elemN: element object or id | Hides each element by setting its `style.display` to `'none'`. |
| makeClipping(element) | instance | element: element object or id | |
| makePositioned(element) | instance | element: element object or id | |
| remove(element) | instance | element: element object or id | Removes the element from the document. |
| removeClassName(element, className) | instance | element: element object or id, className: name of a CSS class | Removes the given class name from the element's class names. |
| scrollTo(element) | instance | element: element object or id | Scrolls the window to the element position. |
| setStyle(element, cssPropertyHash) | instance | element: element object or id, cssPropertyHash Hash object with the styles to be applied. | Sets the value of the CSS properties in the given element, according to the values in the `cssPropertyHash` argument. |
| show(elem1 [, elem2 [, elem3 [...]]]) | instance | elemN: element object or id | Shows each element by resetting its `style.display` to `''`. |
| toggle(elem1 [, elem2 [, elem3 [...]]]) | instance | elemN: element object or id | Toggles the visibility of each passed element. |
| undoClipping(element) | instance | element: element object or id | |
| undoPositioned(element) | instance | element: element object or id | |
| update(element, html) | instance | element: element object or id, html: | Replaces the inner html of the element with the given html argument. If the |

|  |  |  |  |
|---|---|---|---|
|  |  | html content | given html contains `<script>` blocks they will not be included but they will be evaluated. |
| visible(element) | instance | element: element object or id | Returns a `Boolean` value indicating if the element is visible. |

## The `Element.ClassNames` class

*Inherits from* **`Enumerable`**

Represents the collection of CSS class names associated with an element.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `[ctor](element)` | constructor | `element: any DOM element object or id` | Creates an `Element.ClassNames` object representing the CSS class names of the given element. |
| `add(className)` | instance | `className: a CSS class name` | Includes the given CSS class name in the list of class names associated with the element. |
| `remove(className)` | instance | `className: a CSS class name` | Removes the given CSS class name from the list of class names associated with the element. |
| `set(className)` | instance | `className: a CSS class name` | Associates the element with the given CSS class name, removing any other class names from the element. |

## The `Abstract` object

This object serves as the root for other classes in the library. It does not have any properties or methods. The classes defined in this object are also treated as traditional abstract classes.

## The `Abstract.Insertion` class

This class is used as the base class for the other classes that will provide dynamic content insertion. This class is used like an abstract class.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `[ctor](element, content)` | constructor | `element: element object or id, content: HTML to be inserted` | Creates an object that will help with dynamic content insertion. |
| `contentFromAnonymousTable()` | instance | `(none)` | |

| Property | Type | Kind | Description |
|---|---|---|---|
| `adjacency` | `String` | `static, parameter` | Parameter that specifies where the content will be placed relative to the given element. The possible values are: `'beforeBegin'`, `'afterBegin'`, `'beforeEnd'`, and `'afterEnd'`. |
| `element` | `Object` | `instance` | The element object that the insertion will be made relative to. |
| `content` | `String` | `instance` | The HTML that will be inserted. |

## The `Insertion` object

This object serves as the root for other classes in the library. It does not have any properties or methods. The classes defined in this object are also treated as traditional abstract classes.

## The `Insertion.Before` class

*Inherits from* ***`Abstract.Insertion`***

Inserts HTML before an element.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `[ctor](element, content)` | constructor | `element`: element object or id, `content`: HTML to be inserted | Inherited from **`Abstract.Insertion`**. Creates an object that will help with dynamic content insertion. |

The following code

```
<br>Hello, <span id="person" style="color:red;">Wiggum. How's it
going?</span>

<script> new Insertion.Before('person', 'Chief '); </script>
```

Will change the HTML to

```
<br>Hello, Chief <span id="person" style="color:red;">Wiggum. How's it
going?</span>
```

## The `Insertion.Top` class

*Inherits from* ***`Abstract.Insertion`***

Inserts HTML as the first child under an element. The content will be right after the opening tag of the element.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `[ctor](element, content)` | constructor | `element`: element object or id, `content`: HTML to be inserted | Inherited from **`Abstract.Insertion`**. Creates an object that will help with dynamic content insertion. |

The following code

```
<br>Hello, <span id="person" style="color:red;">Wiggum. How's it
going?</span>

<script> new Insertion.Top('person', 'Mr. '); </script>
```

Will change the HTML to

```
<br>Hello, <span id="person" style="color:red;">Mr. Wiggum. How's it
going?</span>
```

## The `Insertion.Bottom` class

*Inherits from* ***`Abstract.Insertion`***

Inserts HTML as the last child under an element. The content will be right before the element's closing tag.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| [ctor](element, content) | constructor | element: element object or id, content: HTML to be inserted | Inherited from `Abstract.Insertion`. Creates an object that will help with dynamic content insertion. |

The following code

```
<br>Hello, <span id="person" style="color:red;">Wiggum. How's it
going?</span>

<script> new Insertion.Bottom('person', " What's up?"); </script>
```

Will change the HTML to

```
<br>Hello, <span id="person" style="color:red;">Wiggum. How's it going?
What's up?</span>
```

## The `Insertion.After` class

*Inherits from* ***`Abstract.Insertion`***

Inserts HTML right after the element's closing tag.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| [ctor](element, content) | constructor | element: element object or id, content: HTML to be inserted | Inherited from `Abstract.Insertion`. Creates an object that will help with dynamic content insertion. |

The following code

```
<br>Hello, <span id="person" style="color:red;">Wiggum. How's it
going?</span>

<script> new Insertion.After('person', ' Are you there?'); </script>
```

Will change the HTML to

```
<br>Hello, <span id="person" style="color:red;">Wiggum. How's it
going?</span> Are you there?
```

## The `Field` object

This object provides some utility functions for working with input fields in forms.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `clear(field1 [, field2 [, field3 [...]]])` | instance | `fieldN: field element object or id` | Clears the value of each passed form field element. |
| `present(field1 [, field2 [, field3 [...]]])` | instance | `fieldN: field element object or id` | Returns `true` only if all forms fields contain non-empty values. |
| `focus(field)` | instance | `field: field element object or id` | Moves the input focus to the given form field. |
| `select(field)` | instance | `field: field element object or id` | Selects the value in fields that support text selection |
| `activate(field)` | instance | `field: field element object or id` | Move the focus and selects the value in fields that support text selection |

## The `Form` object

This object provides some utility functions for working with data entry forms and their input fields.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `serialize(form)` | instance | `form: form element object or id` | Returns a url-formatted list of field names and their values, like `'field1=value1&field2=value2&field3=value3'` |
| `findFirstElement(form)` | instance | `form: form element object or id` | Returns the first enabled field element in the form. |
| `getElements(form)` | instance | `form: form element object or id` | Returns an `Array` containing all the input fields in the form. |
| `getInputs(form [, typeName [, name]])` | instance | `form: form element object or id, typeName: the type of the input element, name: the name of the input element.` | Returns an `Array` containing all the `<input>` elements in the form. Optionally, the list can be filtered by the `type` or `name` attributes of the elements. |
| `disable(form)` | instance | `form: form element object or id` | Disables all the input fields in the form. |
| `enable(form)` | instance | `form: form element object or id` | Enables all the input fields in the form. |

| | | | |
|---|---|---|---|
| focusFirstElement(form) | instance | form: form element object or id | Activates the first visible, enabled input field in the form. |
| reset(form) | instance | form: form element object or id | Resets the form. The same as calling the **reset()** method of the form object. |

## The **Form.Element** object

This object provides some utility functions for working with form elements, visible or not.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| serialize(element) | instance | element: element object or id | Returns the element's name=value pair, like **'elementName=elementValue'** |
| getValue(element) | instance | element: element object or id | Returns the value of the element. |

## The **Form.Element.Serializers** object

This object provides some utility functions that are used internally in the library to assist extracting the current value of the form elements.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| inputSelector(element) | instance | element: object or id of a form element that has the *checked* property, like a radio button or checkbox. | Returns an **Array** with the element's name and value, like **['elementName', 'elementValue']** |
| textarea(element) | instance | element: object or id of a form element that has the *value* property, like a textbox, button or password field. | Returns an **Array** with the element's name and value, like **['elementName', 'elementValue']** |
| select(element) | instance | element: object of a &lt;select&gt; element | Returns an **Array** with the element's name and all selected options' values or texts, like **['elementName', 'selOpt1 selOpt4 selOpt9']** |

## The `Abstract.TimedObserver` class

This class is used as the base class for the other classes that will monitor one element until its value (or whatever property the derived class defines) changes. This class is used like an abstract class.

Subclasses can be created to monitor things like the input value of an element, or one of the style properties, or number of rows in a table, or whatever else you may be interested in tracking changes to.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `[ctor](element, frequency, callback)` | constructor | element: element object or id, frequency: interval in seconds, callback: function to be called when the element changes | Creates an object that will monitor the element. |
| Derived classes have to implement this method to determine what is the current value being monitored in the element. | | | |
| `registerCallback()` | instance | (none) | This method is typically not called externally. It is called by the object itself to start monitoring the element. |
| `onTimerEvent()` | instance | (none) | This method is typically not called externally. It is called by the object itself periodically to check the element. |

| Property | Type | Description |
|---|---|---|
| element | Object | The element object that is being monitored. |
| frequency | Number | This is actually the interval in seconds between checks. |
| callback | Function(Object, String) | The function to be called whenever the element changes. It will receive the element object and the new value. |
| lastValue | String | The last value verified in the element. |

## The `Form.Element.Observer` class

*Inherits from* **`Abstract.TimedObserver`**

Implementation of an **`Abstract.TimedObserver`** that monitors the value of form input elements. Use this class when you want to monitor an element that does not expose an event that reports the value changes. In that case you can use the **`Form.Element.EventObserver`** class instead.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `[ctor](element, frequency, callback)` | constructor | element: element object or id, frequency: interval in seconds, callback: function to be called when the element changes | Inherited from **`Abstract.TimedObserver`**. Creates an object that will monitor the element's **`value`** property. |
| `getValue()` | instance | (none) | Returns the element's value. |

## The `Form.Observer` class

*Inherits from* **`Abstract.TimedObserver`**

Implementation of an **`Abstract.TimedObserver`** that monitors any changes to any data entry element's value in a form. Use this class when you want to monitor a form that contais a elements that do not expose an event that reports the value changes. In that case you can use the **`Form.EventObserver`** class instead.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `[ctor](form, frequency, callback)` | constructor | `form`: form object or id, `frequency`: interval in seconds, `callback` function to be called when any data entry element in the form changes | Inherited from **`Abstract.TimedObserver`**. Creates an object that will monitor the form for changes. |
| `getValue()` | instance | (none) | Returns the serialization of all form's data. |

## The `Abstract.EventObserver` class

This class is used as the base class for the other classes that execute a callback function whenever a value-changing event happens for an element.

Multiple objects of type **`Abstract.EventObserver`** can be bound to the same element, without one wiping out the other. The callbacks will be executed in the order they are assigned to the element.

The triggering event is **`onclick`** for radio buttons and checkboxes, and **`onchange`** for textboxes in general and listboxes/dropdowns.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| `[ctor](element, callback)` | constructor | `element`: element object or id, `callback`: function to be called when the event happens | Creates an object that will monitor the element. |
| Derived classes have to implement this method to determine the current value being monitored in the element. | | | |
| `registerCallback()` | instance | (none) | This method is typically not called externally. It is called by the object to bind itself to the element's event. |
| `registerFormCallbacks()` | instance | (none) | This method is typically not called externally. It is called by the object to bind itself to the events of each data entry element in the form. |
| `onElementEvent()` | instance | (none) | This method is typically not called externally. It will be bound to the element's event. |

| Property | Type | Description |
|---|---|---|
| `element` | `Object` | The element object that is being monitored. |
| `callback` | `Function(Object, String)` | The function to be called whenever the element changes. It will receive the element object and the new value. |
| `lastValue` | `String` | The last value verified in the element. |

## The `Form.Element.EventObserver` class

*Inherits from* **`Abstract.EventObserver`**

Implementation of an **`Abstract.EventObserver`** that executes a callback function to the appropriate event of the form data entry element to detect value changes in the element. If the element does not expose any event that reports changes, then you can use the **`Form.Element.Observer`** class instead.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| [ctor](element, callback) | constructor | element: element object or id, callback: function to be called when the event happens | Inherited from **`Abstract.EventObserver`**. Creates an object that will monitor the element's **`value`** property. |
| getValue() | instance | (none) | Returns the element's value |

## The `Form.EventObserver` class

*Inherits from* **`Abstract.EventObserver`**

Implementation of an **`Abstract.EventObserver`** that monitors any changes to any data entry element contained in a form, using the elements' events to detect when the value changes. If the form contains elements that do not expose any event that reports changes, then you can use the **`Form.Observer`** class instead.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| [ctor](form, callback) | constructor | form: form object or id, callback: function to be called when any data entry element in the form changes | Inherited from **`Abstract.EventObserver`**. Creates an object that will monitor the form for changes. |
| getValue() | instance | (none) | Returns the serialization of all form's data. |

## The `Position` object (preliminary documentation)

This object provides a host of functions that help when working with positioning.

| Method | Kind | Arguments | Description |
|---|---|---|---|
| prepare() | instance | (none) | Adjusts the **`deltaX`** and **`deltaY`** properties to accommodate changes in the scroll position. Remember to call this method before any calls to **`withinIncludingScrolloffset`** after the page scrolls. |
| realOffset(element) | instance | element: object | Returns an **`Array`** with the correct scroll offsets of the element, including any scroll offsets that affect the element. The resulting array is similar to **`[total_scroll_left, total_scroll_top]`** |
| cumulativeOffset(element) | instance | element: object | Returns an **`Array`** with the correct positioning offsets of the element, including any offsets that are imposed by positioned parent elements. The resulting array is similar to **`[total_offset_left, total_offset_top]`** |

| | | | |
|---|---|---|---|
| `within(element, x, y)` | instance | element: object, x and y: coordinates of a point | Tests if the given point coordinates are inside the bounding rectangle of the given element |
| `withinIncludingScrolloffsets(element, x, y)` | instance | element: object, x and y: coordinates of a point | |
| `overlap(mode, element)` | instance | mode: 'vertical' or 'horizontal', element: object | **`within()`** needs to be called right before calling this method. This method will return a decimal number between 0.0 and 1.0 representing the fraction of the coordinate that overlaps on the element. As an example, if the element is a square DIV with a 100px side and positioned at (300, 300), then **`within(divSquare, 330, 330); overlap('vertical', divSquare);`** should return 0.10, meaning that the point is at the 10% (30px) mark from the top border of the DIV. |
| `clone(source, target)` | instance | source: element object or id, target: element object or id | Resizes and repositions the target element identically to the source element. |

*The documentation for v1.4.0 is still in progress. Stay tuned for updates in this document.*
*If you find errors, inaccurate or incomplete information, or flat-out nonsense, please let me know and I'll try to fix it as soon as possible.*

PDF version created for inclusion in Webucator's Ajax training courses (http://www.webucator.com/WebDesign/JSC401.cfm).