prototype

javascript framework

tutorialspoint

SIMPLYEASYLEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

This tutorial gives a complete understanding on *Prototype. Prototype* is distributed as a single file called *prototype.js*. Prototype is an object in javascript from which other objects inherit properties.

Audience

This tutorial has been written for users willing to learn the Javascript Prototype object and its usage. Beginners as well as experienced users can refer this tutorial to brush up or learn the Javascript prototypes.

Prerequisites

To learn prototype you should have the basic knowledge of Javascript and its properties.

Copyright & Disclaimer

© Copyright 2016 by Tutorials Point (I) Pvt. Ltd.

All the content and graphics published in this e-book are the property of Tutorials Point (I) Pvt. Ltd. The user of this e-book is prohibited to reuse, retain, copy, distribute or republish any contents or a part of contents of this e-book in any manner without written consent of the publisher.

We strive to update the contents of our website and tutorials as timely and as precisely as possible, however, the contents may contain inaccuracies or errors. Tutorials Point (I) Pvt. Ltd. provides no guarantee regarding the accuracy, timeliness or completeness of our website or its contents including this tutorial. If you discover any errors on our website or in this tutorial, please notify us at contact@tutorialspoint.com

Table of Contents

1. Prototype – Overview

What is Prototype ?

Prototype is a JavaScript Framework that aims to ease the development of dynamic web applications. Prototype was developed by Sam Stephenson.

Prototype is a JavaScript library, which enables you to manipulate DOM in a very easy and fun way that is also safe (cross-browser).

Scriptaculous and other libraries, such as *Rico* are build on Prototype's foundations to create widgets and other end-user stuff.

Prototype:

- Extends DOM elements and built-in types with useful methods.
- Has built-in support for class-style OOP including inheritance.
- Has advanced support for event management.
- Has powerful Ajax features.
- Is not a complete application development framework.
- Does not provide widgets or a full set of standard algorithms or I/O systems.

How to Install Prototype?

Prototype is distributed as a single file called prototype.js. Follow the below mentioned steps to setup the prototype library:

- Go to the download page (http://prototypejs.org/download/) to grab the latest version in a convenient package.
- Now, put prototype.js file in a directory of your website, e.g. /javascript.

You are now ready to use the powerful Prototype framework in your web pages.

How to Use Prototype Library?

Now, you can include the *Prototype* script as follows:

```
<html>

<head>
```

```
<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

</head>

<body>

........


</body>

</html>
```

Example

Here is a simple example showing how you can use Prototype's $() function to get DOM elements in your JavaScript:

```
<html>

<head>


<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

   <script>

   function test(){

       node = $("firstDiv");

       alert(node.innerHTML);

   }

   </script>

```

```
    </head>


    <body>

       <div id="firstDiv">

          <p>This is first paragraph</p>

       </div>

       <div id="secondDiv">


          <p>This is another paragraph</p>

       </div>


       <input type="button" value="Test $()" onclick="test();"/>


    </body>

    </html>
```

Why This Tutorial?

A very good documentation for Prototype Framework is available at prototypejs.org then why should one refer to this tutorial!

The answer is that we have put all the most commonly used functionalities together in this tutorial. Secondly, we have explained all the useful methods along with suitable examples, which are not available at the official site.

If you are an advanced user of Prototype Framework, then you can directly jump to the official website, otherwise this tutorial could be a good start for you and you can use it like a reference manual.

2. Prototype – Useful Features

Let's now look at what Prototype can do specifically for us to develop a Dynamic Web Application.

Cross Browser Support

While doing JavaScript programming, it is required to handle different Web Browsers differently. Prototype Library has been written in such a way that it takes care of all the compatibility issues and you can do cross browser programming without any hassle.

The Document Object Model

Prototype provides helper methods that ease some of the strain of DOM programming. Using Prototype, you can manipulate DOM very easily.

HTML Forms

With Ajax, other input mechanisms such as drag and drop, can be used as part of a conversation between the browser and the server. With conventional JavaScript programming, it is difficult to capture these inputs and pass them to the server. Prototype provides a set of utilities for working with HTML forms.

JavaScript Events

Prototype provides some excellent cross-browser support while coding events, and also extends the Function object to make it easy to work with event handling.

Ajax Utilities

The most important feature of Prototype is it's support for Ajax. All major browsers support a version of the XMLHttpRequest object that makes Ajax possible, either as an ActiveX component or as a native JavaScript object.

XMLHttpRequest, however, exposes the HTTP protocol at a very low level, which gives the developer a lot of power, but also requires her to write a lot of code in order to do simple things.

Prototype uses it's own object inheritance system to provide a hierarchy of Ajax helper objects, with more generic base classes being subclassed by more focused helpers that allow the most common types of Ajax request to be coded in a single line.

3. Prototype – Utility Methods

The Prototype library comes with lot of predefined objects and utility functions. You can use those functions and objects directly in your JavaScript programming.

These methods are one of the cornerstones of efficient Prototype-based JavaScript coding. Spend some time to study them to become comfortable with the methods.

This chapter details all these useful methods with examples.

Method	Description
$()	If provided with a string, returns the element in the document with matching ID; otherwise returns the passed element.
$$()	Takes an arbitrary number of CSS selectors (strings) and returns a document-order array of extended DOM elements that match any of them.
$A()	Converts the single argument it receives into an Array object.
$F()	Returns the value of a form control. This is a convenience alias of Form.Element.getValue.
$H()	Converts objects into enumerable Hash objects that resemble associative arrays.
$R()	Creates a new ObjectRange object.
$w()	Splits a string into an Array, treating all whitespace as delimiters.
Try.these	Accepts an arbitrary number of functions and returns the result of the first one that doesn't throw an error.

$() Method

The most commonly used and convenient function, $(), provides an easy way of getting a handle on a DOM element.

Syntax

```
$(id | element)


OR


$((id | element)...)
```

Return Value

- Found HTMLElement.
- In case it finds more than one elements, then it returns array of HTML elements.

Here is an old way of writing Javascript statement to get a node of DOM.

```
node = document.getElementById("elementID");
```

Using $(), we can shorten it up as follows:

```
node = $("elementID");
```

Example

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

   <script>

   function test(){

       node = $("firstDiv");

       alert(node.innerHTML);
```

```
      }
   </script>
</head>

<body>
   <div id="firstDiv">
      <p>This is first paragraph</p>
   </div>
   <div id="secondDiv">
      <p>This is another paragraph</p>
   </div>

   <input type="button" value="Test $()" onclick="test();"/>

</body>
</html>
```

Fetching Multiple Values Using $()

The $() function is also more powerful than **document.getElementById()** because the ability to retrieve multiple elements is built into the function.

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.

Example

In this example, we see the $() function now returning an array of our elements, which can then be accessed with a simple **for** loop.

```
<html>
<head>
<title>Prototype examples</title>
```

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

   src="/javascript/prototype.js">

   </script>

   <script>

   function test(){

       allNodes = $("firstDiv", "secondDiv");

       for(i = 0; i < allNodes.length; i++) {

         alert(allNodes[i].innerHTML);

       }

   }

   </script>

</head>


<body>

   <div id="firstDiv">

      <p>This is first paragraph</p>

   </div>

   <div id="secondDiv">

      <p>This is another paragraph</p>

   </div>


   <input type="button" value="Test $()" onclick="test();"/>


</body>

</html>
```

$$() Method

The $$() method parses one or more CSS filtering expressions, analogous to the ones used to define CSS rules, and returns the elements that match these filters.

Syntax

```
$$(cssRule...);
```

Return Value

- An array of HTML elements.

Example

Here is an old way of writing Javascript statement to get all the nodes of DOM with name div.

```
nodes = document.getElementsByTagName('div');
```

Using $$(), we can shorten it up as follows:

```
nodes = $$('div');
```

Following is same as $('contents'), only it returns an array anyway.

```
$$('#contents');
```

Example

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

   <script>
```

```
      function test(){

          allNodes = $$("div");

          for(i = 0; i < allNodes.length; i++) {

            alert(allNodes[i].innerHTML);

          }

       }

       </script>

    </head>


    <body>

       <div id="firstDiv" name="div">

          <p>This is first paragraph</p>

       </div>


       <div id="secondDiv" name="div">

          <p>This is another paragraph</p>

       </div>


       <input type="button" value="Test $()" onclick="test();"/>


    </body>

    </html>
```

More Examples

Following returns all links inside the element of ID "contents" with a rel attribute.

```
$$('#contents a[rel]');
```

Following returns all links with a href attribute of value "#" (eyeew!).

```
$$('a[href="#"]');
```

Following returns all links within the elements of ID "navbar" or "sidebar".

```
$$('#navbar a', '#sidebar a');
```

Following returns all links, excluding those whose rel attribute contains the word "nofollow".

```
$$('a:not([rel~=nofollow])');
```

Following returns all even rows within all table bodies.

```
$$('table tbody > tr:nth-child(even)');
```

Following returns all DIVs without content (i.e., whitespace-only).

```
$$('div:empty');
```

$A() Method

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

One suggested use is to convert DOM NodeLists into regular arrays, which can be traversed more efficiently.

Syntax

```
$A(iterable)
```

Return Value

- List of elements in the form of array .

Example

```
<html>

<head>

<title>Prototype examples</title>
```

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

function showOptions(){
   var NodeList = $('employees').getElementsByTagName('option');
   var nodes = $A(NodeList);

   nodes.each(function(node){
      alert(node.nodeName + ': ' + node.innerHTML);
   });
}
</script>
</head>

<body>

   <select id="employees" size="10" >
      <option value="5">Mohtashim, Mohd</option>
      <option value="8">Debi, Patnaik</option>
      <option value="1">Madisetti, Praveen</option>
   </select>
   <br />

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

```
</body>


</html>
```

$F() Method

The $F() function returns the value of any field input control, like text boxes or drop-down lists. This is a convenience alias of *Form.Element.getValue*.

The function can take as argument either the element id or the element object itself.

Syntax

```
$F(element)
```

Return Value

- Form's element Value.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function ShowValue(){

      var value = $F('userName');


      alert( 'Entred  Value :' + value);
}
</script>
</head>

<body>

   <p>Enter any value in the box and then click "Show Value"</p>

   <form>
   <input type="text" id="userName" value="Madisetti, Praveen">
   <br/>

   <input type="button" value="Show Value" onclick="ShowValue();"/>
   </form>

</body>
</html>
```

$H() Method

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

The $H function is the shorter way to obtain a hash.

Syntax

```
$H([obj])
```

Return Value

- A hash object.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function ShowHash()

{

   //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());


}

</script>

</head>


<body>
```

```
    <p>Click "Show Value" button to see the result</p>


    <form>

    <input type="button" value="Show Value" onclick="ShowHash();"/>

    </form>


</body>

</html>
```

This will display the following result:

```
first=10&second=20&third=30
```

$R() Method

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

Syntax

```
$R(start, end[, exclusive = false]);
```

Here, *start* is the starting element of the range and *end* is the last element of the range. If *exclusive* flag is set to false, then it will include the ending elements, otherwise it will not be included in the range.

Return Value:

- Range Object.

Example

```
<html>

<head>

<title>Prototype examples</title>
```

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

function ShowValues()
{
   var range = $R(10, 20, false);
   range.each(function(value, index){
         alert(value);
   });

}
</script>
</head>

<body>

   <p>Click "Show Value" button to see the result</p>

   <form>
   <input type="button" value="Show Value" onclick="ShowValues();"/>
   </form>

</body>
</html>
```

More Examples

Following statement returns *true* value:

```
$R(0, 10).include(10);
```

Following statement returns a string "0, 1, 2, 3, 4, 5":

```
$A($R(0, 5)).join(', ');
```

Following statement returns a string "aa, ab, ac, ad, ae, af, ag, ah":

```
$A($R('aa', 'ah')).join(', ');
```

Following statement returns *false*:

```
$R(0, 10, true).include(10);
```

Following statement will be invoked 10 times for value = 0 to 9:

```
$R(0, 10, true).each(function(value) {

  // invoked 10 times for value = 0 to 9
```

$w() Method

The $w() function splits a string into an Array, treating all whitespace as delimiters.

Syntax

```
$w(String);
```

Return Value

- An array of strings.

Example

```
<html>

<head>
```

```
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function ShowValues()
{
   var str = "Apples Bananas Kiwis";

   // Convert string into Array
   var arr = $w(str);

   arr.each(function(value){
         alert(value);
   });

}
</script>
</head>

<body>

   <p>Click "Show Value" button to see the result</p>

   <form>
   <input type="button" value="Show Value" onclick="ShowValues();"/>
```

```
      </form>



   </body>

   </html>
```

Try.these Method

The Try.these() function makes it easy when you want to 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.

If none of the blocks succeeded, Try.these will return undefined, i.e., false.

Syntax

```
   Try.these(Function...);
```

Return Value

- First OK result.

Example

There are different ways to create XMLHttp object in different browsers. Using the Try.these() function we can return the one that works.

```
   <html>

   <head>

   <title>Prototype examples</title>

   <script type="text/javascript"

      src="/javascript/prototype.js">

   </script>

   <script>



   createXMLHttp: function()
```

```
{
   return Try.these(
    function() { return new XMLHttpRequest() },
    function() { return new ActiveXObject('Msxml2.XMLHTTP') },
    function() { return new ActiveXObject('Microsoft.XMLHTTP') }
  ) || false;


}
</script>
</head>


<body>
......
</body>
</html>
```

If none of the blocks succeeded, Try.these will return undefined, which will cause the createXMLHttp method in the example above to return false, provided as a fallback result value.

document.getElementsByClassName

This method retrieves (and extends) all the elements that have a CSS class name of *className*.

However, this method has been deprecated in the latest versions of Prototype.

4. Prototype – Element Object

The *Element* object provides various utility functions for manipulating elements in the DOM. Here is a list of all the utility functions with examples. All the methods defined here are automatically added to any element accessed using the $() function.

So, writing `Element.show('firstDiv');` is the same as writing `$('firstDiv').show();`

Prototype Element Method

NOTE: Make sure you have at least version 1.6 of prototype.js.

Method	Description
absolutize()	Turns element into an absolutely-positioned element without changing its position in the page layout.
addClassName()	Adds the given CSS class name to the element's class names.
addMethods()	Makes it possible to mix in your own methods to the Element object, which you can later use as methods of extended elements.
adjacent()	Finds all siblings of the current element that match the given selector(s).
ancestors()	Collects all of element's ancestors and returns them as an array of extended elements.
childElements()	Collects all of the element's children and returns them as an array of extended elements.
classNames()	Deprecated. Returns a new instance of ClassNames, an Enumerable object used to read and write CSS class names of element.
cleanWhitespace()	Removes all of element's text nodes, which contain only whitespace. Returns element.

clonePosition()	Clones the position and/or dimensions of source onto element as defined by the optional argument options.
cumulativeOffset()	Returns the offsets of element from the top left corner of the document.
cumulativeScrollOffset()	Calculates the cumulative scroll offset of an element in nested scrolling containers.
descendantOf()	Checks if the element is a descendant of ancestor.
descendants()	Collects all of element's descendants and returns them as an array of extended elements.
down()	Returns element's first descendant that matches cssRule. If no cssRule is provided, all descendants are considered. If no descendant matches these criteria, undefined is returned.
empty()	Tests whether element is empty (i.e., contains only whitespace).
extend()	Extends element with all of the methods contained in Element.Methods and Element.Methods.Simulated.
fire()	Fires a custom event with the current element as its target.
firstDescendant()	Returns the first child that is an element. This is opposed to firstChild DOM property, which will return any node.
getDimensions()	Finds the computed width and height of an element and returns them as key/value pairs of an object.
getElementsByClassName	Deprecated. Fetches all of element's descendants, which have a CSS class of className and returns them as an array of extended elements. Please use $$().
getElementsBySelector	Deprecated. Takes an arbitrary number of CSS selectors (strings) and returns an array of extended children of element that match any of them. Please use $$().

getHeight()	Finds and returns the computed height of element.
getOffsetParent()	Returns element's closest positioned ancestor. If none is found, the body element is returned.
getStyle()	Returns the given CSS property value of element. Property can be specified in either of its CSS or camelized form.
getWidth()	Finds and returns the computed width of element.
hasClassName()	Checks whether element has the given CSS className.
hide()	Hides and returns element.
identify()	Returns element's id attribute if it exists, or sets and returns a unique, auto-generated id.
immediateDescendants()	Deprecated. Collects all of the element's immediate descendants (i.e., children) and returns them as an array of extended elements. Please use childElements().
insert()	Inserts content before, after, at the top of, or at the bottom of element.
inspect()	Returns the debug-oriented string representation of element.
makeClipping()	Simulates the poorly supported CSS clip property by setting element's overflow value to 'hidden'. Returns element.
makePositioned()	Allows for the easy creation of CSS containing block by setting element's CSS position to 'relative' if its initial position is either 'static' or undefined. Returns element.
match()	Checks if element matches the given CSS selector.
next()	Returns element's following sibling that matches the given cssRule.

nextSiblings()	Collects all of element's next siblings and returns them as an array of extended elements.
observe()	Registers an event handler on element and returns element.
positionedOffset ()	Returns element's offset relative to its closest positioned ancestor.
previous ()	Returns element's previous sibling that matches the given cssRule.
previousSiblings ()	Collects all of element's previous siblings and returns them as an array of extended elements.
readAttribute ()	Returns the value of element's attribute or null if attribute has not been specified.
recursivelyCollect ()	Recursively collects elements whose relationship is specified by property.
relativize ()	Turns element into an relatively-positioned element without changing its position in the page layout.
remove ()	Completely removes element from the document and returns it.
removeClassName ()	Removes element's CSS className and returns element.
replace ()	Replaces element by the content of the html argument and returns the removed element.
scrollTo ()	Scrolls the window so that element appears at the top of the viewport. Returns element.
select ()	Takes an arbitrary number of CSS selectors (strings) and returns an array of extended descendants of element that match any of them.

setOpacity ()	Sets the visual opacity of an element while working around inconsistencies in various browsers.
setStyle ()	Modifies element's CSS style properties.
show ()	Displays and returns element.
siblings ()	Collects all of element's siblings and returns them as an array of extended elements.
stopObserving ()	Unregisters handler and returns element.
toggle ()	Toggles the visibility of element.
toggleClassName ()	Toggles element's CSS className and returns element.
undoClipping ()	Sets element's CSS overflow property back to the value it had before Element.makeClipping() was applied. Returns element.
undoPositioned ()	Sets element back to the state it was before Element.makePositioned was applied to it. Returns element.
up()	Returns element's first ancestor that matches the given cssRule.
update()	Replaces the content of element with the provided newContent argument and returns element.
viewportOffset()	Returns the X/Y coordinates of element relative to the viewport.
visible()	Returns a Boolean indicating whether or not element is visible.
wrap()	Wraps an element inside another, then returns the wrapper.
writeAttribute()	Adds, specifies or removes attributes passed as either a hash or a name/value pair.

absolutize() Method

This method turns element into an absolutely-positioned element without changing its position in the page layout.

Syntax

```
element.absolutize();
```

Return Value

- An absolutely-positioned HTML element.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function changetoAbs(){

   node = $("firstDiv");

   node.style.fontSize='20px';

   //node.style.position = 'absolute';

   node.absolutize();

   node.style.border = '1px dashed #f00';

   node.style.left = '100px';

}

</script>
```

```
</head>


<body>
   <div id="firstDiv">
      <p>This is first paragraph</p>
   </div>


   <br />
   <input type="button" value="Make Absolute"
              onclick="changetoAbs();"/>


</body>
</html>
```

addClassName() Method

This method Adds a CSS class to element.

Syntax

```
element.addClassName(className);
```

Return Value

- An HTML element in which CSS class is added.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
```

```
   src="/javascript/prototype.js">
</script>
<script>

function addClass(){
   node = $("firstDiv");
   node.addClassName("title");
}
</script>
</head>

<style type="text/css">
.title{
    color:#36C;
    font-size:20px;
}
</style>
<body>
   <div id="firstDiv">
      <p>This is first paragraph</p>
   </div>

   <br />
   <input type="button" value="Add Class"
              onclick="addClass();"/>

</body>
```

```
</html>
```

addMethods() Method

This method makes it possible to mix in your own methods to the Element object, which you can later use as methods of extended elements.

To add new methods, simply feed Element.addMethods with a hash of methods. Note that each method's first argument has to be an element.

Syntax

```
element.addMethods([hash of methods]);


OR


element.addMethods(tagName, methods);
```

Here, second form of the method will make added method available for a particular tag only.

Return Value

- None.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>
```

```
// Make changeColor method available for all the elements

Element.addMethods({

    changeColor: function(element, colorName) {

    element = $(element);

    element.style.color = colorName;

    return element;

  }

});



function ShowEffect(){

   node = $("firstDiv");

   // Now call changeColor method

   node.changeColor( "red" );

}


</script>

</head>


<body>

   <div id="firstDiv">

      <p>This is first paragraph</p>

   </div>


   <br />

   <input type="button" value="ShowEffect"

             onclick="ShowEffect();"/>
```

```
</body>

</html>
```

adjacent() Method

This method finds all siblings of the current element that matches the given selector(s) and returns them as an array.

Syntax

```
element.adjacent([ selectors...]);
```

Return Value

- An array of HTML elements.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function listCities(){

   var arr = $('nyc').adjacent('li.us');


   arr.each(function(node){

      alert(node.nodeName + ': ' + node.innerHTML);
   });


}


</script>

</head>


<body>

   <ul id="cities">

    <li class="us" id="nyc">New York</li>

    <li class="uk" id="lon">London</li>

    <li class="us" id="chi">Chicago</li>

    <li class="jp" id="tok">Tokyo</li>

    <li class="us" id="la">Los Angeles</li>

    <li class="us" id="aus">Austin</li>

  </ul>


   <br />

   <input type="button" value="List Cities"

             onclick="listCities();"/>

</body>

</html>
```

ancestors() Method

This method collects all of element's ancestors and returns them as an array of extended elements.

Keep in mind that the body and html elements will also be included.

Syntax

```
element.ancestors();
```

Return Value

- An array of HTML elements.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>



function showElements(){
   var arr =  $('kid').ancestors();
   arr.each(function(node){
      alert(node.nodeName + ': ' + node.innerHTML);
   });

}

</script>
</head>
```

```
<body>
   <div id="father">
      <p id="kid">This is first paragraph</p>
   </div>

   <br />
   <input type="button" value="showElements"
             onclick="showElements();"/>

</body>
</html>
```

childElements() Method

Collects all of the element's children and returns them as an array of extended elements.

An index of 0 refers to the topmost child of an element.

Syntax

```
element.childElements();
```

Return Value

- An array of HTML elements.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
```

```
</script>

<script>



function showElements(){

   var arr =  $('father').childElements();

   arr.each(function(node){

      alert(node.nodeName + ': ' + node.innerHTML);

   });


}


</script>

</head>


<body>

   <div id="father">

      <p id="kid1">This is first paragraph</p>

    <p id="kid2">This is second paragraph</p>

   </div>


   <br />

   <input type="button" value="showElements"

             onclick="showElements();"/>


</body>

</html>
```

cleanWhitespace() Method

This method removes all of element's text nodes, which contain only whitespace and returns element.

Element.cleanWhitespace removes whitespace-only text nodes. This can be very useful when using standard methods like *nextSibling, previousSibling, firstChild* or *lastChild* to walk the DOM.

Syntax

```
element.cleanWhitespace();
```

Return Value

- An HTML element

Example

Consider the following example:

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>



function showElements(){

   var element = $('apples');

   alert(element.firstChild.innerHTML);

}
```

```
</script>

</head>


<body>

  <ul id="apples">

     <li>Mutsu</li>

     <li>McIntosh</li>

     <li>Ida Red</li>

  </ul>


   <br />

   <input type="button" value="showElements"

             onclick="showElements();"/>


</body>

</html>
```

That doesn't seem to work to well. Why is that ? ul#apples's first child is actually a text node containing only whitespace that sits between <ul id="apples"> and <li>Mutsu</li>.

Now, let's use cleanWhitespace function and see the result:

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>
```

```
function showElements(){
   var element = $('apples');
   element.cleanWhitespace();
   alert(element.firstChild.innerHTML);
}

</script>
</head>

<body>
  <ul id="apples">
     <li>Mutsu</li>
     <li>McIntosh</li>
     <li>Ida Red</li>
  </ul>

   <br />
   <input type="button" value="showElements"
             onclick="showElements();"/>

</body>
</html>
```

This will display the following result:

```
'Mutsu'
```

clonePostion() Method

This method clones the position and/or dimensions of source into element as defined by the optional argument options.

Syntax

```
element.clonePosition(source[, options]);
```

Here is the list of possible options:

Name	Default	Description
setLeft	true	clones source's left CSS property onto element.
setTop	true	clones source's top CSS property onto element.
setWidth	true	clones source's width onto element.
setHeight	true	clones source's width onto element.
offsetLeft	0	Number by which to offset element's left CSS property.
offsetTop	0	Number by which to offset element's top CSS property.

Return Value

- An HTML element.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>
```

tutorialspoint
SIMPLYEASYLEARNING

```
<script>



function clonePosition(){

   var firstElement = $('firstDiv');

   var secondElement = $('secondDiv');

   secondElement.clonePosition( firstElement);

}


</script>

</head>


<body>


   <p>Click Clone Position button to see the result.</p>

   <div id="firstDiv">

      <p>This is first paragraph</p>

   </div>


   <div id="secondDiv">

      <p>This is second paragraph</p>

   </div>


   <br />

   <input type="button" value="Clone Position"

             onclick="clonePosition();"/>
```

```
</body>

</html>
```

cumulativeOffset() Method

This method returns the offsets of element from the top left corner of the document.

This method returns an array keeping offsetLeft and offsetTop of the element.

Note that all values are returned as numbers, only although they are expressed in pixels.

Syntax

```
element.cumulativeOffset();
```

Return Value

- An array of two numbers [offset left, offset top].

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>



function getOffset(){

   var firstElement = $('firstDiv');

   var arr = firstElement.cumulativeOffset();

   alert ( "Offset Left: " +arr[0]+ " Offset Top : " +arr[0] );
}


</script>
</head>


<body>


   <p>Click getOffset button to see the result.</p>
   <div id="firstDiv">
      <p>This is first paragraph</p>
   </div>


   <br />
   <input type="button" value="getOffset"
             onclick="getOffset();"/>


</body>
</html>
```

cumulativeScrollOffset() Method

This method calculates and returns the cumulative scroll offset of an element in nested scrolling containers. This adds the cumulative scrollLeft and scrollTop of an element and all its parents.

This is used for calculating the scroll offset of an element that is in more than one scroll container (e.g., a draggable in a scrolling container, which is itself part of a scrolling document).

This method returns an array keeping offsetLeft and offsetTop of the element.

Syntax

```
element.cumulativeScrollOffset();
```

Return Value

- An array of two numbers [offset let, offset top].

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>




function getOffset(){

   firstElement = $('firstDiv');

   var arr = firstElement.cumulativeScrollOffset();

   alert ( "Offset Left: " +arr[0]+ " Offset Top : " +arr[0]);

}


</script>

</head>


<body>
```

```
      <p>Click getOffset button to see the result.</p>

      <div id="firstDiv">

         <p>This is first paragraph</p>

      </div>


      <br />

      <input type="button" value="getOffset"

                onclick="getOffset();"/>


   </body>

   </html>
```

descendantOf() Method

This method checks if element is a descendant of ancestor.

As Element.descendantOf internally applies $() to ancestor, it accepts indifferently an element or an element's id as its second argument.

Syntax

```
   element.descendantOf(ancestor);
```

Return Value

- If it finds that element is a decendant of an ancestor, then it returns true, otherwise false.

Example

```
   <html>

   <head>

   <title>Prototype examples</title>

   <script type="text/javascript"
   src="/javascript/prototype.js">

</script>

<script>



function isDescendant(){

   var father = $('father');

   var kid = $('kid');

   // This is correct relationship and will be printed

   if( kid.descendantOf(father) ){

       alert( "Kid is descendant of father" );

   }

   // This is wrong relationship and will not  be printed

   if( father.descendantOf(kid) ){

       alert( "Father is descendant of kid" );

   }

}


</script>

</head>


<body>


   <p>Click isDescendant button to see the result.</p>

   <div id="grandfather">

      <div id="father">

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

```
          </div>

       </div>


       <br />

       <input type="button" value="isDescendant"

                 onclick="isDescendant();"/>


    </body>

    </html>
```

descendants() Method

This method collects all of element's descendants and returns them as an array of extended elements.

Note that all of Prototype's DOM traversal methods ignore text nodes and return element nodes only.

Syntax

```
element.descendants() ;
```

Return Value

- An array of HTML elements.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>
```

```
<script>



function showElements(){

   var arr = $('father').descendants();

   arr.each(function(node){

      alert(node.nodeName + ': ' + node.innerHTML);

   });

}


</script>

</head>


<body>


   <p>Click descendants button to see the result.</p>


   <div id="father">

      <p id="kid">This is first paragraph</p>

   </div>


   <br />

   <input type="button" value="descendants"

             onclick="showElements();"/>


</body>

</html>
```

down() Method

This method returns element's first descendant or the n-th descendant if index is specified that matches *cssRule*.

If no *cssRule* is provided, all descendants are considered. If no descendant matches these criteria, undefined is returned.

Syntax

```
element.down([cssRule][, index = 0]);
```

Return Value

- Found HTMLElement.
- Undefined in case it does not find any element.

NOTE: element.down() and element.down(0) are equivalent.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   alert( "1 : " + $('fruits').down(3).innerHTML );

   alert( "2 : " + $('apples').down('li').innerHTML );

   alert( "3 : " + $('apples').down('li.yummy').innerHTML );

   alert( "4 : " + $('fruits').down('.yummy', 1).innerHTML );

   alert( "5 : " + $('fruits').down(99) );
```

tutorialspoint
SIMPLYEASYLEARNING

```
}


</script>

</head>


<body>


   <p>Click the button to see the result.</p>


   <ul id="fruits">

   <li id="apples">

    <ul>

      <li id="golden-delicious">Golden Delicious</li>

      <li id="mutsu" class="yummy">Mutsu</li>

      <li id="mcintosh" class="yummy">McIntosh</li>

      <li id="ida-red">Ida Red</li>

    </ul>

   </li>

   </ul>


   <br />

   <input type="button" value="showResult"

             onclick="showResult();"/>


</body>

</html>
```

empty() Method

This method tests whether element is empty (i.e., contains only whitespace).

Syntax

```
element.empty;
```

Return Value

- If it finds that element is an empty one, then it returns true, otherwise false.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){
   if($('wallet').empty() ){
       alert( "Wallet is empty " );
   }
   if($('cart').empty() ){
       alert( "Cart is full" );
   }
}
</script>
</head>
```

```
<body>


   <p>Click the button to see the result.</p>


   <div id="wallet">     </div>

   <div id="cart">full!</div>


   <br />

   <input type="button" value="showResult"

             onclick="showResult();"/>


</body>

</html>
```

In this example:

```
$('wallet').empty();

// -> true

$('cart').empty();

// -> false
```

extend() Method

This method extends element with all of the methods contained in Element.Methods and Element.Methods.Simulated.

If element is an input, textarea or select tag, it will also be extended with the methods from*Form.Element.Methods*.

If it is a form tag, it will also be extended with *Form.Methods*.

Syntax

```
element.extend();
```

Return Value

- None.

Example

By extending an element with Prototype's custom methods, we can achieve that syntactic sugar and ease of use we all crave for. For example, you can do the following with an extended element:

```
element.update('hello world');
```

And since most methods of Element return the element they are applied to, you can chain methods like so:

```
element.update('hello world').addClassName('greeting');
```

Note that all of the elements returned by Element methods are extended (yes even for methods like **Element.siblings**, which return arrays of elements) and Prototype's flagship utility methods $() and $$() obviously also return extended elements.

fire() Method

This method is used to fire a custom event with the current element as its target.

The custom event has all the same properties and methods of native events. Like a native event, it will bubble up through the DOM unless its propagation is explicitly stopped.

Custom events are dispatched synchronously: Element#fire waits until the event finishes its life cycle, then returns the event itself.

Syntax

```
element.fire(eventName[, memo]);
```

The optional second argument will be assigned to the **memo** property of the event object so that it can be read by event handlers.

Return Value

- It returns custom event.

Example

In this example, an element with ID (firstDiv) frobbed widget #19.

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

document.observe("widget:frobbed", function(event) {
   alert("Element with ID (" + event.target.id +
   ") frobbed widget #" + event.memo.widgetNumber + ".");
});

function showResult(){
   someNode = $('firstDiv');
   someNode.fire("widget:frobbed", { widgetNumber: 19 });
}
</script>
</head>

<body>

   <p>Click the button to see the result.</p>
```

```
      <div id="firstDiv">
         <p>This is first paragraph</p>
      </div>

      <br />
      <input type="button" value="showResult"
                onclick="showResult();"/>

   </body>
   </html>
```

firstDescendant() Method

This method returns the first child that is an element. This is opposed to *firstChild* DOM property, which will return any node.

Syntax

```
element.firstDescendant();
```

Return Value

- It returns an HTML element.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
```

```
</script>
<script>

function showResult(){
   desc = $('grandfather').firstDescendant();
   alert("First Descendant is : "
          + desc.nodeName + ':' + desc.innerHTML );

   child = $('grandfather').firstChild;
   alert("First Child is : " + child.toString() );
}
</script>
</head>

<body>

   <p>Click the button to see the result.</p>

   <div id="grandfather">This is grand father
      <div id="father"> This is father.
         <div id="kid">This is kid</div>
      </div>
   </div>

   <br />
   <input type="button" value="showResult"
             onclick="showResult();"/>
```

```
</body>

</html>
```

getDimensions() Method

This method finds the computed width and height of element and returns them as key/value pairs of an object.

This method returns the correct values on elements whose display is set to none either in an inline style rule or in an CSS stylesheet.

Note that the value returned is a number, only although it is expressed in pixels.

Syntax

```
element.getDimensions();
```

Return Value

- It returns key/value pairs of an object {height: Number, width: Number}.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   var dimensions = $('rectangle').getDimensions();

   alert("Element width is " +  dimensions.width );
```

```
      alert("Element height is " +  dimensions.height );

   }

   </script>

   </head>


   <body>


      <p>Click the button to see the result.</p>


      <div id="rectangle"

              style="font-size: 10px; width: 20em; height: 10em">

           <p>This is the paragraph.</p>

      </div>


      <br />

      <input type="button" value="showResult"

               onclick="showResult();"/>


   </body>

   </html>
```

getHeight() Method

This method finds and returns the computed height of element.

This method returns correct values on elements whose display is set to none either in an inline style rule or in an CSS stylesheet.

Note that the value returned is a number only although it is expressed in pixels.

Syntax

```
element.getHeight();
```

Return Value

- It returns the computed height of element.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   var height = $('rectangle').getHeight();

   alert("Element height is " +  height );

}

</script>

</head>


<body>


   <p>Click the button to see the result.</p>


   <div id="rectangle"
```

```
               style="font-size: 10px; width: 20em; height: 10em">
            <p>This is the paragraph.</p>
       </div>

       <br />
       <input type="button" value="showResult"
                 onclick="showResult();"/>

    </body>
    </html>
```

getOffsetParent() Method

This method returns the element's closest positioned ancestor. If none is found, the body element is returned.

Syntax

```
element.getOffsetParent();
```

Return Value

- It returns an HTML element.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
```

```
<script>


function showResult(){
   var node = $('father').getOffsetParent();
   alert("Offset Parent is : "
      + node.nodeName + ':' + node.innerHTML );
 }
</script>
</head>


<body>


  <p>Click the button to see the result.</p>


   <div id="grandfather">This is grand father
      <div id="father"> This is father.
         <div id="kid">This is kid</div>
      </div>
   </div>


   <br />
   <input type="button" value="showResult"
             onclick="showResult();"/>


</body>
</html>
```

getStyle() Method

This method returns the given CSS property value of element. Property can be specified in either of its CSS or camelized form.

Thus providing *font-size* and *fontSize* are equivalent.

Syntax

```
element.getStyle( property );
```

Return Value

- Either CSS property value or NULL if it does not find a property set. Internet Explorer returns literal values while other browsers return computed values. Safari returns null for any non-inline property if the element is hidden.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){
   var str = $('grandfather').getStyle('margin-left');
   alert("Element left margin is : " + str );
 }
</script>
<style>
  #grandfather {
    font-size: 12px;

    margin-left: 1em;

  }

</style>

</head>


<body>


  <p>Click the button to see the result.</p>


   <div id="grandfather">This is grand father

      <div id="father"> This is father.

         <div id="kid">This is kid</div>

      </div>

   </div>


   <br />

   <input type="button" value="showResult"

             onclick="showResult();"/>


</body>

</html>
```

getWidth() Method

This method finds and returns the computed width of element.

This method returns correct values on elements whose display is set to none either in an inline style rule or in an CSS stylesheet.

Note that the value returned is a number only although it is expressed in pixels.

Syntax

```
element.getWidth();
```

Return Value

- It returns the computed width of an element.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   var width = $('rectangle').getWidth();

   alert("Element width is " +  width );

}

</script>

</head>


<body>


   <p>Click the button to see the result.</p>


   <div id="rectangle"
```

```
               style="font-size: 10px; width: 20em; height 10em">
         <p>This is the paragraph.</p>
    </div>

    <br />
    <input type="button" value="showResult"
              onclick="showResult();"/>

 </body>
 </html>
```

hasClassName() Method

This method checks whether element has the given CSS className.

Syntax

```
element.hasClassName( className );
```

Return Value

- If element has the given class name, then it returns true, otherwise false.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>
```

```
function showResult(){

   if( $('grandfather').hasClassName("test") ){
      alert("Grandfather has CSS test class.");
   }
   if( $('father').hasClassName("test") ){
      alert("Father has CSS test class.");
   }
   if( $('kid').hasClassName("test") ){
      alert("Kid has CSS test class.");
   }
 }
</script>
<style>
.test {
    font-size: 12px;
    margin-left: 1em;
  }
</style>
</head>

<body>

  <p>Click the button to see the result.</p>

   <div id="grandfather" class="test">This is grand father
```

```
          <div id="father"> This is father.

             <div id="kid" class="test">This is kid</div>

          </div>

       </div>


       <br />

       <input type="button" value="showResult"

                 onclick="showResult();"/>


    </body>

    </html>
```

hide() Method

This method hides and returns element.

Syntax

```
    element.hide();
```

Return Value

- Hides and returns HTML element.

Example

```
    <html>

    <head>

    <title>Prototype examples</title>

    <script type="text/javascript"

       src="/javascript/prototype.js">

    </script>
```

```
<script>


function hideElement(){


   $('grandfather').hide();

}

function showElement(){


   $('grandfather').show();

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>


   <div id="grandfather" >This is grand father

      <div id="father"> This is father.

         <div id="kid" >This is kid</div>

      </div>

   </div>


   <br />

   <input type="button" value="Hide" onclick="hideElement();"/>

   <input type="button" value="Show" onclick="showElement();"/>
```

```
</body>

</html>
```

identify() Method

This method returns element's id attribute if it exists, or sets and returns a unique, auto-generated id.

Syntax

```
element.identify();
```

Return Value

- Returns HTML element ID.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   var str = $('apple').identify();

   alert("Apple's ID: " + str );

   var str = $('apple').next().identify()

   alert("Orange's ID: " + str );
}
</script>
</head>


<body>


  <p>Click the button to see the result.</p>


  <ul>
   <li id="apple">apple</li>
   <li>orange</li>
  </ul>


  <br />
  <input type="button" value="Click" onclick="showResult();"/>


</body>
</html>
```

insert() Method

This method inserts content before, after, at the top of, or at the bottom of element, as specified by the position property of the second argument. If the second argument is the content itself, insert will **append** it to element.

Insert accepts the following kind of content:

- text
- HTML
- DOM element
- Any kind of object with a toHTML or toElement method.

NOTE: Note that if the inserted HTML contains any <script> tag, these will be automatically evaluated after the insertion.

Syntax

```
element.insert({ position: content });


OR


element.insert(content)
```

Return Value

- Returns HTML element after inserted content.

Example

```
<html>

<head>

<title>Prototype examples</title>



<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>



function showResult(){

   var str = $('apple').insert(  "<li>mangoes</li>" );

   alert(str.innerHTML );

}

</script>
```

```
</head>


<body>


  <p>Click the button to see the result.</p>


  <ul>
   <li id="apple">apple</li>
   <li>orange</li>
  </ul>


  <br />
  <input type="button" value="Click" onclick="showResult();"/>


</body>
</html>
```

inspect() Method

This method returns the debug-oriented string representation of element. Prototype provides inspect methods for many types, both built-in and library-defined, such as in String, Array, Enumerable and Hash, which attempts to provide most-useful string representations for their respective types.

- *undefined* and *null* are represented as such.
- Other types are looked up for a *inspect* method: if there is one, it is used, otherwise, it reverts to the *toString* method.

Syntax

```
element.inspect();
```

Return Value

- Returns HTML element after inserted content.

Example

```
<html>

<head>

<title>Prototype examples</title>


<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   var str = $('golden-delicious').inspect();

   alert("$('golden-delicious').inspect(): " + str );


   var str = $('mutsu').inspect();

   alert("$('mutsu').inspect() : " + str );


   var str = $('mutsu').next().inspect();

   alert("$('mutsu').next().inspect() : " + str );

}

</script>

</head>


<body>
```

```
    <p>Click the button to see the result.</p>

    <ul>
       <li id="golden-delicious">Golden Delicious</li>
       <li id="mutsu" class="yummy apple">Mutsu</li>
       <li id="mcintosh" class="yummy">McIntosh</li>
       <li></li>
    </ul>

    <br />
    <input type="button" value="Click" onclick="showResult();"/>

  </body>
  </html>
```

makeClipping() Method

This method simulates the poorly supported CSS clip property by setting element's overflow value to 'hidden'.

Syntax

```
element.makeClipping();
```

Return Value

- Returns an HTML element.

Example

```
<html>
```

```
<head>
<title>Prototype examples</title>

<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function Clipping(){
   $('framer').makeClipping().setStyle(
               {width: '50px', height: '50px'})
}

function Original(){
   $('framer').undoClipping();
}
</script>
</head>

<body>

  <p>Click the button to see the result.</p>

  <div id="framer">
  <img src="/images/prototype.gif" alt="Prototype Logo" />
  </div>
```

```
    <br />

    <input type="button" value="Clipping" onclick="Clipping();"/>

    <input type="button" value="Original" onclick="Original();"/>


  </body>

  </html>
```

makePositioned() Method

This method allows for the easy creation of CSS, containing block by setting element's CSS position to 'relative' if its initial position is either 'static' or undefined.

Syntax

```
element.makePositioned();
```

Return Value

- Returns HTML element.

Example

```
<html>

<head>

<title>Prototype examples</title>


<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){
   $('container').makePositioned();
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />

  <div id="container">
  <div id="element"
          style="position:absolute; top: 20px; left: 20px;
          border:1px solid red;">
   This is Element Box
  </div>
  </div>

  <br />
  <input type="button" value="Click" onclick="showResult();"/>

</body>
</html>
```

match() Method

This method checks if element matches the given CSS selector.

Syntax

```
element.match(selector);
```

Return Value

- Returns Boolean value. If it finds a match, then it returns a *true*, otherwise *false*.

Example

```
<html>
<head>
<title>Prototype examples</title>

<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){
   if( $('fruits').match('ul') ){
   alert( " $('fruits').match('ul') returns true " );
   }else{
   alert( " $('fruits').match('ul') returns false " );
   }

   if( $('mcintosh').match('li#mcintosh.yummy') ){
   alert("$('mcintosh').match('li#mcintosh.yummy') returns true");
   }else{
   alert("$('mcintosh').match('li#mcintosh.yummy') returns false");
```

```
      }


      if( $('fruits').match('p') ){

      alert( " $('fruits').match('p') returns true " );

      }else{

      alert( " $('fruits').match('p') returns false " );

      }

   }


   </script>

   </head>


   <body>


     <p>Click the button to see the result.</p>


     <ul id="fruits">

       <li id="apples">

       <ul>

         <li id="golden-delicious">Golden Delicious</li>

         <li id="mutsu" class="yummy">Mutsu</li>

         <li id="mcintosh" class="yummy">McIntosh</li>

         <li id="ida-red">Ida Red</li>

       </ul>

       </li>

     </ul>
```

```
    <br />

    <input type="button" value="Show Result" onclick="showResult();"/>


  </body>

  </html>
```

next() Method

This method checks if element matches the given CSS selector.

Syntax

```
  element.next( [cssRule][, index = 0]);
```

Return Value

- If there is one found element, then it returns HTML element, otherwise it returns *undefined*.

Example

```
  <html>

  <head>

  <title>Prototype examples</title>


  <script type="text/javascript"

     src="/javascript/prototype.js">

  </script>

  <script>


  function showResult(){

     var str = $('list-of-apples').next();
```

```
       alert( "$('list-of-apples').next() " + str.innerHTML );


       var str = $('list-of-apples').next(0);

       alert( "$('list-of-apples').next(0) " + str.innerHTML  );


       var str = $('title').next('p');

       alert( "$('$('title').next('p') " + str.innerHTML  );


       var str = $('golden-delicious').next('.yummy', 1);

       alert( "$('golden-delicious').next('.yummy', 1) " +

                                         str.innerHTML);


       var str = $('ida-red').next();

       alert( "$('ida-red').next() " + str );

    }


    </script>

    </head>


    <body>


      <p>Click the button to see the result.</p>


      <ul id="fruits">

      <li id="apples">

        <h3 id="title">Apples</h3>

        <ul id="list-of-apples">
```

```
          <li id="golden-delicious">Golden Delicious</li>

          <li id="mutsu">Mutsu</li>

          <li id="mcintosh" class="yummy">McIntosh</li>

          <li id="ida-red" class="yummy">Ida Red</li>

        </ul>

        <p id="saying">An apple a day keeps the doctor away.</p>

      </li>

      </ul>


      <br />

      <input type="button" value="Show Result" onclick="showResult();"/>


    </body>

    </html>
```

nextSiblings() Method

This method collects all of element's next siblings and returns them as an array of extended elements.

NOTE: Two elements are siblings if they have the same parent.

Syntax

```
element.nextSiblings();
```

Return Value

- An array of HTML elements.

Example

```
<html>

<head>
```

```
<title>Prototype examples</title>


<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   var str = $('mutsu').nextSiblings();

   alert( "$('mutsu').nextSiblings() " + str[0].innerHTML );

   alert( "$('mutsu').nextSiblings() " + str[1].innerHTML );


   var str = $('ida-red').nextSiblings();

   alert( "$('ida-red').nextSiblings() " + str[0] );


 }


</script>

</head>


<body>


  <p>Click the button to see the result.</p>


  <ul>

     <li id="golden-delicious">Golden Delicious</li>

     <li id="mutsu">Mutsu</li>
```

```
         <li id="mcintosh">McIntosh</li>

         <li id="ida-red">Ida Red</li>

      </ul>


      <br />

      <input type="button" value="Click" onclick="showResult();"/>


    </body>

    </html>
```

observe () Method

This method registers an event handler on element and returns element.

Syntax

```
    element.observe(eventName, handler[, useCapture = false]);
```

Return Value

- An HTML element.

Example

```
    <html>

    <head>

    <title>Prototype examples</title>


    <script type="text/javascript"

       src="/javascript/prototype.js">

    </script>

    <script>
```

```
function RegisterFunction()

{


   $('test').observe('click', function(event){

      alert(Event.element(event).innerHTML);

   });


}


</script>

</head>


<body onload="RegisterFunction();">


  <p id="test">Click me to see the result.</p>


</body>

</html>
```

positionedOffset() Method

This method returns element.s offset relative to its closest positioned ancestor. This calculates the cumulative *offsetLeft* and *offsetTop* of an element and all its parents until it reaches an element with a position of static.

NOTE: Note that all values are returned as numbers, only although they are expressed in pixels.

Syntax

```
element.positionedOffset();
```

Return Value

- Returns element.s offset relative to its closest positioned ancestor.

Example

```
<html>

<head>

<title>Prototype examples</title>


<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   var arr = $('container2').positionedOffset();

   alert("offsetLeft : " + arr[0] );

   alert("offsetTop : " + arr[1] );

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />
```

```
      <div id="container1">
      <div id="element"
              style="position:absolute; top: 30px; left: 20px;
              border:1px solid red;">
       This is the first Element Box
      </div>
      </div>

      <div id="container2">
      <div id="element"
              style="top: 30px; left: 20px;
              border:1px solid red;">
       This is the second Element Box
      </div>
      </div>

      <br />

      <input type="button" value="Click" onclick="showResult();"/>

    </body>
    </html>
```

previous() Method

This method returns the element's previous sibling or the index'th one, if index is specified that matches *cssRule*. If no cssRule is provided, all previous siblings are considered. If no previous sibling matches these criteria, undefined is returned.

Syntax

```
element.previous([cssRule][, index = 0] );
```

Return Value

- Returns HTML element. If no previous sibling matches these criteria, undefined is returned.

Example

```
<html>

<head>

<title>Prototype examples</title>


<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   var str = $('saying').previous();

   alert( "$('saying').previous() " + str.innerHTML );


   var str = $('saying').previous(0);

   alert( "$('saying').previous(0) " + str.innerHTML  );


   var str = $('saying').previous('h3');

   alert( "$('$('saying').previous('h3') " + str.innerHTML  );
   var str = $('ida-red').previous( '.yummy');
   alert( "$('ida-red').previous( .yummy ) " + str.innerHTML );

   var str = $('ida-red').previous( 5 );
   alert( "$('ida-red').previous() " + str );
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>

  <ul id="fruits">

  <li id="apples">
    <h3 id="title">Apples</h3>
    <ul id="list-of-apples">
      <li id="golden-delicious">Golden Delicious</li>
      <li id="mutsu">Mutsu</li>
      <li id="mcintosh" class="yummy">McIntosh</li>
      <li id="ida-red" class="yummy">Ida Red</li>
    </ul>
    <p id="saying">An apple a day keeps the doctor away.</p>
  </li>
  </ul>


  <br />

  <input type="button" value="Show Result" onclick="showResult();"/>


</body>

</html>
```

previousSiblings() Method

This method collects all of element's previous siblings and returns them as an array of extended elements.

Two elements are siblings if they have the same parent. So for example, the head and body elements are siblings (their parent is the html element). Previous siblings are simply the ones which precede element in the document.

Syntax

```
element.previousSiblings();
```

Return Value

- Returns an array of HTML elements.

Example

```
<html>

<head>

<title>Prototype examples</title>


<script type="text/javascript"

   src="/javascript/prototype.js">

</script>
```

```
<script>


function showResult(){

   var arr = $('ida-red').previousSiblings();

   arr.each(function(node){

      alert(node.nodeName + ': ' + node.innerHTML);

   });


}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>


  <ul id="fruits">


  <li id="apples">

    <h3 id="title">Apples</h3>

    <ul id="list-of-apples">

      <li id="golden-delicious">Golden Delicious</li>

      <li id="mutsu">Mutsu</li>

      <li id="mcintosh" class="yummy">McIntosh</li>

      <li id="ida-red" class="yummy">Ida Red</li>

    </ul>
```

```
      <p id="saying">An apple a day keeps the doctor away.</p>

    </li>


    </ul>


    <br />

    <input type="button" value="Show Result" onclick="showResult();"/>


</body>

</html>
```

readAttribute() Method

This method returns the value of element's attribute or null, if attribute has not been specified.

Syntax

```
element.readAttribute(attribute);
```

Return Value

- Returns the value of element's attribute or null, if attribute has not been specified.

Example

```
<html>

<head>

<title>Prototype examples</title>


<script type="text/javascript"

   src="/javascript/prototype.js">
```

```
</script>
<script>

function showResult(){
   var str = $('tag').readAttribute('href');
   alert("Value of href : " + str );

   var str = $('tag').readAttribute('title');
   alert("Value of title : " + str );

   var str = $('tag').readAttribute('somethingelse');
   alert("Value ofsomething else : " + str );

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>

  <a id="tag" href="/tags/prototype" rel="tag"
         title="view related bookmarks.">Prototype</a>

  <br />
  <br />
```

```
      <input type="button" value="Show Result" onclick="showResult();"/>


    </body>

    </html>
```

recursivelyCollect() Method

This method recursively collects elements whose relationship is specified by property. Property has to be a property of element that points to a single DOM node.

Syntax

```
    element.recursivelyCollect(property);
```

Value of property could be any of the following :

- parentNode
- previousSibling
- nextSibling

Return Value

- Returns an array of HTML elements.

Example

```
    <html>

    <head>

    <title>Prototype examples</title>


    <script type="text/javascript"

       src="/javascript/prototype.js">

    </script>

    <script>
```

```
function showResult(){
   var arr = $('fruits').recursivelyCollect('nextSibling');
   arr.each(function(node){
      alert(node.nodeName + ': ' + node.innerHTML);
   });
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>

  <ul id="fruits">
  <li id="apples">
    <ul id="list-of-apples">
      <li id="golden"><p>Golden</p></li>
      <li id="mutsu">Mutsu</li>
      <li id="mcintosh">McIntosh</li>
      <li id="ida-red">Ida Red</li>
    </ul>
  </li>
  </ul>

  <p>This is the paragraph</p>
  <br />
```

```
      <input type="button" value="Show Result" onclick="showResult();"/>


    </body>
    </html>
```

relativize() Method

This method turns element into a relatively-positioned element without changing its position in the page layout.

Syntax

```
    element.relativize();
```

Return Value

- Returns an HTML element.

Example

```
    <html>
    <head>
    <title>Prototype examples</title>
    <script type="text/javascript"
       src="/javascript/prototype.js">
    </script>
    <script>


    function changetoRel(){
       node = $("firstDiv");
       node.style.fontSize='20px';
       //node.style.position = 'relative';
```

```
      node.relativize();
      node.style.border = '1px dashed #f00';
      node.style.left = '100px';
   }

   </script>
   </head>

   <body>
      <div id="firstDiv" style="position:absolute;top:100px;">
         <p>This is first paragraph</p>
      </div>

      <br />
      <input type="button" value="Make Relative"
                onclick="changetoRel();"/>

   </body>
   </html>
```

remove() Method

This method completely removes element from the document and returns it.

Syntax

```
   element.remove();
```

Return Value

- Returns removed HTML element.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   var str = $('mutsu').remove();

   alert("Value of str : " + str.innerHTML );


   var str = $('ida-red').remove();

   alert("Value of str : " + str.innerHTML );


}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>


  <ul>

  <li id="golden-delicious">Golden Delicious</li>
```

```
    <li id="mutsu">Mutsu</li>

    <li id="mcintosh">McIntosh</li>

    <li id="ida-red">Ida Red</li>

    </ul>


    <input type="button" value="Show Result" onclick="showResult();"/>


  </body>

  </html>
```

removeClassName() Method

This method removes the element's CSS className and returns element.

Syntax

```
  element.remove();
```

Return Value

- Returns HTML element from which it removes CSS class.

Example

```
  <html>

  <head>

  <title>Prototype examples</title>

  <script type="text/javascript"

     src="/javascript/prototype.js">

  </script>

  <script>
```

```
function showResult(){

   // Before removing class name
   var arr = $('dingo').classNames();
   arr.each(function(class){
      alert( "Before Removing, Class name is " + class );
   });

   $('dingo').removeClassName('food');

   // After removing class name
   var arr = $('dingo').classNames();
   arr.each(function(class){
      alert( "After removing, Class name is " + class );
   });

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>

  <div id="dingo" class="apple fruit food">
  <ul>
```

```
      <li id="golden-delicious">Golden Delicious</li>

      <li id="mutsu">Mutsu</li>

      <li id="mcintosh">McIntosh</li>

      <li id="ida-red">Ida Red</li>

      </ul>

      </div>


      <input type="button" value="Show Result" onclick="showResult();"/>


    </body>

    </html>
```

replace() Method

This method replaces element by the content of the html argument and returns the removed element.

Syntax

```
    element.replace(html);
```

Here *html* can be either plain text, an HTML snippet or any JavaScript object, which has a toString() method.

If it contains any <script> tags, these will be evaluated after the element has been replaced.

NOTE: If no argument is provided, Element.replace will simply clear element of its content.

Return Value

- Returns the removed HTML element.

Example

```
    <html>
```

```
<head>
<title>Prototype examples</title>

<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult()
{

   $('first').replace('<ul id="favorite">' +
                           '<li>kiwi</li>' +
                          '<li>banana</li>' +
                          '<li>apple</li>' +
                '</ul>');

}

</script>
</head>

<body">

  <p id="test">Click the button to see the result.</p>

  <div id="food">
```

```
      <div id="fruits">
        <p id="first">Kiwi, banana <em>and</em> apple.</p>
      </div>
      </div>

      <input type="button" value="Click" onclick="showResult();"/>

    </body>
    </html>
```

scrollTo() Method

This method scrolls the window so that element appears at the top of the viewport.

This has a similar effect than what would be achieved using HTML anchors (except the browser's history is not modified).

Syntax

```
element.scrollTo();
```

Return Value

- Returns an HTML element.

Example

```
<html>
<head>
<title>Prototype examples</title>

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

```
<script>


function showResult()

{


   $('heading').scrollTo();


}


</script>

</head>


<body">

  <h1 id="heading">This is the heading.</h1>

  <p id="test">Click the button see the result.</p>

  <p id="test">Click the button see the result.</p>

  <p id="test">Click the button see the result.</p>

  <p id="test">Click the button see the result.</p>

  <p id="test">Click the button see the result.</p>

  <p id="test">Click the button see the result.</p>

  <p id="test">Click the button see the result.</p>

  <p id="test">Click the button see the result.</p>

  <p id="test">Click the button see the result.</p>

  <p id="test">Click the button see the result.</p>

  <p id="test">Click the button see the result.</p>

  <p id="test">Click the button see the result.</p>

  <p id="test">Click the button see the result.</p>
```

```
    <p id="test">Click the button see the result.</p>

    <p id="test">Click the button see the result.</p>

    <p id="test">Click the button see the result.</p>

    <p id="test">Click the button see the result.</p>

    <p id="test">Click the button see the result.</p>

    <p id="test">Click the button see the result.</p>

    <p id="test">Click the button see the result.</p>

    <p id="test">Click the button see the result.</p>

    <p id="test">Click the button see the result.</p>

    <p id="test">Click the button see the result.</p>

    <p id="test">Click the button see the result.</p>


    <input type="button" value="Click" onclick="showResult();"/>


  </body>

  </html>
```

select() Method

This method takes an arbitrary number of CSS selectors (strings) and returns an array of extended descendants of element that matches any of them.

This method is very similar to $$() but can be used within the context of one element, rather than the whole document. The supported CSS syntax is identical, so please refer to the $$() docs for details.

Syntax

```
element.select(selector...);
```

Return Value

- Returns an array of HTML elements.

Example

```
<html>
<head>
<title>Prototype examples</title>

<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult()
{

   var arr = $('apples').select('[title="yummy!"]');
   // returns [h3, li#golden-delicious, li#mutsu]
   arr.each(function(node){
      alert("First : " + node.nodeName + ': ' + node.innerHTML);
   });

   arr = $('apples').select( 'p#saying', 'li[title="yummy!"]');
   // returns [li#golden-delicious, li#mutsu,  p#saying]
   arr.each(function(node){
      alert("Second : " + node.nodeName + ': ' + node.innerHTML);
   });

   arr = $('apples').select('[title="disgusting!"]');
   // returns []
```

```
      arr.each(function(node){
         alert("Third : " + node.nodeName + ': ' + node.innerHTML);
      });
   }

   </script>
   </head>

   <body">

     <p id="test">Click the button to see the result.</p>

     <ul id="fruits">
     <li id="apples">
       <h3 title="yummy!">Apples</h3>
       <ul id="list-of-apples">
         <li id="golden" title="yummy!" >Golden</li>
         <li id="mutsu" title="yummy!">Mutsu</li>
         <li id="mcintosh">McIntosh</li>
         <li id="ida-red">Ida Red</li>
       </ul>
       <p id="saying">An apple a day keeps the doctor away.</p>
     </li>
     </ul>

     <input type="button" value="Click" onclick="showResult();"/>
```

```
</body>

</html>
```

setOpacity() Method

This method sets the visual opacity of an element while working around inconsistencies in various browsers.

The opacity argument should be a floating point number, where the value of 0 is fully transparent and 1 is fully opaque.

Element.setStyle method uses *setOpacity* internally to set opacity.

Syntax

```
element.setOpacity(opacity);
```

Return Value

- Returns an HTML element.

Example

```
<html>

<head>

<title>Prototype examples</title>



<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>



function reduceOpacity()

{
   $('test').setOpacity( 0.5 );

   // This is equivalent to

   // Element.setStyle({ opacity: 0.5 });


}


</script>

</head>


<body">


  <p id="test">Click the button to see the result.</p>


  <input type="button" value="Click" onclick="reduceOpacity();"/>


</body>

</html>
```

setStyle() Method

This method modifies element's CSS style properties. Styles are passed as a hash of property-value pairs in which the properties are specified in their camelized form.

Syntax

```
element.setStyle(styles);
```

Return Value

- Returns an HTML element.

Example

```
<html>
<head>
<title>Prototype examples</title>

<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function setColor()
{

   $('test').setStyle({
     backgroundColor: '#900',
     fontSize: '12px'
   });

}

</script>
</head>

<body">

  <p id="test">Click the button to see the result.</p>


  <input type="button" value="Click" onclick="setColor();"/>
```

```
</body>

</html>
```

show() Method

This method displays and returns element.

This method cannot display elements hidden via CSS stylesheets. Note that this is not a Prototype limitation but a consequence of how the CSS display property works.

Syntax

```
element.show();
```

Return Value

- Displays and returns HTML element

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function hideElement(){


   $('grandfather').hide();

}

function showElement(){
   $('grandfather').show();
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>

   <div id="grandfather" >This is grand father
      <div id="father"> This is father.
         <div id="kid" >This is kid</div>
      </div>
   </div>

   <br />
   <input type="button" value="Hide" onclick="hideElement();"/>
   <input type="button" value="Show" onclick="showElement();"/>

</body>
</html>
```

siblings() Method

This method collects all of element's siblings and returns them as an array of extended elements.

Two elements are siblings if they have the same parent. So for example, the head and body elements are siblings (their parent is the html element).

Syntax

```
element.siblings();
```

Return Value

- Returns an array of HTML elements.

Example

```
<html>

<head>

<title>Prototype examples</title>


<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult()

{


   var arr = $('mutsu').siblings();

   arr.each(function(node){

      alert(node.nodeName + ': ' + node.innerHTML);

   });


}
```

```
</script>
</head>

<body">

  <p id="test">Click the button to see the result.</p>

  <ul>
     <li id="golden-delicious">Golden Delicious</li>
     <li id="mutsu">Mutsu</li>
     <li id="mcintosh">McIntosh</li>
     <li id="ida-red">Ida Red</li>
  </ul>

  <input type="button" value="Click" onclick="showResult();"/>

</body>
</html>
```

stopObserving() Method

This method unregisters handler and returns element.

Syntax

```
element.stopObserving(eventName, handler);
```

Return Value

- Returns an HTML element.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function handler(event){
   alert(Event.element(event).innerHTML);
}

function RegisterFunction()
{

   $('test').observe('click', handler );
   alert("Registering the handler");

}

function UnRegister()
{

   $('test').stopObserving('click', handler);
   alert("Now unregistering the handler");
}
```

```
</script>
</head>

<body onload="RegisterFunction();">

  <p id="test">Click me to see the result.</p>

  <br />
  <p>Click the button to unregister the handler.</p>
  <input type="button" value="UnReg" onclick="UnRegister();"/>

</body>
</html>
```

toggle() Method

This method toggles the visibility of an element.

NOTE: Element.toggle cannot display elements hidden via CSS stylesheets. Note that this is not a Prototype limitation but a consequence of how the CSS display property works.

Syntax

```
element.toggle();
```

Return Value

- Returns an HTML element.

Example

```
<html>
<head>
```

```
<title>Prototype examples</title>


<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>


function toggleMsg(){
   $('test').toggle();
}
</script>
</head>


<body>


  <p>Click the button to see the result.</p>


  <div id="test" style="display:inline;">
  <p>This message will toggle when you click the button</p>
  </div>


  <br />
  <input type="button" value="Click" onclick="toggleMsg();"/>


</body>
</html>
```

toggleClassName() Method

This method toggles element's CSS className and returns element.

Syntax

```
element.toggleClassName( className );
```

Return Value

- Returns an HTML element.

Example

```
<html>
<head>
<title>Prototype examples</title>

<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){
   if( !$('mutsu').hasClassName('fruit') ){
      // This will return false
      alert( "Class name is not set" );
   }

   $('mutsu').toggleClassName('fruit');
   // This will toggle the class presence
```

```
       if( $('mutsu').hasClassName('fruit') ){

          // This returns true

          alert( "Now Class name is set" );

       }

    }

    </script>

    </head>


    <body>


      <p>Click the button to see the result.</p>


      <div id="mutsu" class="apple">

         <p>This is test division.</p>

      </div>


      <br />

      <input type="button" value="showResult"

                                onclick="showResult();"/>


    </body>

    </html>
```

undoClipping() Method

This method sets element's CSS overflow property back to the value it had before *Element.makeClipping()* was applied.

Syntax

```
element.undoClipping();
```

Return Value

- Returns HTML element.

Example

```
<html>

<head>

<title>Prototype examples</title>


<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function Clipping(){

   $('framer').makeClipping().setStyle(

               {width: '50px', height: '50px'})

}


function Original(){

   $('framer').undoClipping();

}

</script>

</head>
```

```
<body>


  <p>Click the button to see the result.</p>


  <div id="framer">


  <img src="/images/prototype.gif" alt="Prototype Logo" />

  </div>


  <br />

  <input type="button" value="Clipping" onclick="Clipping();"/>

  <input type="button" value="Original" onclick="Original();"/>


</body>

</html>
```

undoPositioned() Method

This method sets element back to the state it was before *Element.makePositioned* was applied to it.

Syntax

```
element.undoPositioned();
```

Return Value

- Returns HTML element.

Example

```
<html>
```

```
<head>
<title>Prototype examples</title>

<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function Positioned(){
   $('container').makePositioned();
}
function UnPositioned(){
   $('container').undoPositioned();
}
</script>
</head>

<body>

  <p>Click the buttons to see the result.</p>
  <br />

  <div id="container">
  <div id="element"
          style="position:absolute; top: 60px; left: 20px;
          border:1px solid red;">
   This is Element Box
```

```
      </div>

      </div>


      <br />


      <input type="button" value="Postioned"

                onclick="Positioned();"/>


      <input type="button" value="UnPositioned"

                onclick="UnPositioned();"/>


    </body>

    </html>
```

up() Method

Returns element's first ancestor (or the index'th ancestor, if index is specified) that matches *cssRule*.

If no *cssRule* is provided, all ancestors are considered. If no ancestor matches these criteria, undefined is returned.

Syntax

```
    element.up([cssRule][, index = 0])
```

Return Value

- Found HTMLElement.
- undefined in case it does not find any element.

NOTE: element.up() and element.up(0) are equivalent.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   alert( "1 : " + $('fruits').up(1).innerHTML );

   alert( "2 : " + $('mcintosh').up('li').innerHTML );

   alert( "5 : " + $('fruits').up(99) );

}




</script>

</head>


<body>


   <p>Click the button to see the result.</p>


   <ul id="fruits">


   <li id="apples">

    <ul>

      <li id="golden-delicious">Golden Delicious</li>
```

```
          <li id="mutsu" class="yummy">Mutsu</li>

          <li id="mcintosh" class="yummy">McIntosh</li>

          <li id="ida-red">Ida Red</li>

        </ul>

       </li>

       </ul>


       <br />


       <input type="button" value="showResult"

                 onclick="showResult();"/>


    </body>

    </html>
```

update() Method

This method replaces the content of the element with the provided *newContent* argument and returns the element.

Given *newContent* can be a plain text, an HTML snippet, or any JavaScript object, which has a toString() method. If it contains any <script> tags, these will be evaluated after element has been updated.

If no argument is provided, Element.update will simply clear element of its content.

Syntax

```
element.update( newContent );
```

Return Value

- Updated HTML element.

Example

```
<html>
<head>
<title>Prototype examples</title>

<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult()
{
   $('movies').update("Spider Man, USV-315");
}

</script>
</head>

<body>

  <p>Click the button to see the result</p>
  <div id="movies">
      Speed, Titanic, Brave Heart
  </div>

  <input type="button" value="Click" onclick="showResult();"/>

</body>
```

```
</html>
```

viewportOffset() Method

This method returns the X/Y coordinates of element relative to the viewport. Returned positions will be absolute.

NOTE: All values are returned as numbers, only although they are expressed in pixels.

Syntax

```
element.viewportOffset();
```

Return Value

- An array of numbers [Left, Top].

Example

```
<html>

<head>

<title>Prototype examples</title>



<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>



function showResult()

{



   var coordinates = $('visible').viewportOffset();

   alert("Distance from the Left : " +  coordinates[0] );

   alert("Distance from the Top  : " +  coordinates[1] );
}


</script>

</head>


<body>


  <div id="visible" style="position:absolute;left:50px; top:50px;">

     This is visible division 50 pixels down from the top.

  </div>


  <input type="button" value="Click" onclick="showResult();"/>


</body>

</html>
```

visible() Method

This method returns a Boolean indicating whether or not the element is visible, i.e., whether its inline style property is set to "display: none;".

NOTE: Styles applied via a CSS stylesheet are not taken into consideration. Note that this is not a Prototype limitation, it is a CSS limitation.

Syntax

```
element.visible();
```

Return Value

- An HTML element.

Example

```
<html>
<head>
<title>Prototype examples</title>

<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult()
{
   if( $('visible').visible() ){
      alert("$('visible').visible() returns true" );
   }else{
      alert("$('visible').visible() returns false" );
   }

   if( $('hidden').visible() ){
      alert("$('hidden').visible() returns true" );
   }else{
      alert("$('hidden').visible() returns false" );
   }
}

</script>
</head>
```

```
<body>


  <p>Click button to see the result</p>


  <div id="visible" style="display:block;">

     This is visible division

  </div>


  <div id="hidden" style="display: none;">

     This is hidden division

  </div>




  <input type="button" value="Click" onclick="showResult();"/>


</body>

</html>
```

wrap() Method

This method wraps an element inside another, then returns the wrapper.

Syntax

```
element.wrap(element, wrapper[, attributes]);


OR


someElement.wrap(wrapper[, attributes]);
```

Return Value

- An HTML element.

Example

```
<html>

<head>

<title>Prototype examples</title>


<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult()

{

   $('test').wrap( $("bold") );

}


</script>

</head>


<body>


  <p id="test">Formatted Paragraph</p>


  <b id="bold">Above Paragraph should be wrapped here....<br />

  </b>
```

```
    <br />

    <input type="button" value="Click" onclick="showResult();"/>

</body>
</html>
```

writeAttribute() Method

This method adds, specifies, or removes attributes passed as either a hash or a name/value pair.

Syntax

```
element.writeAttribute(attribute[, value]);

OR

element.writeAttribute(attributes);
```

Return Value

- An HTML element.

Example

```
<html>
<head>
<title>Prototype examples</title>

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

```
      src="/javascript/prototype.js">
   </script>
   <script>

   function showResult()
   {
      $('test').writeAttribute("align", "right" );
   }

   </script>
   </head>

   <body>

     <p id="test" align="center" >Formatted Paragraph</p>

     <br />

     <input type="button" value="Click" onclick="showResult();"/>

   </body>
   </html>
```

5. Prototype – Number Processing

Prototype extends native JavaScript numbers in order to provide:

- ObjectRange compatibility, through Number#succ.
- Ruby-like numerical loops with Number#times.
- Simple utility methods such as Number#toColorPart and Number#toPaddedString.

Here is the list of all the functions with examples dealing with Numbers.

Prototype Number Method

NOTE: Make sure you have the prototype.js version of 1.6.

Methods	Description
abs()	Returns the absolute value of the number.
ceil()	Returns the smallest integer greater than or equal to the number.
floor()	Returns the largest integer less than or equal to the number.
round()	Rounds the number to the nearest integer.
succ()	Returns the successor of the current Number, as defined by current + 1. Used to make numbers compatible with ObjectRange.
times()	Encapsulates a regular [0..n] loop, Ruby-style.
toColorPart()	Produces a 2-digit hexadecimal representation of the number (which is therefore assumed to be in the [0..255] range). Useful for composing CSS color strings.
toJSON()	Returns a JSON string.
toPaddedString()	Converts the number into a string padded with 0s so that the string's length is at least equal to length.

abs() Method

This method returns the absolute value of the number.

Syntax

```
number.abs();
```

Return Value

- Returns the absolute value of the number.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


   alert( "Math.abs(-5) : " + Math.abs(-5) );

   alert( "(-5).abs() : " + (-5).abs() );

   alert( "(5).abs() : " + (5).abs() );

}


</script>

</head>
```

```
<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

ceil() Method

This method returns the smallest integer greater than or equal to the number.

Syntax

```
number.ceil();
```

Return Value

- Returns the smallest integer greater than or equal to the number.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>
```

```
function showResult(){


   alert( "Math.ceil(4.1) : " + Math.ceil(4.1) );

   alert( "(4.1).ceil()  : " + (4.1).ceil()  );

   alert( "(-4.1).ceil()  : " + (-4.1).ceil()  );

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

floor() Method

This method returns the largest integer less than or equal to the number.

Syntax

```
number.floor();
```

Return Value

- Returns the largest integer less than or equal to the number.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


   alert( "Math.floor(4.6) : " + Math.floor(4.6) );

   alert( "(4.6).floor()  : " + (4.6).floor() );

   alert( "(-4.1).floor()  : " + (-4.1).floor()  );

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>
```

```
</html>
```

round() Method

This method rounds the number to the nearest integer.

Syntax

```
number.round();
```

Return Value

- Rounds the number to the nearest integer.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


   alert( "Math.round(4.5) : " + Math.round(4.5) );

   alert( "(4.5).round()  : " + (4.5).round() );

   alert( "(4.49).round()  : " + (4.49).round()  );

   alert( "(-4.5).round()  : " + (-4.5).round()  );

}

```

```
</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

succ() Method

This method returns the successor of the current Number, as defined by current + 1. Used to make numbers compatible with ObjectRange.

Syntax

```
number.succ();
```

Return Value

- Rounds the successor of the current Number.

Example

```
<html>

<head>

<title>Prototype examples</title>

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

```
      src="/javascript/prototype.js">
   </script>
   <script>

   function showResult(){

      alert( "(5).succ() : " + (5).succ() );
      alert( "(-5).succ()  : " + (-5).succ() );
   }

   </script>
   </head>

   <body>

     <p>Click the button to see the result.</p>
     <br />
     <br />
     <input type="button" value="Result" onclick="showResult();"/>

   </body>
   </html>
```

times() Method

This method calls the callback function N number of times starting with 0. The callback function is invoked with a single argument, ranging from 0 to the number, exclusive.

Syntax

```
N.times(Callback Function);
```

Return Value

- Encapsulated Number.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


   var s = '';

   (5).times(function(n) {

      s += n;

   });

   alert("Encapsulated Number : " + s );

}


</script>

</head>
```

```
<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

toColorPart() Method

This method produces a 2-digit hexadecimal representation of the number, which is therefore assumed to be in the [0..255] range.

This function is useful for composing CSS color strings.

Syntax

```
number.toColorPart();
```

Return Value

- Encapsulated Number.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>
```

```

function showResult(){


  alert("(128).toColorPart() : " + (128).toColorPart() );

  alert("(10).toColorPart() : " + (10).toColorPart() );


}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

toJSON() Method

This method returns a JSON string.

Syntax

```
number.toJSON();
```

Return Value

- JSON string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


  alert("(45).toJSON() : " + (45).toJSON() );


}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>
```

```


    </body>

    </html>
```

toPaddedString() Method

This method converts the number into a string padded with 0s so that the string's length is at least equal to length. Takes an optional radix argument, which specifies the base to use for conversion.

Syntax

```
    number.toPaddedString(length[, radix]);
```

Return Value

- Returns a string.

Example

```
    <html>

    <head>

    <title>Prototype examples</title>

    <script type="text/javascript"

       src="/javascript/prototype.js">

    </script>

    <script>


    function showResult(){


      alert("(13).toPaddedString(4) : " + (13).toPaddedString(4) );

      alert("(13).toPaddedString(2) : " + (13).toPaddedString(2) );

      alert("(13).toPaddedString(1) : " + (13).toPaddedString(1) );
    alert("(13).toPaddedString(4, 2): " + (13).toPaddedString(4, 2));


  }


  </script>

  </head>


  <body>


    <p>Click the button to see the result.</p>

    <br />

    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

6. Prototype – String Processing

Prototype enhances the String object with a series of useful methods ranging from the trivial to the complex.

Here is the list of all the functions with examples dealing with String.

Prototype String Methods

NOTE: Make sure you have the prototype.js version of 1.6.

Methods	Description
blank()	Checks if the string is 'blank', meaning either empty or containing only whitespace.
camelize()	Converts a string separated by dashes into a camelCase equivalent. For instance, 'foo-bar' would be converted to 'fooBar'.
capitalize()	Capitalizes the first letter of a string and downcases all the others.
dasherize()	Replaces every instance of the underscore character ("_") by a dash ("-").
empty()	Checks if the string is empty.
endsWith()	Checks if the string ends with substring.
escapeHTML()	Converts HTML special characters to their entity equivalents.
evalJSON()	Evaluates the JSON in the string and returns the resulting object.
evalScripts()	Evaluates the content of any script block present in the string. Returns an array containing the value returned by each script.
extractScripts()	Extracts the content of any script block present in the string and returns them as an array of strings.

gsub()	Returns the string with every occurrence of a given pattern replaced by either a regular string, the returned value of a function or a Template string.
include()	Checks if the string contains a substring.
inspect()	Returns a debug-oriented version of the string.
interpolate()	Treats the string as a Template and fills it with object's properties.
isJSON()	Checks if the string is valid JSON by the use of regular expressions. This security method is called internally.
parseQuery()	Parses a URI-like query string and returns an object composed of parameter/value pairs.
scan()	Allows iterating over every occurrence of the given pattern.
startsWith()	Checks if the string starts with substring.
strip()	Strips all the leading and trailing whitespace from a string.
stripScripts()	Strips a string of anything that looks like an HTML script block.
stripTags()	Strips a string of any HTML tag.
sub()	Returns a string with the first count occurrences of pattern replaced by either a regular string, the returned value of a function or a Template string.
succ()	Used internally by ObjectRange. Converts the last character of the string to the following character in the Unicode alphabet.
times()	Concatenates the string count times.
toArray()	Splits the string character-by-character and returns an array with the result.

toJSON()	Returns a JSON string.
toQueryParams()	Parses a URI-like query string and returns an object composed of parameter/value pairs.
truncate()	Truncates a string to the given length and appends a suffix to it (indicating that it is only an excerpt).
underscore()	Converts a camelized string into a series of words separated by an underscore ("_").
unescapeHTML()	Strips tags and converts the entity forms of special HTML characters to their normal form.
unfilterJSON ()	Strips comment delimiters around Ajax JSON or JavaScript responses. This security method is called internally.

blank() Method

This method checks if the string is 'blank', meaning either empty or containing only whitespace.

Syntax

```
string.blank();
```

Return Value

- Returns a boolean value. Either true if string is empty or false if string is not empty.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">
```

tutorialspoint
SIMPLYEASYLEARNING

```
</script>
<script>

function showResult(){

  var str = '';

  alert("''.blank() : " + str.blank() );

  str = '  ';
  alert("'  '.blank() : " + str.blank() );

  str = 'a';
  alert("a.blank() : " + str.blank() );
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
```

```
</html>
```

camelize() Method

This method converts a string separated by dashes into a camelCase equivalent. For instance, 'foo-bar' would be converted to 'fooBar'.

Prototype uses this internally for translating CSS properties into their DOM style property equivalents.

Syntax

```
string.camelize();
```

Return Value

- Returns a string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


  var str = 'background-color';


  alert("background-color.camelize() : " + str.camelize() );

```

```
      str = '-moz-binding';

      alert("-moz-binding.camelize() : " + str.camelize() );


    }


    </script>

    </head>


    <body>


      <p>Click the button to see the result.</p>

      <br />

      <br />

      <input type="button" value="Result" onclick="showResult();"/>


    </body>

    </html>
```

capitalize() Method

This method capitalizes the first letter of a string and downcases all the others.

Syntax

```
string.capitalize();
```

Return Value

- Returns a string.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

  var str = 'hello';

  alert("hello.capitalize() : " + str.capitalize() );

  str = 'HELLO WORLD!';
  alert("HELLO WORLD!.capitalize() : " + str.capitalize() );

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
```

```
      <input type="button" value="Result" onclick="showResult();"/>


    </body>

    </html>
```

dasherize() Method

This method replaces every instance of the underscore character ("_") by a dash ("-").

Syntax

```
string.dasherize();
```

Return Value

- Returns a string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


  var str = 'border_bottom_width';


  alert("border_bottom_width.dasherize():"+str.dasherize());
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

empty() Method

This method checks if the string is empty. This method is different from blank(). See example carefully.

Syntax

```
string.empty();
```

Return Value

- Returns a boolean value. If string is empty, then it returns true otherwise false.

Example

```
<html>
```

```
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

  var str = '';
  alert("''.empty() : " + str.empty() );
  alert("''.blank() : " + str.blank() );

  var str = ' ';
  alert("' '.empty() : " + str.empty() );
  alert("' '.blank() : " + str.blank() );

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
```

```
    <input type="button" value="Result" onclick="showResult();"/>


  </body>
  </html>
```

endsWith() Method

This method checks if the string ends with substring.

Syntax

```
  string.endsWith( substring );
```

Return Value

- Returns a boolean value. If string ends with the substring, then it returns true otherwise false.

Example

```
  <html>
  <head>
  <title>Prototype examples</title>
  <script type="text/javascript"
     src="/javascript/prototype.js">
  </script>
  <script>


  function showResult(){


    var str = 'slaughter';

```

```
      alert("First Check : " + str.endsWith("ghter") );

      alert("Second Check : " + str.endsWith("TATA") );

    }


    </script>

    </head>


    <body>


      <p>Click the button to see the result.</p>

      <br />

      <br />

      <input type="button" value="Result" onclick="showResult();"/>


    </body>

    </html>
```

escapeHTML() Method

This method converts HTML special characters to their entity equivalents.

Syntax

```
    string.escapeHTML();
```

Return Value

- Returns a string.

Example

```
    <html>
```

```
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

  var str = '<div class="article">This is an article</div>';

  alert( str.escapeHTML() );
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

evalJSON() Method

This method evaluates the JSON in the string and returns the resulting object. If the optional sanitize parameter is set to true, the string is checked for possible malicious attempts and eval is not called if one is detected.

Syntax

```
string.evalJSON([sanitize = false]);
```

Return Value

- Returns a string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


  var str = '{ "name": "Violet", "occupation": "character" }';

  var person = str.evalJSON();


  alert( "Name :" + person.name);

  alert( "Occupation :" + person.occupation);

}
```

```
</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

evalScripts() Method

This method evaluates the content of any script block present in the string. Returns an array containing the value returned by each script.

Syntax

```
ScriptString.evalScripts();
```

Return Value

- Returns an array containing the value returned by each script.

Example

```
<html>

<head>

<title>Prototype examples</title>

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

```
     src="/javascript/prototype.js">
  </script>
  <script>

  function showResult(){

    var str = 'other HTML... <script>2 + 2</script>';
    alert( str.evalScripts());

  }

  </script>
  </head>

  <body>

    <p>Click the button to see the result.</p>
    <br />
    <br />
    <input type="button" value="Result" onclick="showResult();"/>

  </body>
  </html>
```

This returns:

```
  [4]
```

extractScripts() Method

This method extracts the content of any script block present in the string and returns them as an array of strings.

Syntax

```
ScriptString.extractScripts();
```

Return Value

- Returns an array of strings.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


  var str = 'other HTML... <script>2 + 2</script>';

  alert( str.extractScripts());


}


</script>

</head>
```

```

    <body>


      <p>Click the button to see the result.</p>

      <br />

      <br />

      <input type="button" value="Result" onclick="showResult();"/>


    </body>

    </html>
```

This returns:

```
    [2 + 2 ]
```

gsub() Method

This method returns the string with every occurence of a given pattern replaced by either a regular string, the returned value of a function or a Template string. The pattern can be a string or a regular expression.

Syntax

```
    string.gsub(pattern, replacement);
```

Return Value

- Returns a string.

Example

```
    <html>

    <head>

    <title>Prototype examples</title>
```

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

function showResult(){

  var mouseEvents = 'click dblclick mousedown';

  alert( "mouseEvents :" + mouseEvents.gsub(/\s+/, ', ') );

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

include () Method

This method checks if the string contains a substring.

Syntax

```
string.include(substring);
```

Return Value

- Returns a boolean value. If it finds subsrting then it returns true, otherwise false.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


  var str = 'Prototype framework';


  alert( "First Check :" + str.include('framework' ) );

  alert( "Second Check :" + str.include('No framework' ) );


}


</script>
```

```
</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

inspect() Method

This method returns a debug-oriented version of the string, i.e., wrapped in single or double quotes, with backslashes and quotes escaped.

Syntax

```
string.inspect([useDoubleQuotes = false]);
```

Return Value

- Returns a debug-oriented version of the string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">
```

```
</script>
<script>

function showResult(){

  var str = 'I\'m so happy.';

  alert( "Check without inspect :" + str );
  alert( "First Check :" + str.inspect() );
  alert( "Second Check :" + str.inspect( true ) );

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

This returns the following value in actual but it will return different values in alert dialog or the console.

```
Check without inspect: I'm so happy.

First Check : '\'I\\\'m so happy.\''

Second Check : '"I'm so happy."'
```

interpolate() Method

This method treats the string as a Template and fills it with object's properties.

Syntax

```
string.inspect([useDoubleQuotes = false]);
```

Return Value

- Returns an interpolated string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


  var str = "#{animals} on a #{transport}";


  alert(str.interpolate({animals:"Cow",transport:"Train"}));

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

isJSON() Method

This method checks if the string is valid JSON by the use of regular expressions. This security method is called internally.

Syntax

```
string.isJSON();
```

Return Value

- Returns a boolean value.

Example

```
<html>

<head>
```

```
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

  var str = "something";
  alert("something.isJSON() : " + str.isJSON());
  var str = "\"something\"";
  alert("\"something\" : " + str.isJSON());
  var str = "{ foo: 42 }";
  alert("{ foo: 42 } : " + str.isJSON());
  var str = "{ \"foo\": 42 }";
  alert("{ \"foo\": 42 } : " + str.isJSON());
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>
```

```
</body>

</html>
```

parseQuery() Method

This method parses a URI-like query string and returns an object composed of parameter/value pairs. This method is similar to the toQueryParams();

This method is targeted at parsing query strings (hence the default value of "&" for the separator argument).

Syntax

```
string.parseQuery([separator = '&']);
```

Return Value

- Returns an object having key value pairs.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


  var str = "http://www.example.com?section=blog&id=45#comments";

  var obj = str.parseQuery();
```

```
    alert ( "obj.section :  " + obj.section );
    alert ( "obj.id :  " + obj.id );

    var str = "tag=ruby%20on%20rails";
    var obj = str.parseQuery();
    alert ( "obj.tag:  " + obj.tag );

  }

  </script>
  </head>

  <body>

    <p>Click the button to see the result.</p>
    <br />
    <br />
    <input type="button" value="Result" onclick="showResult();"/>

  </body>
  </html>
```

scan() Method

This method allows iterating over every occurrence of the given pattern (which can be a string or a regular expression).

Syntax

```
string.scan(pattern, iterator);
```

Return Value

- Returns the original string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


  var str = "apple, pear & orange";

  alert ("Actaul Alert");

  alert ( str.scan(/\w+/, alert ) );

}


</script>

</head>


<body>
```

```
      <p>Click the button to see the result.</p>

      <br />

      <br />

      <input type="button" value="Result" onclick="showResult();"/>


    </body>

    </html>
```

Following example can be used to populate an array:

```
    <html>

    <head>

    <title>Prototype examples</title>

    <script type="text/javascript"

       src="/javascript/prototype.js">

    </script>

    <script>


    function showResult(){


      var fruits = [];

      var str = "apple, pear & orange";

      str.scan(/\w+/, function(match){ fruits.push(match[0])});

      fruits.inspect();

      alert ( "fruits[0] : " + fruits[0]);

      alert ( "fruits[1] : " + fruits[1]);

      alert ( "fruits[2] : " + fruits[2]);

    }
```

```
</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

startsWith() Method

This method checks if the string starts with a substring.

Syntax

```
string.startsWith(substring);
```

Return Value

- Returns the boolean value.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
```

```
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

  var str = "Prototype JavaScript";
  alert ("First Check : " + str.startsWith("JavaScript") );
  alert ("Second Check : " + str.startsWith("Prototype") );

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

strip() Method

This method strips all leading and trailing whitespace from a string.

Syntax

```
string.strip();
```

Return Value

- Returns the trimmed string.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

  var str = "        Prototype JavaScript    ";
  alert ("Before strip : " + str );
  alert ("After strip : " + str.strip() );

}

</script>
</head>

<body>
```

```
    <p>Click the button to see the result.</p>

    <br />

    <br />

    <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

stripScripts() Method

This method strips a string of anything that looks like an HTML script block.

Syntax

```
string.stripScripts();
```

Return Value

- Returns the trimmed string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){
  var str = 'a &lt;a href="#"&gt;link&lt;/a&gt; ' +
            '    &lt;script&gt; ' +
            '         alert("Hello!") ' +
            '    &lt;/script&gt;';

  var str = str.unescapeHTML();

  alert( str);
  alert (str.stripScripts() );
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

This will produce the following result:

```
  a <a href="#">link</a>
```

stripTags() Method

This method strips a string of any HTML tag.

Syntax

```
string.stripTags();
```

Return Value

- Returns the stripped string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

  var str = 'a &lt;a href="#"&gt;link&lt;/a&gt; ' +

            '    &lt;script&gt; ' +

            '         alert("Hello!") ' +

            '    &lt;/script&gt;';


  var str = str.unescapeHTML();


  alert( str);

  alert (str.stripTags() );
}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

sub() Method

This method returns a string with the first count occurrences of pattern replaced by either a regular string, the returned value of a function or a Template string. Pattern can be a string or a regular expression.

Syntax

```
string.sub(pattern, replacement[, count = 1]);
```

Return Value

- Returns a string.

Example

```
<html>

<head>
```

```
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){
  var fruits = 'apple pear orange';

  alert("fruits.sub(' ', ', '):"+  fruits.sub(' ', ', '));
  alert("fruits.sub(' ', ', ', 1): " +  fruits.sub(' ', ', ', 1));
  alert("fruits.sub(' ', ', ', 2): " +  fruits.sub(' ', ', ', 2));

  var str = fruits.sub(/\w+/,
      function(match){ return match[0].capitalize() + ','}, 2);
  alert(str);

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
```

```
      <input type="button" value="Result" onclick="showResult();"/>


    </body>
    </html>
```

succ() Method

This method is used internally by the ObjectRange and converts the last character of the string to the following character in the Unicode alphabet.

Syntax

```
string.sub(pattern, replacement[, count = 1]);
```

Return Value

- Returns a string.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>


function showResult(){
   var str = "a";
   alert("a.succ : " + str.succ() );
   var str = "aaaa";
```

```
       alert("aaaa.succ : " + str.succ() );

       var str = "abcd";

       alert("abcd.succ : " + str.succ() );


    }


    </script>

    </head>


    <body>


      <p>Click the button to see the result.</p>

      <br />

      <br />

      <input type="button" value="Result" onclick="showResult();"/>


    </body>

    </html>
```

times() Method

This method concatenates the string count times.

Syntax

```
    string.times(count) ;
```

Return Value

- Returns a concatenated string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   var str = "a";

   alert("a.times(3) : " + str.times(3) );

   var str = "aaaa";

   alert("aaaa.times(2) : " + str.times(2) );

   var str = "abcd";

   alert("abcd.times(4) : " + str.times(4) );


}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />
  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

toArray() Method

This method splits the string character-by-character and returns an array with the result.

Syntax

```
string.toArray() ;
```

Return Value

- Returns an array of characters.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   var str = "a";


   alert("a.toArray() : " + str.toArray()[0] );

```

```
      var str = "hello world!";

      var arr = str.toArray();

      arr.each(function(alpha){

         alert(alpha);

      });

   }


   </script>

   </head>


   <body>


     <p>Click the button to see the result.</p>

     <br />

     <br />

     <input type="button" value="Result" onclick="showResult();"/>


   </body>

   </html>
```

toJSON() Method

This method returns a JSON string.

Syntax

```
   string.toJSON() ;
```

Return Value

- Returns a JSON string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   var str = 'The "Quoted" chronicles';


   alert("JSON String : " + str.toJSON() );

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>
```

```
</html>
```

toQueryParams () Method

This method parses a URI-like query string and returns an object composed of parameter/value pairs. This method is similar to parseQuery().

This method is really targeted at parsing query strings (hence the default value of "&" for the separator argument).

Syntax

```
string.toQueryParams ([separator = '&']);
```

Return Value

- Returns an object having key value pairs.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


  var str = "http://www.example.com?section=blog&id=45#comments";

  var obj = str.toQueryParams ();


  alert ( "obj.section :  " + obj.section );
```

```
      alert ( "obj.id :  " + obj.id );


      var str = "tag=ruby%20on%20rails";

      var obj = str.toQueryParams ();

      alert ( "obj.tag:  " + obj.tag );


    }


    </script>

    </head>


    <body>


      <p>Click the button to see the result.</p>

      <br />

      <br />

      <input type="button" value="Result" onclick="showResult();"/>


    </body>

    </html>
```

truncate() Method

This method truncates a string to the given length and appends a suffix to it (indicating that it is only an excerpt).

If unspecified, the length parameter defaults to 30 and the suffix to "...".

Syntax

```
    string.truncate([length = 30[, suffix = '...']]) ;
```

Return Value

- Returns a truncated string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){

   var str = 'A random sentence whose length exceeds 30 characters.';


   alert("str.truncate() : " + str.truncate() );

   alert("str.truncate( 10 ) : " + str.truncate(10) );

   alert("str.truncate( 10, '...') : " +

          str.truncate(10, '...') );

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>
```

```
    <br />

    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

underscore() Method

This method converts a camelized string into a series of words separated by an underscore ("_").

Syntax

```
  string.underscore() ;
```

Return Value

- Returns a truncated string.

Example

```
  <html>

  <head>

  <title>Prototype examples</title>

  <script type="text/javascript"

     src="/javascript/prototype.js">

  </script>

  <script>


  function showResult(){

     var str = 'borderBottomWidth';

     alert("str.underscore() : " + str.underscore() );


  }


  </script>

  </head>


  <body>


    <p>Click the button to see the result.</p>

    <br />

    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

unescapeHTML() Method

This method strips tags and converts the entity forms of special HTML characters to their normal form.

Syntax

```
   string.unescapeHTML()  ;
```

Return Value

- Returns an unescaped HTML trimmed string.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

   var str = 'x &gt; 10';
   alert("First : " + str.unescapeHTML() );
   var str = "<h1>Pride &amp; Prejudice</h1>"
   alert("Second : " + str.unescapeHTML() );

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>
```

```
</body>

</html>
```

unfilterJSON() Method

This method strips comment delimiters around Ajax JSON or JavaScript responses. This security method is called internally.

Syntax

```
string.unfilterJSON([filter = Prototype.JSONFilter])  ;
```

Return Value

- Returns a stripped string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


   var str = '/*-secure-\n{"name": "Violet", "age": 25}\n*/';

   alert(str. unfilterJSON() );


}
```

```
</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

7. Prototype – Array Processing

Prototype extends all the native JavaScript arrays with quite a few powerful methods.

This is done in two ways:

- It mixes in the Enumerable module, which brings a ton of methods in already.
- It adds quite a few extra methods, which are documented in this section.

Using Iterators

One important support provided by Prototype is that you can use java like iterator in JavaScript. See the difference below:

Traditional way of writing a **for** loop:

```
for (var index = 0; index < myArray.length; ++index) {

  var item = myArray[index];

  // Your code working on item here...

}
```

Now if you are using Prototype, then you can replace the above code as follows:

```
myArray.each(function(item) {

  // Your code working on item here...

});
```

Here is the list of all the functions with examples dealing with Array.

Prototype Array Methods

NOTE: Make sure you have the prototype.js version of 1.6.

Methods	Description
clear()	Clears the array (makes it empty).

clone()	Returns a duplicate of the array, leaving the original array intact.
compact()	Returns a new version of the array, without any null/undefined values.
each()	Iterates over the array in ascending numerical index order.
first()	Returns the first item in the array, or undefined if the array is empty.
flatten()	Returns a "flat" (one-dimensional) version of the array.
from()	Clones an existing array or creates a new one from an array-like collection.
indexOf()	Returns the position of the first occurrence of the argument within the array.
inspect()	Returns the debug-oriented string representation of an array.
last()	Returns the last item in the array, or undefined if the array is empty.
reduce()	Reduces arrays: one-element arrays are turned into their unique element, while multiple-element arrays are returned untouched.
reverse()	Returns the reversed version of the array. By default, directly reverses the original. If inline is set to false, uses a clone of the original array.
size()	Returns the size of the array.
toArray()	This is just a local optimization of the mixed-in toArray from Enumerable.
toJSON()	Returns a JSON string.
uniq()	Produces a duplicate-free version of an array. If no duplicates are found, the original array is returned.

without()	Produces a new version of the array that does not contain any of the specified values.

clear() Method

This method clears the array (makes it empty).

Syntax

```
array.clear()  ;
```

Return Value

- Returns an empty array.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


   var guys = ['Sam', 'Justin', 'Andrew', 'Dan'];

   alert("Before using clear" );

   guys.each(function(item) {

      alert(item );

   });
```

```
      guys.clear();

      alert("After using clear" );

      guys.each(function(item) {

         alert(item );

      });


   }


   </script>

   </head>


   <body>


     <p>Click the button to see the result.</p>

     <br />

     <br />

     <input type="button" value="Result" onclick="showResult();"/>


   </body>

   </html>
```

clone() Method

This method returns a duplicate of the array, leaving the original array intact.

Syntax

```
array.clone()  ;
```

Return Value

- Returns a new clone array.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


   var oldArr = ['Sam', 'Justin', 'Andrew', 'Dan'];

   var newArr = oldArr.clone();

   alert("Guys in old Arr" );

   oldArr.each(function(item) {

      alert(item );

   });

   alert("Guys in new Arr" );

   newArr.each(function(item) {

      alert(item );

   });


}


</script>
```

```
</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

compact() Method

This method returns a new version of the array, without any null/undefined values.

Syntax

```
array.compact()  ;
```

Return Value

- Returns a new array without any null of undefined values.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>
```

```
<script>

function showResult(){

   var oldArr =['Sam','Justin','Andrew','Dan', null, undefined];
   alert("Guys in old Arr" );
   oldArr.each(function(item) {
      alert(item );
   });
   var newArr = oldArr.compact();
   alert("Guys in new Arr" );
   newArr.each(function(item) {
      alert(item );
   });

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>
```

```
</body>

</html>
```

each() Method

This method iterates over the array in ascending numerical index order.

Syntax

```
array.each(iterator)  ;
```

Return Value

- Returns an array element.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


   var oldArr = ['Sam', 'Justin', 'Andrew', 'Dan', null, undefined];

   alert("Guys in old Arr" );

   oldArr.each(function(item) {

      alert(item );

   });
}


</script>
</head>


<body>


  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>


</body>
</html>
```

We are using each method in many other methods related to array.

first() Method

This method returns the first item in the array, or undefined if the array is empty.

Syntax

```
array.first()  ;
```

Return Value

- Returns the first element of the array.

Example

```
<html>
<head>
```

```
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

   var arr = ['Sam', 'Justin', 'Andrew', 'Dan'];
   alert("First element is : "  + arr.first() );

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

flatten() Method

This method returns a *flat* (one-dimensional) version of the array. Nested arrays are recursively injected *inline.*

This can prove very useful when handling the results of a recursive collection algorithm.

Syntax

```
array.flatten()  ;
```

Return Value

- Returns a flat (one - dimensional) array.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


   var arr =['frank',['bob','lisa'],['jill',['tom','sally']]];

   // Now flatten this array.

   var newArr = arr.flatten();

   newArr.each(function(item) {

     alert(item);

   });

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

from() Method

This method clones an existing array or creates a new one from an array-like collection.

This is an alias for the $A() method.

Syntax

```
Array.from(iterable)  ;
```

Return Value

- Returns an array having all the elements from the passed selector.

Example

```
<html>
```

```
<head>
<title>Prototype examples</title>
   <script type="text/javascript"
   src="/javascript/prototype.js">
   </script>
<script>

function showOptions(){
   var NodeList = $('employees').getElementsByTagName('option');
   var nodes = Array.from(NodeList);

   nodes.each(function(node){
      alert(node.nodeName + ': ' + node.innerHTML);
   });
}
</script>
</head>

<body>

   <select id="employees" size="10" >
      <option value="5">Mohtashim, Mohd</option>
      <option value="8">Debi, Patnaik</option>
      <option value="1">Madisetti, Praveen</option>
   </select>
   <br />
   <input type="button" value="Show the options"

             onclick="showOptions();"/>


</body>

</html>
```

indexOf() Method

This method returns the position of the first occurrence of the argument within the array. If the argument doesn't exist in the array, returns -1.

NOTE: First index of the Array starts from zero.

Syntax

```
array.indexOf(value)  ;
```

Return Value

- Returns the position of the first occurrence of the argument within the array. If the argument doesn't exist in the array, returns -1.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){
   var arr = [3, 5, 6, 1, 20];

   alert("Found position : "  + arr.indexOf(6) );

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

inspect() Method

This method returns the debug-oriented string representation of an array.

Syntax

```
array.inspect()  ;
```

Return Value

- Returns a String.

Example

```
<html>
```

```
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

   var arr = ['Apples', {good: 'yes', bad: 'no'}, 3, 34];
   alert("Debuig Version : "  + arr.inspect() );
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

last() Method

This method returns the last item in the array, or undefined if the array is empty.

Syntax

```
array.last()  ;
```

Return Value

- Returns the last element of the array.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


   var arr = ['Apples', {good: 'yes', bad: 'no'}, 3, 34];

   alert("Last Element : "  + arr.last() );

}


</script>

</head>


<body>
```

```
    <p>Click the button to see the result.</p>
    <br />
    <br />
    <input type="button" value="Result" onclick="showResult();"/>

  </body>
  </html>
```

reduce() Method

This method reduces arrays: one-element arrays are turned into their unique element, while multiple-element arrays are returned untouched.

Syntax

```
  array.reduce()  ;
```

Return Value

- If it is one-element array, then it turned into their unique element.
- If it is multi-elements array, then it remains untouched.

Example

```
  <html>
  <head>
  <title>Prototype examples</title>
  <script type="text/javascript"
     src="/javascript/prototype.js">
  </script>
  <script>
```

```
function showResult(){


   var arr = [10];

   alert("Reduced Element : "  + arr.reduce());

   var arr = [10, 20, 30, 40];

   alert("Reduced Element : "  + arr.reduce().inspect());

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

reverse() Method

This method returns the reversed version of the array. By default, directly reverses the original. If inline is set to false, uses a clone of the original array.

Syntax

```
array.reverse([inline = true])  ;
```

Return Value

- Returns reversed version of the array.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

   var arr = [3, 5, 6, 1, 20];
   alert("Reversed Elements : "  + arr.reverse(true).inspect());
   alert("Reversed Elements : "  + arr.reverse(false).inspect());
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>
```

```
</body>

</html>
```

size() Method

This method returns the size of the array.

Syntax

```
array.size()  ;
```

Return Value

- Returns size of the array.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


   var arr = [3, 5, 6, 1, 20];

   alert("Element Size : "  + arr.size());

}
```

```
</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

toArray() Method

This method aliases to clone, avoiding the default iterative behavior.

Syntax

```
array.toArray()  ;
```

Return Value

- Returns a new cloned array.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">
```

```
</script>
<script>

function showResult(){

   var arr = [3, 5, 6, 1, 20];
   var newArr = arr.toArray();
   alert("New Array : "  + newArr.toArray().inspect());
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

toJSON() Method

This method returns a JSON string.

Syntax

```
array.toJSON()  ;
```

Return Value

- Returns a JSON string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


   var arr = ['a', {b: null}];

   alert("JSON String : "  + arr.toJSON() );

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>
```

```
    <br />

    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

uniq() Method

This method produces a duplicate-free version of an array. If no duplicates are found, the original array is returned.

Syntax

```
  array.uniq()  ;
```

Return Value

- Returns a new array without duplicate elements.

Example

```
  <html>

  <head>

  <title>Prototype examples</title>

  <script type="text/javascript"

     src="/javascript/prototype.js">

  </script>

  <script>


  function showResult(){


     var arr = ['Sam', 'Justin', 'Andrew', 'Dan', 'Sam'];
```

```
      alert("Old Array : "  + arr.inspect() );

      alert("New Array : "  + arr.uniq().inspect() );

   }


   </script>

   </head>


   <body>

     <p>Click the button to see the result.</p>

     <br />

     <br />

     <input type="button" value="Result" onclick="showResult();"/>

   </body>

   </html>
```

without() Method

This method produces a new version of the array that does not contain any of the specified values.

Syntax

```
   array.without(value...)  ;
```

Return Value

- Returns a new array without specified elements.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

   var arr = ['Sam', 'Justin', 'Andrew', 'Dan', 'Sam'];
   alert("Without Sam : "  + arr.without('Sam').inspect() );
   alert("Without Dan : "  + arr.without('Dan').inspect() );
}

</script>
</head>

<body>
  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>
</body>
</html>
```

8. Prototype – Hash Processing

Hash can be thought of as an associative array binding unique keys to values. Only difference is that you can use any string as an index instead of just using a number as index.

Creating a Hash

There are two ways to construct a Hash instance:

- Use JavaScript keyword *new*.
- Using Prototype Utility function *$H*.

To create an empty hash you call any of the constructor methods without arguments, too.

Following is the example showing how to create hash, setting values and getting values in a simple way:

```
// Creating Hash

var myhash = new Hash();

var yourhash = new Hash( {fruit: 'apple'} );

var hishash = $H( {drink: 'pepsi'} );


// Set values in terms of key and values.

myhash.set('name', 'Bob');


// Get value of key 'name' as follows.

myhash.get('name');

yourhash.get('fruit');

hishash.get('drink');


// Unset a key & value

 myhash.unset('name');
```

```
    yourhash.unset('fruit');

    hishash.unset('drink');
```

Prototype provides a wide range of methods to evaluate Hash with ease. This tutorial will explain every method in detail with suitable examples.

Here is a complete list of all the methods related to Hash.

Prototype Hash Methods

NOTE: Make sure you at least have the version 1.6 of prototype.js.

Methods	Description
clone()	Returns a clone of hash.
each()	Iterates over the name/value pairs in the hash.
get()	Returns the value of the hash key's property.
inspect()	Returns the debug-oriented string representation of the hash.
keys()	Provides an Array of keys (that is, property names) for the hash.
merge()	Merges object to hash and returns the result of that merge.
remove()	Removes keys from a hash and returns their values. This method has been deprecated in version 1.6.
set()	Sets the hash key's property to value and returns value.
toJSON()	Returns a JSON string.
toObject()	Returns a cloned, vanilla object.
toQueryString()	Turns a hash into its URL-encoded query string representation.
unset()	Deletes the hash key's property and returns its value.

update()	Updates hash with the key/value pairs of object. The original hash will be modified.
values()	Collects the values of a hash and returns them in an array.

clone() Method

This method returns a clone of hash.

Syntax

```
hash.clone() ;
```

Return Value

- Returns a clone of hash.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


   var hash = new Hash({ a: 'apple'});

   var clone = hash.clone();

   alert("Value in hash : " + hash.get('a') );

   alert("Value in clone : " + clone.get('a') );
```

```
      hash.unset('a');
      clone.unset('a');
   }

   </script>
   </head>

   <body>

     <p>Click the button to see the result.</p>
     <br />
     <br />
     <input type="button" value="Result" onclick="showResult();"/>

   </body>
   </html>
```

each() Method

This method is used to iterate over the key/value pairs in the hash.

Syntax

```
hash.each(iterator)  ;
```

Return Value

- Returns a key and value pair.

Example

```
<html>
```

```
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

   var h = $H({ version: 1.5, author: 'Sam Stephenson' });
   h.each(function(pair) {
     alert(pair.key + ' = "' + pair.value + '"');
   });
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

get() Method

This method is used to the value of the hash key’s property.

Syntax

```
hash.get(key)  ;
```

Return Value

- Returns a value corresponding to a paased key.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


  var h = new Hash({ a: 'apple', b: 'banana', c: 'coconut' });


  alert("Value of a : " + h.get('a') );

  // -> 'apple'


  alert("Value of b : " + h.get('b') );

  // -> 'banana'

```

```
    alert("Value of d : " + h.get('d') );
    // -> undefined
  }

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

inspect() Method

This method is used to return the debug-oriented string representation of the hash.

Syntax

```
hash.inspect()  ;
```

Return Value

- Returns a string representation of the hash.

Example

```
<html>
```

```
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

  var h = $H({ name: 'Prototype', version: 1.6 });

  alert("String Representation : " + h.inspect() );
  // -> "<#Hash:{name: 'Prototype', version: 1.6}>"

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
```

```
</html>
```

keys() Method

This method returns an Array of all keys for the hash.

Syntax

```
hash.keys()  ;
```

Return Value

- Returns an array of the strings having all the keys of the hash.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


  var h = $H({ name: 'Prototype', version: 1.6 });

  var arr = h.keys();


  arr.each(function(item) {

      alert(item );

   });
}


</script>
</head>


<body>


  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>


</body>
</html>
```

merge() Method

This method merges object to hash and returns the result of that merge. Duplicate keys will cause an overwrite.

Syntax

```
hash.merge(object) ;
```

Return Value

- Returns the resultant merged hash.

Example

```
<html>
<head>
```

```
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

  var h = $H({ name: 'Prototype', version: 1.6 });
  h = h.merge({ version: 1.6, author: 'Sam' });

  h.each(function(pair) {
     alert(pair.key + ' = "' + pair.value + '"');
  });

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>
```

```
</body>

</html>
```

remove() Method

This method removes keys from a hash and returns their values. **This method is not available since v1.6.0.**

Syntax

```
hash.remove(key1, key2...) ;
```

Return Value

- Returns the resultant merged hash.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


  var h = new Hash({ a:'apple', b:'banana', c:'coconut' });

  alert("Before removing elements : " + h.inspect() );


  // Remove two elements.

  h.remove('a', 'c');
```

```
   alert("After removing elements : " + h.inspect() );
 }


</script>
</head>


<body>


   <p>Click the button to see the result.</p>
   <br />
   <br />
   <input type="button" value="Result" onclick="showResult();"/>


</body>
</html>
```

set() Method

This method sets the hash key’s property to value and returns value.

Syntax

```
hash.set(key, value) ;
```

Return Value

- Returns the value of the hash.

Example

```
<html>
```

```
<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


 var h = new Hash({ a: 'apple', b: 'banana', c: 'coconut' });

 alert( "h.inspect() : " + h.inspect() );


 h.set('d', 'orange');

 alert( "h.inspect() : " + h.inspect() );


 h.set('a', 'kiwi');

 alert( "h.inspect() : " + h.inspect() );


}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />
```

```
    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

toJSON() Method

This method returns a JSON string.

Syntax

```
hash.toJSON() ;
```

Return Value

- Returns a JSON string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


 var h = new Hash({ a: 'apple', b: 'banana', c: 'coconut' });

```

```
     alert( "h.toJSON() : " + h.toJSON() );


   }


   </script>

   </head>


   <body>


     <p>Click the button to see the result.</p>

     <br />

     <br />

     <input type="button" value="Result" onclick="showResult();"/>


   </body>

   </html>
```

toObject() Method

This method returns a cloned, vanilla object.

Syntax

```
   hash.toObject() ;
```

Return Value

- Returns a cloned, vanilla object.

Example

```
   <html>
```

```
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

 var h = new Hash({ a: 'apple', b: 'banana', c: 'coconut' });
 var obj = h.toObject();

 alert( "Object.inspect(h) : " + Object.inspect(h) );
 alert( "Object.inspect(obj) : " + Object.inspect(obj) );

 alert( "h.get('a') : " + h.get('a') );
 alert( "Object.keys(obj) : " + Object.keys(obj) );

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
```

```
      <br />

      <input type="button" value="Result" onclick="showResult();"/>


    </body>

    </html>
```

toQueryString() Method

This method turns a hash into its URL-encoded query string representation.

This is a form of serialization, and is mostly useful to provide complex parameter sets for stuff such as objects in the Ajax namespace (e.g. Ajax.Request).

Undefined-value pairs will be serialized as if empty-valued. Array-valued pairs will get serialized with one name/value pair per array element. All values get URI-encoded using JavaScript's native encodeURIComponent function.

Syntax

```
hash.toQueryString() ;
```

Return Value

- Returns a URL-encoded query string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){
 var h = $H({action:'ship',order_id:123,fees: ['f1', 'f2']});
 alert( "h.toQueryString() : " + h.toQueryString() );

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

unset() Method

This method deletes the hash key's property and returns its value. If it does not find that key in the hash, then it does nothing and returns undefined.

Syntax

```
hash.unset(key) ;
```

Return Value

- Returns value of the deleted key.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


 var h = new Hash({ a: 'apple', b: 'banana', c: 'coconut' });

 // Returns 'apple'

 alert( "h.unset('a') : " +  h.unset('a') );


}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>
```

```
</body>

</html>
```

update() Method

This method updates hash with the key/value pairs of object. The original hash will be modified.

Duplicate keys will cause an overwrite of hash keys. If it does not find a key, then it will add it as a new key/value pair.

Syntax

```
hash.update(object) ;
```

Return Value

- Returns a modified hash.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


 var h = new Hash({ a: 'apple', b: 'banana', c: 'coconut' });


 var t = h.update({a: 'amrood', d:'drum'});
```

```
     alert( "New Hash : " +  t.inspect() );

   }


   </script>
   </head>

   <body>

     <p>Click the button to see the result.</p>
     <br />
     <br />
     <input type="button" value="Result" onclick="showResult();"/>

   </body>
   </html>
```

values() Method

This method collect the values of a hash and returns them in an array.

Syntax

```
hash.values() ;
```

Return Value

- Returns an array of all the values.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


 var h = $H({ name: 'Prototype', version: 1.6 });

 var arr = h.values();


 arr.each(function(item) {

      alert(item );

 });


}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />
```

```
    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

9. Prototype – Basic Object

Object is used by Prototype as a namespace and to call related function using *Object* object. This is used in the following two ways:

- If you are a simple developer, then you can use the existing functions like *inspect* or *clone*.
- If you are one who wishes to create their own objects like Prototype does, or explore objects as if they were hashes, will turn to *extend*, *keys* and *values*.

Prototype Object Methods

NOTE: Make sure you at least have the version 1.6 of prototype.js.

Methods	Description
clone()	Clones the passed object using shallow copy (copies all the original's properties to the result).
extend()	Copies all properties from the source to the destination object.
inspect()	Returns the debug-oriented string representation of the object.
isArray ()	Returns true if obj is an array, false otherwise.
isElement()	Returns true if obj is a DOM node of type 1, false otherwise.
isFunction()	Returns true if obj is of type function, false otherwise.
isHash()	Returns true if obj is an instance of the Hash class, false otherwise.
isNumber()	Returns true if obj is of type number, false otherwise.
isString()	Returns true if obj is of type string, false otherwise.
isUndefined()	Returns true if obj is of type undefined, false otherwise.

keys()	Treats any object as a Hash and fetches the list of its property names.
toHTML()	Returns the return value of obj's to HTML method if it exists, else runs obj through String.interpret.
toJSON()	Returns a JSON string.
toQueryString()	Turns an object into its URL-encoded query string representation.
values()	Treats any object as a Hash and fetches the list of its property values.

clone() Method

This method clones the passed object using the shallow copy. It copies all the original's properties to the result.

Syntax

```
object.clone()  ;
```

Return Value

- Returns a clone of object.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>



<script>
```

```
function showResult(){

   var o = { name: 'Prototype', version: 1.5,
             authors: ['sam', 'contributors'] };
   var o2 = Object.clone(o);

   o2.version = '1.5 weird';
   o2.authors.pop();

   alert( " Value of o.version : " + o.version );
   // Returns 1.5

   alert( " Value of o2.version : "  + o2.version );
   // Returns 1.5 weird

   alert( " Value of o.authors : " +  o2.authors );
   // Returns ['sam']
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
```

```
    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

extend() Method

This method copies all properties from the source to the destination object. This is used by Prototype to simulate inheritance by copying to prototypes.

Syntax

```
  Object.extend(dest, src);
```

Return Value

- Returns an altered object.

Example

```
   <html>

  <head>

  <title>Prototype examples</title>

  <script type="text/javascript"

     src="/javascript/prototype.js">

  </script>


  <script>


  function showResult(){


     var o1 = { name: 'Prototype', version: 1.5 };
```

```
       // An empty object

       var o2 = {};

       var o2 = Object.extend(o2, o1);


       alert( " Value of o2.name :" + o2.name );

       // Returns Prototype


       alert( " Value of o2.version :" + o2.version );

       // Returns 1.5


    }


    </script>

    </head>


    <body>


      <p>Click the button to see the result.</p>

      <br />

      <br />

      <input type="button" value="Result" onclick="showResult();"/>


    </body>

    </html>
```

inspect() Method

This method returns the debug-oriented string representation of the object.

undefined and *null* are represented as such.

Syntax

```
Object.inspect(obj);
```

Return Value

- Returns a string version of the object.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>

<script>

function showResult(){

   var o = [1,2,3,4,5];
   alert( "Value of Object.inspect() : " + Object.inspect(o) );

}

</script>
</head>
```

```
<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

isArray() Method

This method returns true if obj is an array, false otherwise.

Syntax

```
Object.isArray(obj);
```

Return Value

- Returns true if passed object is an array otherwise returns false.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>
```

```
function showResult(){

   var o1 = [1,2,3,4,5];
   var o2 = {name:"prototype", version:1.6};

   alert( "Object.isArray(o1): " + Object.isArray(o1) );
   alert( "Object.isArray(o2): " + Object.isArray(o2) );

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

isElement() Method

This method returns true if obj is a DOM node of type 1, false otherwise.

Syntax

```
Object.isElement(obj);
```

Return Value

- Returns true if the passed object is a DOM node of type 1, otherwise returns false.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){


   alert( "1: " + Object.isElement(new Element('div')) );

   alert( "2: " + Object.isElement(document.createElement('div')) );

   alert( "3: " + Object.isElement(document.createTextNode('foo')) );

   alert( "4: " + Object.isElement($('para')) );


}


</script>

</head>
```

```
<body>


  <p id="para">Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

isFunction() Method

This method returns true if obj is of type function, false otherwise.

Syntax

```
Object.isFunction(obj);
```

Return Value

- Returns true if obj is of type function, false otherwise.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>
```

```
<script>

function showResult(){

   alert( "test 1: " + Object.isFunction($) );
   alert( "test 2: " + Object.isFunction(123) );
   alert( "test 3: " + Object.isFunction(showResult) );
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

isHash() Method

This method returns true if obj is an instance of the Hash class, false otherwise.

Syntax

```
Object.isHash(obj);
```

Return Value

- Returns true if obj is an instance of the Hash class, false otherwise

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){


   myHash = new Hash({name:"prototype", version:1.6 });


   alert( "test 1: " + Object.isHash(new Hash({ })) );

   alert( "test 2: " + Object.isHash($H({ })) );

   alert( "test 3: " + Object.isHash({ }) );

   alert( "test 4: " + Object.isHash(myHash) );

}


</script>

</head>


<body>
```

```
    <p>Click the button to see the result.</p>

    <br />

    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

isNumber() Method

This method returns true if obj is of type number, false otherwise.

Syntax

```
Object.isNumber(obj);
```

Return Value

- Returns true if obj is of type number, false otherwise.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){
      var myNumber = 100;

      alert( "test 1: " + Object.isNumber(0) );
      alert( "test 2: " + Object.isNumber(1.2) );
      alert( "test 3: " + Object.isNumber("foo") );
      alert( "test 4: " + Object.isNumber(myNumber) );
   }

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

isString() Method

This method returns true if obj is of type string, false otherwise.

Syntax

```
Object.isString(obj);
```

Return Value

- Returns true if obj is of type string, false otherwise.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>

<script>

function showResult(){

   var myString = "This is string";

   alert( "test 1: " + Object.isString("foo") );
   alert( "test 3: " + Object.isString(123) );
   alert( "test 4: " + Object.isString(myString) );
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
```

```
    <br />

    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

isUndefined() Method

This method returns true if obj is of type undefined, false otherwise.

Syntax

```
Object.isUndefined(obj);
```

Return Value

- Returns true if obj is of type undefined, false otherwise.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){

   alert( "test 1: " + Object.isUndefined() );

   alert( "test 2: " + Object.isUndefined(undefined) );

   alert( "test 3: " + Object.isUndefined(null) );

   alert( "test 4: " + Object.isUndefined(0) );

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

keys() Method

This method treats any object as a Hash and fetches the list of its property names.

Syntax

```
   Object.keys(obj);
```

Return Value

- Returns an array of the keys.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>

<script>

function showResult(){

   var o = { name: 'Prototype', version: 1.6 };

   alert("Object.keys(o).inspect(): " + Object.keys(o).inspect());
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>
```

```
</body>

</html>
```

toHTML() Method

This method returns the return value of obj's to HTML method if it exists, else runs obj through *String.interpret* method.

Syntax

```
Object.toHTML(obj);
```

Return Value

- Returns the return value of obj's to HTML method if it exists, else runs obj through *String.interpret* method.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>



<script>



var Bookmark = Class.create({

  initialize: function(name, url) {

    this.name = name;

    this.url = url;

  },
  toHTML: function() {
    return '<a href="#{url}">#{name}</a>'.interpolate(this);
  }
});

var api = new Bookmark('Prototype', 'http://prototypejs.org');



function showResult(){
   alert("Test 1: " + Object.toHTML(api));
   alert("Test 2: " + Object.toHTML("Hello world!"));
   alert("Test 3: " + Object.toHTML());
   alert("Test 4: " + Object.toHTML(null));
   alert("Test 5: " + Object.toHTML(undefined));
   alert("Test 6: " + Object.toHTML(true));
   alert("Test 7: " + Object.toHTML(false));
   alert("Test 8: " + Object.toHTML(123));
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
```

```
      <br />

      <input type="button" value="Result" onclick="showResult();"/>


    </body>

    </html>
```

toJSON() Method

This method returns a JSON string.

Syntax

```
    Object.toJSON(obj);
```

Return Value

- Returns a JSON string.

Example

```
    <html>

    <head>

    <title>Prototype examples</title>

    <script type="text/javascript"

       src="/javascript/prototype.js">

    </script>


    <script>


    function showResult(){


       var o = {name: 'Prototype', Version: 1.6};
```

```
      alert("Object.toJSON(o): " + Object.toJSON(o));
   }


   </script>
   </head>


   <body>


     <p>Click the button to see the result.</p>
     <br />
     <br />
     <input type="button" value="Result" onclick="showResult();"/>


   </body>
   </html>
```

toQueryString() Method

This method turns an object into its URL-encoded query string representation.

This is a form of serialization, and is mostly useful to provide complex parameter sets for stuff such as objects in the Ajax namespace (e.g. Ajax.Request).

Syntax

```
Object.toQueryString(obj);
```

Return Value

- Returns an encoded string.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>

<script>

function showResult(){

   var o = { action: 'ship', order_id: 123,
             fees: ['f1', 'f2'], 'label': 'a demo' };

   alert("URI: " + Object.toQueryString(o) );
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>
```

```
</body>
</html>
```

values() Method

This method treats any object as a Hash and fetches the list of its property values.

Syntax

```
Object.values(obj);
```

Return Value

- Returns an array of the values.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>

<script>

function showResult(){

   var o = { name: 'Prototype', version: 1.6 };

   alert("Test: " + Object.values(o).inspect() );
}
```

```
</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

10. Prototype – Templating

Templates are used for formatting group of similar objects and to produce formatted output for these objects.

Prototype provides a *Template* class, which has two methods:

- **Template():** This is a constructor method, which is used to create a template object and call *evaluate()* method to apply template.
- **evaluate():** This method is used to apply a template to format an object.

There are three steps involved to create the formatted output.

- **Create a template:** This involves creating preformatted text. This text contains formatted content along with **#{fieldName}** values. These **#{fieldName}** values will be replaced by the actual values when *evaluate()* method will be called with the actual values.
- **Defining actual values:** This involves creating actual values in the form of Keys and Values. These Keys will be mapped in the template and will be replaced by the corresponding values.
- **Mapping Keys and replacing Values:** This is the final step where *evaluate()* will be called and all the keys available in the formatted object will be replaced by the defined values.

Example1

Step 1

Create a template.

```
var myTemplate = new Template('The \

               TV show #{title} was directed by #{author}.');
```

Step 2

Prepare our set of values, which will be passed in the above object to have a formatted output.

```
var record1 = {title: 'Metrix', author:'Arun Pandey'};

var record2 = {title: 'Junoon', author:'Manusha'};
var record3 = {title: 'Red Moon', author:'Paul, John'};

var record4 = {title: 'Henai', author:'Robert'};

var records = [record1, record2, record3, record4 ];
```

Step 3

Final step is filling up all the values in the template and producing final result as follows:

```
records.each( function(conv){

    alert( "Formatted Output : " + myTemplate.evaluate(conv) );

});
```

So, let's put all these three steps together:

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){


   //  Create template with formatted content and placeholders.

   var myTemplate = new Template('The \

                    TV show #{title} was directed by #{author}.');


   // Create hashes with the required values for placeholders
```

```
       var record1 = {title: 'Metrix', author:'Arun Pandey'};

       var record2 = {title: 'Junoon', author:'Manusha'};

       var record3 = {title: 'Red Moon', author:'Paul, John'};

       var record4 = {title: 'Henai', author:'Robert'};

       var records = [record1, record2, record3, record4 ];


       // Now apply template to produce desired formatted output

       records.each( function(conv){

           alert( "Formatted Output : " + myTemplate.evaluate(conv) );

       });

    }


    </script>

    </head>


    <body>


      <p>Click the button to see the result.</p>

      <br />

      <br />


      <input type="button" value="Result" onclick="showResult();"/>


    </body>

    </html>
```

This will produce the following result:

```
The TV show Metrix was directed by Arun Pandey.

The TV show Junoon was directed by Manusha.

The TV show Red Moon was directed by Paul, John.

The TV show Henai was directed by Robert.
```

11. Prototype – Enumerating

Enumerable class provides a large set of useful methods for enumerations. Enumerations are objects that act as collection of values.

Enumeration methods are mostly used to enumerate *arrays* and *hashes*. There are other objects as well like *ObjectRange* and various DOM- or AJAX-related objects where you can use enumeration methods.

The Context Parameter

Every method of Enumerable that takes an iterator also takes the context object as the next (optional) parameter. The context object is what the iterator is going to be binded to, so the **this** keyword inside it will point to the object.

```
var myObject = {};


['foo', 'bar', 'baz'].each(function(name, index) {
  this[name] = index;
}, myObject); // we have specified the context


myObject;
```

This will produce the following result:

```
{ foo: 0, bar: 1, baz: 2}
```

Using it Efficiently

- When you need to invoke the same method on all the elements, go with *invoke()*method.
- When you need to fetch the same property on all the elements, go with *pluck()*method.
- The *findAll/select* methods retrieve all the elements that match a given predicate. Conversely, the *reject()* method retrieves all the elements that do not match a predicate. In the specific case where you need both the sets, you can avoid looping twice: just use *partition()* method.

Here is a complete list of all the methods related to Enumerable.

Prototype Enumerable Methods

NOTE: Make sure you at least have the version 1.6 of prototype.js.

Methods	Description
all()	Determines whether all the elements are boolean-equivalent to true, either directly or through computation by the provided iterator.
any()	Determines whether at least one element is boolean-equivalent to true, either directly or through computation by the provided iterator.
collect()	Returns the results of applying the iterator to each element. Aliased as map().
detect()	Finds the first element for which the iterator returns true. Aliased by the find() method.
each()	It lets you iterate over all the elements in a generic fashion, then returns the Enumerable, thereby allowing chain-calling.
eachSlice()	Groups items in chunks based on a given size, with last chunk being possibly smaller.
entries()	Alias for the more generic toArray method.
find()	Finds the first element for which the iterator returns true. Convenience alias for detect().
findAll()	Returns all the elements for which the iterator returned true. Aliased as select().
grep()	Returns all the elements that match the filter. If an iterator is provided, it is used to produce the returned value for each selected element.

inGroupsOf()	Groups items in fixed-size chunks, using a specific value to fill up the last chunk if necessary.
include()	Determines whether a given object is in the Enumerable or not, based on the == comparison operator. Aliased as member().
inject()	Incrementally builds a result value based on the successive results of the iterator.
invoke()	Optimization for a common use-case of each() or collect(): invoking the same method, with the same potential arguments, for all the elements.
map()	Returns the results of applying the iterator to each element. Convenience alias for collect().
max()	Returns the maximum element (or element-based computation), or undefined if the enumeration is empty. Elements are either compared directly, or by first applying the iterator and comparing returned values.
member()	Determines whether a given object is in the Enumerable or not, based on the == comparison operator. Convenience alias for include().
min()	Returns the minimum element (or element-based computation), or undefined if the enumeration is empty. Elements are either compared directly, or by first applying the iterator and comparing returned values.
partition()	Partitions the elements in two groups: those regarded as true, and those considered false.
pluck()	Optimization for a common use-case of collect(): fetching the same property for all the elements. Returns the property values.
reject()	Returns all the elements for which the iterator returned false.
select()	Alias for the findAll() method.

size()	Returns the size of the enumeration.
sortBy()	Provides a custom-sorted view of the elements based on the criteria computed, for each element, by the iterator.
toArray()	Returns an Array representation of the enumeration. Aliased as entries().
zip()	Zips together (think of the zip on a pair of trousers) 2 + sequences, providing an array of tuples. Each tuple contains one value per original sequence.

all() Method

This method determines whether all the elements are boolean-equivalent to true, either directly or through computation by the provided iterator.

The optional context parameter is what the iterator function will be bound to. If used, the *this* keyword inside the iterator will point to the object given by the argument.

Syntax

```
Iterator.all([context]);
```

Return Value

- Returns boolean true value if all the values in the iterator are true.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>
```

```
<script>


function showResult(){


  alert( "[].all() : "  + [].all() );

  // true (empty arrays have no elements )


  alert("$R(1, 5).all() : " + $R(1, 5).all() );

  // true (all values in [1..5] are true-equivalent)


  alert("[0, 1, 2].all() : " + [0, 1, 2].all() );

  // false (with only one loop cycle: 0 is false-equivalent)


  alert([9, 10, 15].all(function(n) { return n >= 10; }) );

  //false (the iterator will return false on 9)

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>
```

```
</body>

</html>
```

any() Method

This method determines whether at least one element is boolean-equivalent to true, either directly or through computation by the provided iterator.

The optional context parameter is what the iterator function will be bound to. If used, the *this* keyword inside the iterator will point to the object given by the argument.

Syntax

```
Iterator.any([context]);
```

Return Value

- Returns boolean true value if atleast one value in the iterator is true.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>



<script>



function showResult(){



  alert( "[].any() : "  + [].any() );

  // false  (empty arrays have no elements)
```

```
    alert("$R(0, 2).any() : " + $R(0, 2).any() );
    // true (on the second loop cycle, 1 is true-equivalent)

    alert("[0, 1, 2].any() : " + [0, 1, 2].any() );
    // true (with 1 and 2 loop cycle is true )

    alert([9, 10, 15].any(function(n) { return n >= 10; }) );
    // true (the iterator will return true more than 10 )
  }

  </script>
  </head>

  <body>

    <p>Click the button to see the result.</p>
    <br />
    <br />
    <input type="button" value="Result" onclick="showResult();"/>

  </body>
  </html>
```

collect() Method

This method returns the results of applying the iterator to each element. This method is aliased as map() method.

The optional context parameter is what the iterator function will be bound to. If used, the *this* keyword inside the iterator will point to the object given by the argument.

Syntax

```
Iterator.collect([context]);
```

Return Value

- Returns the results of applying the iterator to each element.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){


  alert($R(1,5).collect(function(n) { return n * n; }) );


  // This returns [1, 4, 9, 16, 25]

}


</script>

</head>
```

```
<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

detect() Method

This method finds the first element for which the iterator returns true. This method is aliased as find() method.

Syntax

```
Iterator.detect([context]);
```

Return Value

- Returns first element for which the iterator returns true otherwise undefined.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>



<script>
```

```
function isPrime(n) {
  if (2 > n) return false;
  if (0 == n % 2) return (2 == n);
  for (var index = 3; n / index > index; index += 2)
    if (0 == n % index) return false;
  return true;
} // isPrime

function showResult(){
  alert($R(10,15).detect(isPrime) );
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

each() Method

This method is the cornerstone of Enumerable. It lets you iterate over all the elements in a generic fashion, then returns the Enumerable, thereby allowing the chain-calling.

The optional context parameter is what the iterator function will be bound to. If used, the *this* keyword inside the iterator will point to the object given by the argument.

Syntax

```
Iterator.find([context]);
```

Return Value

- Returns an enumerable element.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){

   ['one', 'two', 'three'].each(function(s) {

     alert("First loop : " + s);

   });


   [ 'hello', 'world'].each(function(s, index) {

     alert("Second Loop : " +  index + ': ' + s);
```

```
       });

    }



    </script>

    </head>



    <body>



      <p>Click the button to see the result.</p>

      <br />

      <br />

      <input type="button" value="Result" onclick="showResult();"/>



    </body>

    </html>
```

eachSlice() Method

This method groups the items in chunks based on a given size, with last chunk being possibly smaller.

The optional context parameter is what the iterator function will be bound to. If used, the *this* keyword inside the iterator will point to the object given by the argument.

Syntax

```
    Iterator.eachSlice([context]);
```

Return Value

- Returns array of slices (arrays).

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


var students = [

  { name: 'Sunny', age: 20 },  { name: 'Audrey', age: 21 },

  { name: 'Matt', age: 20 },   { name: 'Lodie', age: 26 },

  { name: 'Will', age: 21 },   { name: 'David', age: 23 },

  { name: 'Julien', age: 22 }, { name: 'Thomas', age: 21 },

  { name: 'Serpil', age: 22 }

];


function showResult(){


  alert ( students.eachSlice(4, function(toon) {

     return toon.pluck('name');

   }) );


   var arr = students.eachSlice(2).first();

   arr.each(function(s) {

     var h = $H(s);

     alert("Value : " + h.inspect());
```

```
      });
   }


   </script>
   </head>


   <body>


     <p>Click the button to see the result.</p>
     <br />
     <br />
     <input type="button" value="Result" onclick="showResult();"/>


   </body>
   </html>
```

entries() Method

This method returns an array representation of the enumeration. This is an alias for the more generic toArray() method.

Syntax

```
   Iterator.entries();
```

Return Value

- Returns an array representation of the enumeration.

Example

```
   <html>
   <head>
```

```
<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){


   alert ( $R(1, 5).entries() );


}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

find() Method

This method finds the first element for which the iterator returns true. This method is aliased as detect() method.

Syntax

```
Iterator.find([context]);
```

Return Value

- Returns the first element for which the iterator returns true otherwise undefined.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>



<script>



function isPrime(n) {

  if (2 > n) return false;

  if (0 == n % 2) return (2 == n);

  for (var index = 3; n / index > index; index += 2)

    if (0 == n % index) return false;

  return true;

} // isPrime
```

```
function showResult(){
  alert($R(10,15).find(isPrime) );
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

findAll() Method

This method returns all the elements for which the iterator returned true. This is an alias of select() method.

The optional context parameter is what the iterator function will be bound to. If used, the *this* keyword inside the iterator will point to the object given by the argument.

Syntax

```
Iterator.findAll();
```

Return Value

- Returns all the elements for which the iterator returned true.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){


   alert($R(1, 10).findAll(function(n) { return 0 == n % 2; }));


   alert( [ 'hello', 'world', 'nice'].findAll(function(s) {

          return s.length >= 5;

   }));

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />
```

```
    <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

grep() Method

This method returns all the elements that match the filter. If an iterator is provided, it is used to produce the returned value for each selected element.

The optional iterator parameter will transform the result set in a manner similar to the map() method.

The optional context parameter is what the iterator function will be bound to. If used, the *this* keyword inside the iterator will point to the object given by the argument.

Syntax

```
Iterator.findAll();
```

Return Value

- Returns all the elements for which the iterator returned true.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){
   // Get all strings with a repeated letter somewhere
   alert(['hello', 'world', \
                 'is', 'cool'].grep(/(.)\1/).inspect());
   // Returns ['hello', 'cool']

   // Get all numbers ending with 0 or 5
   alert($R(1,30).grep(/[05]$/).inspect() );
   // Returns [5, 10, 15, 20, 25, 30]
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

inGroupsOf() Method

This method groups the items in fixed-size chunks, using a specific value to fill up the last chunk if necessary.

Syntax

```
Iterator.inGroupsOf(size[, filler = null]);
```

Return Value

- Returns all the elements for which the iterator returned true.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){


  var students = [

  { name: 'Sunny', age: 20 },  { name: 'Audrey', age: 21 },

  { name: 'Matt', age: 20 },   { name: 'Lodie', age: 26 },

  { name: 'Will', age: 21 },   { name: 'David', age: 23 },

  { name: 'Julien', age: 22 }, { name: 'Thomas', age: 21 },

  { name: 'Serpil', age: 22 }

  ];


  alert ( students.pluck('name').inGroupsOf(4)  ) ;
```

```
    // Returns  [ ['Sunny', 'Audrey', 'Matt', 'Lodie'],
    //       ['Will', 'David', 'Julien', 'Thomas'],
    //       ['Serpil', null, null, null] ]
  }


  </script>
  </head>


  <body>


    <p>Click the button to see the result.</p>
    <br />
    <br />
    <input type="button" value="Result" onclick="showResult();"/>


  </body>
  </html>
```

include() Method

This method determines whether a given object is in the Enumerable or not, based on the == comparison operator. This is an alias of the member() method.

If you need to check whether there is an element matching a given predicate, use any() method instead.

Syntax

```
Iterator.include(object);
```

Return Value

- Returns true if finds the given object, otherwise false.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){


  alert("$R(1,15).include(10) : " + $R(1,15).include(10) );

  // Returns true


  alert("['hello', 'world'] : " + $R(1,15).include('HELLO') );

  // Returns false


}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />
```

```
    <br />
    <input type="button" value="Result" onclick="showResult();"/>


  </body>
  </html>
```

inject() Method

This method incrementally builds a result value based on the successive results of the iterator. This can be used for the array construction, numerical sums/averages, etc.

The optional context parameter is what the iterator function will be bound to. If used, the *this* keyword inside the iterator will point to the object given by the argument.

Syntax

```
Iterator.inject(accumulator, context);
```

Return Value

- Returns accumulated value.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>


<script>


function showResult(){
  alert("Test1: " + $R(1,10).inject(0, function(acc, n)
                                     {
                                       return acc + n;
                                     }) );
  // Returns 55 (sum of 1 to 10)

  alert("Test2: " + $R(2,5).inject(1, function(acc, n)
                                    {
                                      return acc * n;
                                    }) );
  // Returns 120 (factorial 5)

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

invoke() Method

This method is an optimization for a common use-case of each() or collect() method. Invoking the same method, with the same potential arguments, for all the elements.

Syntax

```
Iterator.invoke(methodName[, arg...]);
```

Return Value

- Returns the results of the method calls.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){


  alert(['hello', 'world', 'cool!'].invoke('toUpperCase') );

  // Returns ['HELLO', 'WORLD', 'COOL!'];


  alert(['hello', 'world', 'cool!'].invoke('substring', 0, 3));

  // Returns ['hel', 'wor', 'coo']
```

```
    }


    </script>
    </head>


    <body>


      <p>Click the button to see the result.</p>
      <br />
      <br />
      <input type="button" value="Result" onclick="showResult();"/>


    </body>
    </html>
```

map() Method

This method returns the results of applying the iterator to each element. This method is aliased as collect() method.

Syntax

```
Iterator.map([context]);
```

Return Value

- Returns the results of applying the iterator to each element.

Example

```
<html>
<head>
<title>Prototype examples</title>
```

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

<script>

function showResult(){
   alert( $R(1,5).collect(function(n) {return n * n;}) );
   // returns [1, 4, 9, 16, 25]
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

max() Method

This method returns the maximum element (or element-based computation), or undefined if the enumeration is empty. Elements are either compared directly, or by first applying the iterator and comparing the returned values.

The optional context parameter is what the iterator function will be bound to. If used, the *this* keyword inside the iterator will point to the object given by the argument.

Syntax

```
Iterator.max(context);
```

Return Value

- Returns the maximum value.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){


  alert("$R(1,10).max() : " +  $R(1,10).max() );

  // Returns 10


  function Person(name, age) {

     this.name = name;

     this.age = age;

  }
```

```
      var john = new Person('John', 20);

      var mark = new Person('Mark', 35);

      var daisy = new Person('Daisy', 22);


      alert("Max Age:"+[john, mark, daisy].max(function(person) {

                        return person.age ;

                       }) );

    }


    </script>

    </head>


    <body>


      <p>Click the button to see the result.</p>

      <br />

      <br />

      <input type="button" value="Result" onclick="showResult();"/>


    </body>

    </html>
```

member() Method

This method determines whether a given object is in the Enumerable or not, based on the == comparison operator. This is convenience alias for include().

Syntax

```
Iterator.member(object);
```

Return Value

- Returns true value if it finds the given object, otherwise false.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){


  alert("$R(1,15).member(10) : " +  $R(1,15).member(10) );

  // Returns true.

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>
```

```
</body>

</html>
```

min() Method

This method returns the minimum element (or element-based computation), or undefined if the enumeration is empty. Elements are either compared directly, or by first applying the iterator and comparing the returned values.

The optional context parameter is what the iterator function will be bound to. If used, the *this* keyword inside the iterator will point to the object given by the argument.

Syntax

```
Iterator.min([context]);
```

Return Value

- Returns minimum value found.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>



<script>



function showResult(){



  alert("$R(1,10).min() : " +  $R(1,10).min() );
```

```
    // Returns 1.

    function Person(name, age) {
       this.name = name;
       this.age = age;
    }

    var john = new Person('John', 20);
    var mark = new Person('Mark', 35);
    var daisy = new Person('Daisy', 22);

    alert("Min Age:"+[john, mark, daisy].min(function(person) {
                      return person.age ;
                     }) );

  }

  </script>
  </head>

  <body>

    <p>Click the button to see the result.</p>
    <br />
    <br />
    <input type="button" value="Result" onclick="showResult();"/>
```

```
</body>

</html>
```

partition() Method

This method partitions the elements in two groups:

- Those regarded as true.
- Those considered false.

By default, regular JavaScript boolean equivalence is used, but an iterator can be provided, that computes a boolean representation of the elements.

This is a preferred solution to using both findAll/select and reject: it only loops through the elements once!

The optional context parameter is what the iterator function will be bound to. If used, the *this* keyword inside the iterator will point to the object given by the argument.

Syntax

```
Iterator.partition([context]);
```

Return Value

- Returns an array in the form [[Array of the true values], [Array of the false values]].

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>



<script>

```

```
function showResult(){

  var arr = ['hello', null, 42, false, true, , 17].partition();

  alert("Test 1 : " + arr.inspect() );
  // Returns [['hello', 42, true, 17], [null, false, undefined]]

  var arr = $R(1, 10).partition(function(n) {
  return 0 == n % 2;
  })

  alert("Test 2 : " + arr.inspect() );
  // Returns   [[2, 4, 6, 8, 10], [1, 3, 5, 7, 9]]
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

pluck() Method

This method is an optimization for a common use-case of collect() method, fetching the same property for all the elements.

Syntax

```
Iterator.pluck(propertyName);
```

Return Value

- Returns the array of property values.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){


  var arr = ['world', 'this', 'is', 'nice'].pluck('length');


  alert("Test 1 : " + arr.inspect() );

  // Returns [5, 4, 2, 4]


}
```

```
</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

reject() Method

This method returns all the elements for which the iterator returned false.

The optional context parameter is what the iterator function will be bound to. If used, the *this* keyword inside the iterator will point to the object given by the argument.

Syntax

```
Iterator.reject([context]);
```

Return Value

- Returns the array of the returned values.

Example

```
<html>
<head>
<title>Prototype examples</title>
```

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

<script>

function showResult(){

  var arr = $R(1, 10).reject(function(n) { return 0 == n % 2; });

  alert("Test 1 : " + arr.inspect() );
  // Returns [1, 3, 5, 7, 9]

  var arr = [ 'world', 'this', 'is', 'nice'].reject(function(s) {
     return s.length >= 5;
  })
  alert("Test 2 : " + arr.inspect() );
  // Returns ['this', 'is', 'nice']
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
```

```
      <br />

      <input type="button" value="Result" onclick="showResult();"/>



    </body>

    </html>
```

select () Method

This method returns all the elements for which the iterator returned true. This is an alias of findAll().

The optional context parameter is what the iterator function will be bound to. If used, the *this* keyword inside the iterator will point to the object given by the argument.

Syntax

```
Iterator.select([context]);
```

Return Value

- Returns all the elements for which the iterator returned true.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>



<script>



function showResult(){
    var arr = $R(1, 10).select(function(n) { return 0 == n % 2; });


    alert("Test 1 : " + arr.inspect() );

    // Returns [2, 4, 6, 8, 10]


    var arr = [ 'world', 'this', 'is', 'nice'].select(function(s) {

       return s.length >= 5;

    })

    alert("Test 2 : " + arr.inspect() );

    // Returns ['world']

  }


  </script>

  </head>


  <body>


    <p>Click the button to see the result.</p>

    <br />

    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

size() Method

This method returns the size of the enumeration.

Syntax

```
Iterator.size();
```

Return Value

- Returns the size of the enumeration.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>

<script>

function showResult(){

  var arr = $R(1, 10).select(function(n) { return 0 == n % 2; });

  alert("Test 1 : " + $R(1, 10).size() );
  // Returns 10

  alert("Test 2 : " + ['hello', 42, true].size() );
  // Returns 3
```

```
    alert("Test 3 : " + $H().size() );
    // Returns 0



  }


  </script>
  </head>


  <body>


    <p>Click the button to see the result.</p>
    <br />
    <br />
    <input type="button" value="Result" onclick="showResult();"/>


  </body>
  </html>
```

sortBy() Method

This method provides a custom-sorted view of the elements based on the criteria computed, for each element, by the iterator.

The optional context parameter is what the iterator function will be bound to. If used, the *this* keyword inside the iterator will point to the object given by the argument.

Syntax

```
  Iterator.sortBy([context]);
```

Return Value

- Returns an array of the sorted values.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){


  var arr = ['hello', 'world', 'this', 'is', 'nice'];

  var sorted = arr.sortBy(function(s) { return s.length; })


  alert("Test 1 : " + sorted.inspect() );

  // is', 'this', 'nice', 'world','hello']

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>
```

```
      <br />

      <br />

      <input type="button" value="Result" onclick="showResult();"/>



    </body>

    </html>
```

toArray () Method

This method returns an array representation of the enumeration. This is an alias of entries() method.

Syntax

```
Iterator.toArray();
```

Return Value

- Returns an Array representation of the enumeration.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>



<script>



function showResult(){

  alert( $R(1, 5).toArray().inspect() );

  // [1, 2, 3, 4, 5]

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

zip() Method

This method zips together 2 + sequences, providing an array of tuples.

Each tuple contains one value per original sequence. Tuples can be converted to something else by applying the optional iterator on them.

Syntax

```
Iterator.zip(Sequence);
```

Return Value

- Returns an array of zipped values.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>


<script>


function showResult(){

  var firstNames = ['Justin', 'Tobie', 'Christophe'];

  var lastNames = ['Palmer', 'Langel', 'Porteneuve'];

  var zipped = firstNames.zip(lastNames);


  alert("zipped values: " + zipped.inspect());


  var zipped = firstNames.zip(lastNames, function(a)

                            {

                     return a.join(' ');

                   })

  alert("zipped values: " + zipped.inspect());

}


</script>

</head>


<body>
```

```
    <p>Click the button to see the result.</p>

    <br />

    <br />

    <input type="button" value="Result" onclick="showResult();"/>

  </body>

  </html>
```

12. Prototype – Event Handling

Event management is one of the biggest challenges to achieve cross-browser scripting. Every browser has different approaches to handle key strokes.

Prototype Framework handles all cross browser compatibility issues and keeps you free from all trouble related to event management.

Prototype Framework provides *Event* namespace, which is replete with methods, that all take the current event object as an argument, and happily produce the information you're requesting, across all major browsers.

Event namespace also provides a standardized list of key codes you can use with keyboard-related events. The following constants are defined in the namespace:

Key Constant	Description
KEY_BACKSPACE	Represent back space key.
KEY_TAB	Represent tab key.
KEY_RETURN	Represent return key.
KEY_ESC	Represent esc key.
KEY_LEFT	Represent left key.
KEY_UP	Represent up key.
KEY_RIGHT	Represent right key.
KEY_DOWN	Represent down key.
KEY_DELETE	Represent delete key.
KEY_HOME	Represent home key.
KEY_END	Represent end key.

KEY_PAGEUP	Represent page up key.
KEY_PAGEDOWN	Represent page down key.

How to Handle Events

Before we start, let us see an example of using an event method. This example shows how to capture the DOM element on which the event occurred.

```
<html>
<head>
<title>Prototype examples</title>
   <script type="text/javascript"
   src="/javascript/prototype.js">
   </script>
<script>

// Register event 'click' and associated call back.
Event.observe(document, 'click', respondToClick);

// Callback function to handle the event.
function respondToClick(event) {
  var element = event.element();
  alert("Tag Name : " + element.tagName );
}

</script>
</head>

<body>
```

```
    <p id="note"> Click on any part to see the result.</p>

    <p id="para">This is paragraph</p>

    <div id="division">
        This is divsion.
    </div>

</body>
</html>
```

Here is a complete list of all the methods related to **Event**. The functions you're most likely to use a lot are *observe*, *element* and *stop*.

Prototype Event Methods

NOTE: Make sure you at least have the version 1.6 of prototype.js.

Methods	Description
element()	Returns the DOM element on which the event occurred.
extend()	Extends the *event* with all of the methods contained in Event.Methods.
findElement()	Returns the first DOM element with a given tag name, upwards from the one on which the event occurred.
isLeftClick()	Determines whether a button-related mouse event was about the "left" (primary, actually) button.
observe()	Registers an event handler on a DOM element.
pointerX()	Returns the absolute horizontal position for a mouse event.

pointerY()	Returns the absolute vertical position for a mouse event.
stop()	Stops the event's propagation and prevents its default action from being triggered eventually.
stopObserving()	Unregisters an event handler.
unloadCache()	Unregisters all event handlers registered through observe. Automatically wired for you. Not available since 1.6.

element() Method

This method returns the DOM element on which the event occurred.

Syntax

```
Event.element();
```

Return Value

- Returns the DOM element on which the event occurred.

Example

Here's a simple code that lets you click everywhere on the page and, if you click directly on paragraphs, hides them.

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>


<script>
```

```
// Register event 'click' and associated call back.
Event.observe(document, 'click', respondToClick);


// Callback function to handle the event.
function respondToClick(event) {
  var element = event.element();
  alert("Tag Name : " + element.tagName );
  if ('P' == element.tagName){
    element.hide();
  }
}


</script>
</head>


<body>


   <p id="note"> Click on any part to see the result.</p>


   <p id="para">This is paragraph</p>


   <div id="division">
       This is divsion.
   </div>


</body>
</html>
```

extend() Method

This method extends event with all of the methods contained in *Event.Methods*.

Note that all events inside handlers that were registered using *Event.observe* or *Element#observe* will be extended automatically.

Syntax

```
Event.extend(event);
```

Return Value

- None

Example

Not available at this moment.

findElement() Method

This method returns the first DOM element with a given tag name, upwards from the one on which the event occurred.

Sometimes, you're not interested in the actual element that got hit by the event. Sometimes you're interested in its "closest element". This is what findElement is for.

The provided tag name will be compared in a case-insensitive manner.

Syntax

```
Event.findElement(event, tagName);
```

Return Value

- Returns the first DOM element with a given tag name. If no matching element is found, the document itself (HTMLDocument node) is returned.

Example

Here's a simple code that lets you click everywhere on the page and hides the closest-fitting paragraph around your click (if any).

```
<html>

<html>
```

```
    <head>
    <title>Prototype examples</title>
       <script type="text/javascript"
       src="/javascript/prototype.js">
       </script>

    <script>

    // Register event 'click' and associated call back.
    Event.observe(document, 'click', respondToClick);

    // Callback function to handle the event.
    function respondToClick(event) {
      var element = Event.findElement(event, 'P');
      alert("Hiding Tag : " + element.tagName );
      if ( element != document ){
        element.hide();
      }
    }

    </script>
    </head>

    <body>

       <p id="note"> Click anywhere to see the result.</p>
      <p id="para1">This is paragraph 1</p>

      <br />

      <br />

      <p id="para2">This is paragraph 2</p>

      <div id="division">

          This is divsion.

      </div>


   </body>

   </html>
```

isLeftClick() Method

This method determines whether a button-related mouse event was about the "left" (primary, actually) button.

Syntax

```
Event.isLeftClick(event);
```

Return Value

- Returns true if left button is pressed.

Example

NOTE: Looks like it does not work in MSIE.

```
   <html>

   <head>

   <title>Prototype examples</title>

      <script type="text/javascript"

      src="/javascript/prototype.js">

      </script>
```

```
<script>

// Register event 'click' and associated call back.
Event.observe(document, 'click', respondToClick);

// Callback function to handle the event.
function respondToClick(event) {
  var bool = Event.isLeftClick(event);
  if ( bool ){
    alert("Left button is pressed...." );
  }
}

</script>
</head>

<body>

   <p id="note"> Click anywhere to see the result.</p>
   <p>This example may not work in IE.</p>

   <p id="para1">This is paragraph 1</p>
   <p id="para2">This is paragraph 2</p>

   <div id="division">
       This is divsion.
   </div>


</body>

</html>
```

observe() Method

This method registers an event handler on a DOM element.

To register a function as an event handler, the DOM element that you want to observe must already exist in the DOM.

Syntax

```
Event.observe(element,eventName,handler[,useCapture=false]);
```

Here are the explanations about the passed parameters:

- **element:** The DOM element you want to observe; as always in Prototype, this can be either an actual DOM reference, or the ID string for the element.
- **evenetName:** The standardized event name, as per the DOM level supported by your browser. This includes click, mousedown, mouseup, mouseover, mousemove and mouseout.
- **handler:** This is the event handler function. This can be an anonymous function you create on-the-fly.
- **useCapture:** Optionally, you can request *capturing* instead of *bubbling*. The details are in the http://www.w3.org/TR/DOM-Level-2-Events/events.html.

Return Value :

- NA.

Example

Here is an example, which observe *click* event and take an action whenever a click event occurs.

```
<html>

<head>

<title>Prototype examples</title>
```

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

<script>

// Register event 'click' and associated call back.
Event.observe(document, 'click', respondToClick);

// Callback function to handle the event.
function respondToClick(event) {
  alert("You pressed the button...." );
}

</script>
</head>

<body>

   <p id="note"> Click anywhere to see the result.</p>

   <p id="para1">This is paragraph 1</p>
   <p id="para2">This is paragraph 2</p>

   <div id="division">
       This is divsion.
   </div>
```

```
</body>

</html>
```

pointerX() Method

This method returns the absolute horizontal position for a mouse event.

Syntax

```
Event.pointerX(event);
```

Return Value

- Returns the absolute horizontal position for a mouse event.

Example

Try this example by clicking at different places.

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>


<script>


// Register event 'click' and associated call back.

Event.observe(document, 'click', respondToClick);


// Callback function to handle the event.
```

```
function respondToClick(event) {

  var x_axis = Event.pointerX(event);

  alert("Horizontal position : "  + x_axis);

}


</script>

</head>


<body>


   <p id="note">Click anywhere to see the result.</p>


   <p id="para1">This is paragraph 1</p>

   <p id="para2">This is paragraph 2</p>


   <div id="division">

       This is divsion.

   </div>


</body>

</html>
```

pointerY() Method

This method returns the absolute vertical position for a mouse event.

Syntax

```
Event.pointerY(event);
```

Return Value

- Returns the absolute vertical position for a mouse event.

Example

Try this example by clicking at different places.

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>


<script>


// Register event 'click' and associated call back.

Event.observe(document, 'click', respondToClick);


// Callback function to handle the event.

function respondToClick(event) {

  var x_axis = Event.pointerY(event);

  alert("Vertical position : "  + x_axis);

}


</script>

</head>


<body>
```

```
    <p id="note">Click anywhere to see the result.</p>


    <p id="para1">This is paragraph 1</p>

    <p id="para2">This is paragraph 2</p>


    <div id="division">

        This is divsion.

    </div>


 </body>

 </html>
```

stop() Method

This method stops the event's propagation and prevents its default action from being triggered eventually.

There are two aspects to how your browser handles an event once it fires up:

- The browser usually triggers event handlers on the actual element the event occurred on, then on its parent element, and so on and so forth, until the document.s root element is reached. This is called event bubbling, and is the most common form of event propagation. You may very well want to stop this propagation when you just handled an event, and don’t want it to keep bubbling up (or see no need for it).
- Once your code gets a chance to process the event, the browser handles it as well, if that event has a default behavior. For instance, clicking on links navigates to them; submitting forms sends them over to the server side; hitting the Return key in a single-line form field submits it, etc. You may very well want to prevent this default behavior if you do your own handling.

Because stopping one of those aspects means, in 99.9% of the cases, preventing the other one as well, Prototype bundles both in this stop function. Calling it on an event object stop propagation and prevents the default behavior.

Syntax

```
 Event.stop(event);
```

Return Value

- NA.

Example

Here's a code snippet that prevents a form from being sent to the server side if a certain field is empty.

```
Event.observe('signinForm', 'submit', function(event) {

  var login = $F('login').strip();

  if ('' == login) {

    Event.stop(event);

    // Display the issue one way or another

  }

});
```

stopObserving() Method

This method unregisters an event handler.

This function is called with exactly the same argument semantics as observe. It unregisters an event handler, so the handler is not called anymore for this element+event pair.

Syntax

```
Event.stopObserving(element, eventName, handler

                             [, useCapture = false])
```

Here are the explanations about the passed parameters:

- **element:** The DOM element you want to observe; as always in Prototype, this can be either an actual DOM reference, or the ID string for the element.
- **evenetName:** The standardized event name, as per the DOM level supported by your browser. This includes click, mousedown, mouseup, mouseover, mousemove and mouseout.
- **handler:** This is the event handler function. This can be an anonymous function you create on-the-fly.
- **useCapture:** Optionally, you can request *capturing* instead of *bubbling*. The details are in the http://www.w3.org/TR/DOM-Level-2-Events/events.html.

Return Value

- NA.

Example

This example shows how it reacts only once clicked and after that program stops observing.

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


// Register event 'click' and associated call back.

Event.observe(document, 'click', respondToClick);


// Callback function to handle the event.

function respondToClick(event) {

  alert("Left button is pressed...." );

  Event.stopObserving(document, 'click', respondToClick);

}

</script>

</head>


<body>


   <p id="note">Click anywhere to see the result.</p>
```

```
    <p id="para1">This is paragraph 1</p>


    <p id="para2">This is paragraph 2</p>


    <div id="division">
        This is divsion.
    </div>


</body>
</html>
```

unloadCache() Method

This method unregisters all event handlers registered through observe. Automatically wired for you. Not available since 1.6.

This method is removed in the 1.6 release. So, please remove calls to it from any script you currently use.

Syntax

```
Event.unloadCache()
```

Return Value

- NA.

13. Prototype – Form Management

Prototype provides an easy way to manage HTML forms. Prototype's Form is a namespace and a module for all things form-related, packed with form manipulation and serialization goodness.

While it holds methods dealing with forms as a whole, its sub module Form.Element deals with specific form controls.

Form.emement is a submodule of Form which deals with specific form controls on INPUT, SELECT and TEXTAREA elements.

Form.Element is also aliased *Field*. Following two forms are equivalent:

```
Form.Element.activate('myfield')


is equivalent to:


Field.activate('myfield');


Which in turn equivalent to:


$('myfield').activate()
```

Here is a complete list of all the methods related to **Form.Element**.

Prototype Event Methods

NOTE: Make sure you at least have the version 1.6 of prototype.js.

Methods	Description
activate()	Gives focus to a form control and selects its contents if it is a text input.
clear()	Clears the contents of a text input.

disable()	Disables a form control, effectively preventing its value to be changed until it is enabled again.
enable()	Enables a previously disabled form control.
focus()	Gives keyboard focus to an element.
getValue()	Returns the current value of a form control. A string is returned for most controls; only multiple select boxes return an array of values. The global shortcut for this method is $F().
present()	Returns true if a text input has contents, false otherwise.
select()	Selects the current text in a text input.
serialize()	Creates an URL-encoded string representation of a form control in the name=value format.

activate() Method

This method gives focus to a form control and selects its contents if it is a text input.

This method is just a shortcut for focusing and selecting. Therefore, these are equivalent:

```
Form.Element.focus('myelement').select();


is equivalent to:


$('myelement').activate();
```

Syntax

```
formElement.activate();
```

Return Value

- It returns an HTMLElement.

Example

Following example will activate the **age** fields:

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function showResult()

{

   $('age').activate()

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />


  <form id="example" action="#" onsubmit="return false">

  <fieldset><legend>User info</legend>

  <div><label for="username">Username:</label>

  <input name="username" id="username" value="Sai" type="text">
```

```
    </div>

    <div><label for="age">Age:</label>

    <input name="age" id="age" value="23" size="3" type="text">

    </div>

    <div><label for="hobbies">Your hobbies :</label>

    <select name="hobbies" id="hobbies" multiple="multiple">

      <option>coding</option>

      <option>swimming</option>

      <option>hiking</option>

      <option>drawing</option>

    </select></div>

    </fieldset>

    </form>


    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

clear() Method

This method clears the contents of a text input.

Syntax

```
formElement.clear();
```

Return Value

- It returns an HTML Element.

Example

This code sets up a text field in a way that it clears its contents the first time it receives focus.

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function showResult()

{

   $('username').onfocus = function() {

     // if already cleared, do nothing

     if (this._cleared) return


    // otherwise clear this

    this.clear()

     this._cleared = true

   }

}


</script>

</head>


<body onload="showResult();">
```

```
    <p>Focus on username to see the result:</p>
    <br />

    <form id="example" action="#" onsubmit="return false">
    <fieldset><legend>User info</legend>
    <div><label for="username">Username:</label>
    <input name="username" id="username" value="Sai" type="text">
    </div>
    <div><label for="age">Age:</label>
    <input name="age" id="age" value="23" size="3" type="text">
    </div>

    <div><label for="hobbies">Your hobbies :</label>
    <select name="hobbies" id="hobbies" multiple="multiple">
      <option>coding</option>
      <option>swimming</option>
      <option>hiking</option>
      <option>drawing</option>
    </select></div>
    </fieldset>
    </form>

  </body>
  </html>
```

disable() Method

This method disables a form control, effectively preventing its value to be changed until it is enabled again.

Syntax

```
formElement.disable();
```

Return Value

- It returns an HTML Element.

Example

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function showResult()

{

   var form = $('example');


   form[form.disabled ? 'enable' : 'disable']();


   form.disabled = !form.disabled;

}
```

```
</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />

  <form id="example" action="#" onsubmit="return false">
  <fieldset><legend>User info</legend>
  <div><label for="username">Username:</label>
  <input name="username" id="username" value="Sai" type="text">
  </div>
  <div><label for="age">Age:</label>
  <input name="age" id="age" value="23" size="3" type="text">
  </div>
  <div><label for="hobbies">Your hobbies :</label>
  <select name="hobbies" id="hobbies" multiple="multiple">
    <option>coding</option>
    <option>swimming</option>
    <option>hiking</option>
    <option>drawing</option>
  </select></div>
  </fieldset>
  </form>

  <br />
```

```
      <input type="button" value="Toggle" onclick="showResult();"/>


    </body>

    </html>
```

enable() Method

This method enables a previously disabled form control.

Syntax

```
    formElement.enable();
```

Return Value

- It returns an HTML Element.

Example

```
    <html>

    <head>

    <title>Prototype examples</title>

       <script type="text/javascript"

       src="/javascript/prototype.js">

       </script>

    <script>


    function showResult()

    {

       var form = $('example');


       form[form.disabled ? 'enable' : 'disable']();
```

```
     form.disabled = !form.disabled;
  }

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />

  <form id="example" action="#" onsubmit="return false">
  <fieldset><legend>User info</legend>
  <div><label for="username">Username:</label>
  <input name="username" id="username" value="Sai" type="text">
  </div>
  <div><label for="age">Age:</label>
  <input name="age" id="age" value="23" size="3" type="text">
  </div>
  <div><label for="hobbies">Your hobbies :</label>
  <select name="hobbies" id="hobbies" multiple="multiple">
    <option>coding</option>
    <option>swimming</option>
    <option>hiking</option>
    <option>drawing</option>
  </select></div>
```

```
    </fieldset>

    </form>


    <br />

    <input type="button" value="Toggle" onclick="showResult();"/>


  </body>

  </html>
```

focus() Method

This method gives keyboard focus to an element.

Syntax

```
  formElement.focus();
```

Return Value

- It returns an HTML Element.

Example

Following is the example to show how a control receives focus:

```
  <html>

  <head>

  <title>Prototype examples</title>

     <script type="text/javascript"

     src="/javascript/prototype.js">

     </script>

  <script>
```

```
function showResult()

{

   Form.Element.focus('age');

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />


  <form id="example" action="#" onsubmit="return false">

  <fieldset><legend>User info</legend>

  <div><label for="username">Username:</label>

  <input name="username" id="username" value="Sai" type="text">

  </div>

  <div><label for="age">Age:</label>

  <input name="age" id="age" value="23" size="3" type="text">

  </div>

  <div><label for="hobbies">Your hobbies :</label>

  <select name="hobbies" id="hobbies" multiple="multiple">

    <option>coding</option>

    <option>swimming</option>

    <option>hiking</option>

    <option>drawing</option>
```

```
      </select></div>

      </fieldset>

      </form>


      <br />

      <input type="button" value="Toggle" onclick="showResult();"/>


   </body>

   </html>
```

getValue() Method

This method returns the current value of a form control. A string is returned for most controls; only multiple select boxes return an array of values. The global shortcut for this method is $F().

Syntax

```
formElement.getValue();
```

Return Value

- It returns a string or array of strings.

Example

Following is the example to show how a control receives focus:

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>
```

```
<script>


function showResult()
{
   var value = $('age').getValue();
   alert("Returned value is : " + value );
}


</script>
</head>


<body>


  <p>Click the button to see the result.</p>
  <br />


  <form id="example" action="#" onsubmit="return false">
  <fieldset><legend>User info</legend>
  <div><label for="username">Username:</label>
  <input name="username" id="username" value="Sai" type="text">
  </div>
  <div><label for="age">Age:</label>
  <input name="age" id="age" value="23" size="3" type="text">
  </div>
  <div><label for="hobbies">Your hobbies :</label>
  <select name="hobbies" id="hobbies" multiple="multiple">
    <option>coding</option>
```

```
        <option>swimming</option>

        <option>hiking</option>

        <option>drawing</option>

      </select></div>

      </fieldset>

      </form>


      <br />

      <input type="button" value="Toggle" onclick="showResult();"/>


    </body>

    </html>
```

present() Method

This method returns true if a text input has contents, false otherwise.

Syntax

```
    formElement.present();
```

Return Value

- It returns true if a text input has contents, false otherwise.

Example

Try to submit the following form first without filling out the information, then again after typing some text in the fields:

```
    <html>

    <head>

    <title>Prototype examples</title>

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

```
   src="/javascript/prototype.js">
   </script>
<script>
function showResult()
{
  var valid, msg = $('msg')

  // are both fields present?
  valid = $('username').present() && $('email').present()

  if (valid) {
    // in real world we would return true here to allow the
    // form to be submitted return true
    msg.update('Passed validation!').style.color = 'green'
  } else {
    msg.update('Fill out all the fields.').style.color = 'red'
  }
  return false
}
</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />
```

```
      <form id="example" action="#">
      <fieldset>
        <legend>User Details</legend>
        <p style="color:green;" id="msg">
      Fill out all the fields:
        </p>

        <div>
          <label for="username">Username</label>
          <input id="username" name="username" type="text">
        </div>

        <div>
          <label for="email">Email Address</label>
          <input id="email" name="email" type="text">
        </div>
        <div>
           <input value="result" type="button" onclick="showResult();>
        </div>
      </fieldset>
      </form>

    </body>
    </html>
```

select() Method

This method selects the current text in a text input.

Syntax

```
formElement.select();
```

Return Value

- It returns selected HTML Element.

Example

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function showResult()

{

  var elem = $('searchbox');

  Form.Element.select(elem)

}


</script>

</head>


<body>


  <p>Enter some value click the button to see the result.</p>
```

```
      <br />


      <form id="example" action="#">

      <fieldset>

      <div>

          <label for="searchbox">Search</label>

          <input id="searchbox" name="searchbox" type="text">

      </div>


      <div <input value="Result" type="button" onclick="showResult();>

      </div>

      </fieldset>

      </form>


    </body>

    </html>
```

serialize() Method

This method creates a URL-encoded string representation of a form control in the name=value format.

Syntax

```
formElement.serialize();
```

Return Value

- It returns a string.

Example

```
<html>
```

```
<head>

<title>Prototype examples</title>


   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function showResult()

{

   var form = $('example');


   var element = form.serialize();

   alert("form.serialize() : " + element.inspect());

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />


  <form id="example" action="#" onsubmit="return false">

  <fieldset><legend>User info</legend>
```

```
    <div><label for="username">Username:</label>

    <input name="username" id="username" value="Sulien" type="text">

    </div>

    <div><label for="age">Age:</label>

    <input name="age" id="age" value="23" size="3" type="text">


    </div>

    <div><label for="hobbies">Your hobbies are:</label>

    <select name="hobbies" id="hobbies" multiple="multiple">

      <option>coding</option>


      <option>swimming</option>

      <option>hiking</option>

      <option>drawing</option>

    </select></div>


    </fieldset>

    </form>


    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

Here is a complete list of all the methods related to **Form**.

Prototype Form Methods

NOTE: Make sure you at least have the version 1.6 of prototype.js.

Methods	Description
disable()	Disables the form as whole. Form controls will be visible but uneditable.
enable()	Enables a fully or partially disabled form.
findFirstElement()	Finds first non-hidden, non-disabled form control.
focusFirstElement()	Gives keyboard focus to the first element of the form.
getElements()	Returns a collection of all form controls within a form.
getInputs()	Returns a collection of all INPUT elements in a form. Use optional type and name arguments to restrict the search on these attributes.
request()	A convenience method for serializing and submitting the form via an Ajax.Request to the URL of the form's action attribute. The options parameter is passed to the Ajax.Request instance, allowing to override the HTTP method and to specify additional parameters.
reset()	Resets a form to its default values.
serialize()	Serialize form data to a string suitable for Ajax requests (default behavior) or, if optional getHash evaluates to true, an object hash where keys are form control names and values are data.
serializeElements()	Serialize an array of form elements to a string suitable for Ajax requests (default behavior) or, if optional getHash evaluates to true, an object hash where keys are form control names and values are data.

disable() Method

This method disables the form as a whole. Form controls will be visible but uneditable. Check the example below.

Syntax

```
form.disable();
```

Return Value

- It returns HTMLFormElement.

Example

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function showResult()

{

   var form = $('example');


   form[form.disabled ? 'enable' : 'disable']();


   form.disabled = !form.disabled;

}
```

```
</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />


  <form id="example" action="#" onsubmit="return false">

  <fieldset><legend>User info</legend>

  <div><label for="username">Username:</label>

  <input name="username" id="username" value="Sulien" type="text">

  </div>

  <div><label for="age">Age:</label>

  <input name="age" id="age" value="23" size="3" type="text">

  </div>

  <div><label for="hobbies">Your hobbies are:</label>

  <select name="hobbies" id="hobbies" multiple="multiple">

    <option>coding</option>

    <option>swimming</option>

    <option>hiking</option>

    <option>drawing</option>

  </select></div>

  <div class="buttonrow">

  <button onclick="showResult();">Toggle disabled!</button>

  </div>

  </fieldset>
```

```
    </form>


</body>

</html>
```

enable() Method

This method enables a fully or partially disabled form. Enabling the form is done by iterating over form elements and enabling them.

Syntax

```
form.enable();
```

Return Value

- It returns HTMLFormElement.

Example

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function showResult()

{

   var form = $('example');


   form[form.disabled ? 'enable' : 'disable']();
   form.disabled = !form.disabled;
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />

  <form id="example" action="#" onsubmit="return false">
  <fieldset><legend>User info</legend>
  <div><label for="username">Username:</label>
  <input name="username" id="username" value="Sulien" type="text">
  </div>
  <div><label for="age">Age:</label>
  <input name="age" id="age" value="23" size="3" type="text">
  </div>
  <div><label for="hobbies">Your hobbies are:</label>
  <select name="hobbies" id="hobbies" multiple="multiple">
    <option>coding</option>
    <option>swimming</option>
    <option>hiking</option>
    <option>drawing</option>
  </select></div>
```

```
      <div class="buttonrow">

      <button onclick="showResult();">Toggle disabled!</button>

      </div>

      </fieldset>

      </form>


    </body>

    </html>
```

findFirstElement() Method

This method finds the first non-hidden, non-disabled form control.

Syntax

```
    form.findFirstElement();
```

Return Value

- It returns HTMLFormElement.

Example

```
    <html>

    <head>

    <title>Prototype examples</title>

       <script type="text/javascript"

       src="/javascript/prototype.js">

       </script>

    <script>


    function showResult()
```

```
{
   var form = $('example');

   var element = form.findFirstElement();
   alert("First non hidden element : " + element.inspect() );

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />

  <form id="example" action="#" onsubmit="return false">
  <fieldset><legend>User info</legend>
  <div><label for="username">Username:</label>
  <input name="username" id="username" value="Sulien" type="text">
  </div>
  <div><label for="age">Age:</label>
  <input name="age" id="age" value="23" size="3" type="text">
  </div>
  <div><label for="hobbies">Your hobbies are:</label>
  <select name="hobbies" id="hobbies" multiple="multiple">
    <option>coding</option>
```

```
        <option>swimming</option>

        <option>hiking</option>

        <option>drawing</option>

      </select></div>

      </fieldset>

      </form>


      <br />

      <input type="button" value="Result" onclick="showResult();"/>


    </body>

    </html>
```

focusFirstElement() Method

This method gives keyboard focus to the first element of the form.

This is useful for enhancing usability on your site by bringing focus on page load to forms such as search forms or contact forms where a user is ready to start typing right away.

Syntax

```
form.focusFirstElement();
```

Return Value

- It returns HTMLFormElement.

Example

```
<html>

<head>

<title>Prototype examples</title>

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

```
      src="/javascript/prototype.js">
      </script>
   <script>

   function showResult()
   {
      var form = $('example');

      form.focusFirstElement();


   }

   </script>
   </head>

   <body>

     <p>Click the button to see the result.</p>
     <br />

     <form id="example" action="#" onsubmit="return false">
     <fieldset><legend>User info</legend>
     <div><label for="username">Username:</label>
     <input name="username" id="username" value="Sulien" type="text">
     </div>
     <div><label for="age">Age:</label>
```

```
    <input name="age" id="age" value="23" size="3" type="text">

    </div>

    <div><label for="hobbies">Your hobbies are:</label>

    <select name="hobbies" id="hobbies" multiple="multiple">

      <option>coding</option>

      <option>swimming</option>

      <option>hiking</option>

      <option>drawing</option>

    </select></div>

    </fieldset>

    </form>


    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

getElements() Method

This method returns a collection of all form controls within a form.

NOTE: OPTION elements are not included in the result; only their parent SELECT control is.

Syntax

```
form.getElements();
```

Return Value

- It returns an array of HTMLFormElement.

Example

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function showResult()

{

   var form = $('example');


   var arr = form.getElements();

   arr.each(function(item) {

      alert("Return Element : " + item.inspect());

   });

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />
```

```
    <form id="example" action="#" onsubmit="return false">

    <fieldset><legend>User info</legend>

    <div><label for="username">Username:</label>

    <input name="username" id="username" value="Sulien" type="text">

    </div>

    <div><label for="age">Age:</label>

    <input name="age" id="age" value="23" size="3" type="text">

    </div>

    <div><label for="hobbies">Your hobbies are:</label>

    <select name="hobbies" id="hobbies" multiple="multiple">

      <option>coding</option>

      <option>swimming</option>

      <option>hiking</option>

      <option>drawing</option>

    </select></div>

    </fieldset>

    </form>


    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

getInputs() Method

This method returns a collection of all INPUT elements in a form. Use optional type and name arguments to restrict the search on these attributes.

Input elements are returned in the document order, not the tabindex order.

Syntax

```
formElement.getInputs([type [, name]]);
```

Return Value

- It returns an array of HTMLFormElement.

```
form.getInputs()       // -> all INPUT elements

form.getInputs('text') // -> only text inputs


var buttons = form.getInputs('radio', 'education')

// -> only radio buttons of name "education"
```

Example

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function showResult()

{

   var form = $('example');


   var arr = form.getInputs();

   arr.each(function(item) {

      alert("Return Element 1 : " + item.inspect());
   });
}


</script>
</head>


<body>


  <p>Click the button to see the result.</p>
  <br />


  <form id="example" action="#" onsubmit="return false">
  <fieldset><legend>User info</legend>
  <div><label for="username">Username:</label>
  <input name="username" id="username" value="Sulien" type="text">
  </div>
  <div><label for="age">Age:</label>
  <input name="age" id="age" value="23" size="3" type="text">
  </div>
  <div><label for="hobbies">Your hobbies are:</label>
  <select name="hobbies" id="hobbies" multiple="multiple">
    <option>coding</option>
    <option>swimming</option>
    <option>hiking</option>
    <option>drawing</option>
  </select></div>
  </fieldset>
```

```
    </form>


    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

request() Method

This method is a convenient way for serializing and submitting the form via an *Ajax.Request* to the URL of the form's action attribute. The options parameter is passed to the Ajax.Request instance, allowing to override the HTTP method and to specify additional parameters.

Options passed to request() are intelligently merged with the underlying Ajax.Request options:

- If the form has a method attribute, its value is used for the Ajax.Request method option. If a method option is passed to request(), it takes precedence over the form's method attribute. If neither is specified, method defaults to "POST".
- Key-value pairs specified in the parameters option (either as a hash or a query string) will be merged with (and take precedence over) the serialized form parameters.

Syntax

```
  formElement.request([options]);
```

Return Value

- It returns a new Ajax.Request.

Example 1

Consider the following example:

```
  <html>

  <head>
```

```
<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function postIt()

{

   var form = $('example');


   form.request(); //done - it's posted

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />


  <form id="example" action="#" onsubmit="return false">

  <fieldset><legend>User info</legend>

  <div><label for="username">Username:</label>

  <input name="username" id="username" value="Sulien" type="text">

  </div>

  <div><label for="age">Age:</label>
```

```
    <input name="age" id="age" value="23" size="3" type="text">

    </div>

    <div><label for="hobbies">Your hobbies are:</label>

    <select name="hobbies" id="hobbies" multiple="multiple">

      <option>coding</option>

      <option>swimming</option>

      <option>hiking</option>

      <option>drawing</option>

    </select></div>

    </fieldset>

    </form>



    <br />

    <input type="button" value="Post It" onclick="postIt();"/>



  </body>

  </html>
```

Example 2

There could be another example where you can do something in your callback function:

```
  <html>

  <head>

  <title>Prototype examples</title>

     <script type="text/javascript"

     src="/javascript/prototype.js">

     </script>

  <script>
```

```
function postIt()
{
   var form = $('example');

   form.request({
     onComplete: function(){ alert('Form data saved!') }
   })
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />

  <form id="example" action="#" onsubmit="return false">
  <fieldset><legend>User info</legend>
  <div><label for="username">Username:</label>
  <input name="username" id="username" value="Sulien" type="text">
  </div>
  <div><label for="age">Age:</label>
  <input name="age" id="age" value="23" size="3" type="text">
  </div>
  <div><label for="hobbies">Your hobbies are:</label>
```

```
    <select name="hobbies" id="hobbies" multiple="multiple">
      <option>coding</option>
      <option>swimming</option>
      <option>hiking</option>
      <option>drawing</option>
    </select></div>
    </fieldset>
    </form>

    <br />
    <input type="button" value="Post It" onclick="postIt();"/>

  </body>
  </html>
```

Example 3

Here is one more example showing you how to override the HTTP method and add some parameters, by simply using method and parameters in the options. In this example, we set the method to GET and set two fixed parameters: interests and hobbies. The latter already exists in the form but this value will take precedence.

```
  <html>
  <head>
  <title>Prototype examples</title>
     <script type="text/javascript"
     src="/javascript/prototype.js">
     </script>
  <script>

  function postIt()
```

```
{
   var form = $('example');

   form.request({
   method: 'get',
   parameters: { interests:'JavaScript',
                 'hobbies[]':['programming', 'music'] },
   onComplete: function(){ alert('Form data saved!') }
})
}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />

  <form id="example" action="#" onsubmit="return false">
  <fieldset><legend>User info</legend>
  <div><label for="username">Username:</label>
  <input name="username" id="username" value="Sulien" type="text">
  </div>
  <div><label for="age">Age:</label>
  <input name="age" id="age" value="23" size="3" type="text">
  </div>
```

```
    <div><label for="hobbies">Your hobbies are:</label>

    <select name="hobbies[]" id="hobbies" multiple="multiple">

      <option>coding</option>

      <option>swimming</option>

      <option>hiking</option>

      <option>drawing</option>

    </select></div>

    </fieldset>

    </form>


    <br />

    <input type="button" value="Post It" onclick="postIt();"/>


  </body>

  </html>
```

reset() Method

This method resets a form to its default values.

Syntax

```
formElement.reset();
```

Return Value

- It returns an HTMLFormElement.

Example

Consider the following example:

```
<html>
```

```
<head>
<title>Prototype examples</title>
   <script type="text/javascript"
   src="/javascript/prototype.js">
   </script>
<script>

function resetIt()
{
   var form = $('example');

   form.reset();
}

</script>
</head>

<body>

  <p>Type something and click the button to see the result.</p>
  <br />

  <form id="example" action="#" onsubmit="return false">
  <fieldset><legend>User info</legend>
  <div><label for="username">Username:</label>
  <input name="username" id="username" value="Sulien" type="text">
  </div>
```

```
    <div><label for="age">Age:</label>

    <input name="age" id="age" value="23" size="3" type="text">

    </div>

    <div><label for="hobbies">Your hobbies are:</label>

    <select name="hobbies" id="hobbies" multiple="multiple">

      <option>coding</option>

      <option>swimming</option>

      <option>hiking</option>

      <option>drawing</option>

    </select></div>

    </fieldset>

    </form>


    <br />

    <input type="button" value="Rest It" onclick="resetIt();"/>


  </body>

  </html>
```

serialize() Method

This method is used to serialize form data to a string suitable for Ajax requests (default behavior) or, if optional getHash evaluates to true, an object hash where keys are form control names and values are data.

Depending on whether or not the optional parameter getHash evaluates to true, the result is either an object of the form {name: "johnny", color: "blue"} or a string of the form "name=johnny&color=blue", suitable for parameters in an Ajax request.

Syntax

```
formElement.serialize([getHash = false]);
```

Return Value

- It returns a String object.

Here are two hints on how it works. For detail, look at the example below.

```
$('example').serialize()

// 'username=sulien&age=22&hobbies=coding&hobbies=hiking'

$('example').serialize(true)

// {username: 'sulien', age: '22', hobbies: ['coding', 'hiking']}
```

Example

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function showResult()

{

   var form = $('example');


   var element = form.serialize();

   alert("form.serialize() : " + element.inspect());

}


</script>

</head>
```

```
<body>


  <p>Click the button to see the result.</p>

  <br />


  <form id="example" action="#" onsubmit="return false">

  <fieldset><legend>User info</legend>

  <div><label for="username">Username:</label>

  <input name="username" id="username" value="Sulien" type="text">

  </div>

  <div><label for="age">Age:</label>

  <input name="age" id="age" value="23" size="3" type="text">

  </div>

  <div><label for="hobbies">Your hobbies are:</label>

  <select name="hobbies" id="hobbies" multiple="multiple">

    <option>coding</option>

    <option>swimming</option>

    <option>hiking</option>

    <option>drawing</option>

  </select></div>

  </fieldset>

  </form>


  <br />

  <input type="button" value="Result" onclick="showResult();"/>
```

```
</body>

</html>
```

serializeElements() Method

This method serializes an array of form elements to a string suitable for Ajax requests (default behavior) or, if optional getHash evaluates to true, an object hash where keys are form control names and values are data.

The preferred method to serialize a form is *Form.serialize*. However, with *serializeElements* you can serialize specific input elements of your choice.

Syntax

```
Form.serializeElements(elements [,getHash = false]);
```

Return Value

- It returns a String object.

Example

Consider the following example:

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function showResult()

{

   var form = $('example');

   var arr = form.getInputs('text');
```

```
      var element = Form.serializeElements( arr );

      alert("Serialized String : " + element.inspect());

   }


   </script>

   </head>


   <body>


     <p>Click the button to see the result.</p>

     <br />


     <form id="example" action="#" onsubmit="return false">

     <fieldset><legend>User info</legend>

     <div><label for="username">Username:</label>

     <input name="username" id="username" value="Sulien" type="text">

     </div>

     <div><label for="age">Age:</label>

     <input name="age" id="age" value="23" size="3" type="text">

     </div>

     <div><label for="hobbies">Your hobbies are:</label>

     <select name="hobbies" id="hobbies" multiple="multiple">

       <option>coding</option>

       <option>swimming</option>

       <option>hiking</option>

       <option>drawing</option>
```

```
    </select></div>

    </fieldset>

    </form>


    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

14. Prototype – JSON Support

Introduction to JSON

JSON (JavaScript Object Notation) is a lightweight data-interchange format.

- JSON is easy for humans to read and write.
- JSON is easy for machines to parse and generate.
- JSON is based on a subset of the JavaScript Programming Language.
- JSON is notably used by APIs all over the web and is a fast alternative to XML in Ajax requests.
- JSON is a text format that is completely language independent.

Prototype 1.5.1 and later version, features JSON encoding and parsing support.

JSON Encoding

Prototype provides the following methods for encoding:

NOTE: Make sure you at least have the version 1.6 of prototype.js.

Methods	Description
Number.toJSON()	Returns a JSON string for the given Number.
String.toJSON()	Returns a JSON string for the given String.
Array.toJSON()	Returns a JSON string for the given Array.
Hash.toJSON()	Returns a JSON string for the given Hash.
Date.toJSON()	Converts the date into a JSON string (following the ISO format used by JSON).
Object.toJSON()	Returns a JSON string for the given Object.

Number.toJSON() Method

This method returns a JSON string.

Syntax

```
number.toJSON();
```

Return Value

- JSON string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


  alert("(45).toJSON() : " + (45).toJSON() );


}


</script>

</head>


<body>
```

```
    <p>Click the button to see the result.</p>

    <br />

    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

String.toJSON() Method

This method returns a JSON string.

Syntax

```
  string.toJSON() ;
```

Return Value

- Returns a JSON string.

Example

```
  <html>

  <head>

  <title>Prototype examples</title>

  <script type="text/javascript"

     src="/javascript/prototype.js">

  </script>

  <script>


  function showResult(){
     var str = 'The "Quoted" chronicles';


     alert("JSON String : " + str.toJSON() );
  }


  </script>
  </head>


  <body>


    <p>Click the button to see the result.</p>
    <br />
    <br />
    <input type="button" value="Result" onclick="showResult();"/>


  </body>
  </html>
```

Array.toJSON() Method

This method returns a JSON string.

Syntax

```
    array.toJSON()  ;
```

Return Value

- Returns a JSON string.

Example

```
<html>

<head>

<title>Prototype examples</title>

<script type="text/javascript"

   src="/javascript/prototype.js">

</script>

<script>


function showResult(){


   var arr = ['a', {b: null}];

   alert("JSON String : "  + arr.toJSON() );

}


</script>

</head>


<body>


  <p>Click the button to see the result.</p>

  <br />

  <br />

  <input type="button" value="Result" onclick="showResult();"/>


</body>

</html>
```

Hash.toJSON() Method

This method returns a JSON string.

Syntax

```
hash.toJSON() ;
```

Return Value

- Returns a JSON string.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>
<script>

function showResult(){

 var h = new Hash({ a: 'apple', b: 'banana', c: 'coconut' });

 alert( "h.toJSON() : " + h.toJSON() );

}

</script>
</head>
```

```
<body>

  <p>Click the button to see the result.</p>
  <br />
  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

Date.toJSON() Method

This method converts the date into a JSON string (following the ISO format used by JSON).

Syntax

```
Date.toJSON();
```

Return Value

- Returns a string for the given Date.

Example

```
<html>
<head>
<title>Prototype examples</title>
   <script type="text/javascript"
   src="/javascript/prototype.js">
   </script>
<script>
```

```
function showResult()
{

   var str = new Date(1969, 11, 31, 19).toJSON();
   alert ("Returns JSON String: " + str );

}

</script>
</head>

<body>

  <p>Click the button to see the result.</p>
  <br />

  <br />
  <input type="button" value="Result" onclick="showResult();"/>

</body>
</html>
```

Object.toJSON() Method

This method returns a JSON string.

Syntax

```
Object.toJSON(obj);
```

Return Value

- Returns a JSON string.

Example

```
<html>
<head>
<title>Prototype examples</title>
<script type="text/javascript"
   src="/javascript/prototype.js">
</script>

<script>

function showResult(){

   var o = {name: 'Prototype', Version: 1.6};

   alert("Object.toJSON(o): " + Object.toJSON(o));
}

</script>
</head>

<body>
```

```


    <p>Click the button to see the result.</p>

    <br />

    <br />

    <input type="button" value="Result" onclick="showResult();"/>


  </body>

  </html>
```

If you are unsure of the type of data you need to encode, your best bet is to use Object.toJSON so:

```
  var data = {name: 'Violet', occupation: 'character', age: 25 };

  Object.toJSON(data);
```

This will produce the following result:

```
  '{"name": "Violet", "occupation": "character", "age": 25}'
```

Furthermore, if you are using custom objects, you can set your own toJSON method, which will be used by *Object.toJSON*. For example:

```
  var Person = Class.create();

  Person.prototype = {

    initialize: function(name, age) {

      this.name = name;

      this.age = age;

    },

    toJSON: function() {

      return ('My name is ' + this.name +

        ' and I am ' + this.age + ' years old.').toJSON();

    }
```

```
};

var john = new Person('John', 49);

Object.toJSON(john);
```

This will produce the following result:

```
'"My name is John and I am 49 years old."'
```

Parsing JSON

In JavaScript, parsing JSON is typically done by evaluating the content of a JSON string. Prototype introduces String.evalJSON to deal with this. For example:

```
var d='{ "name":"Violet","occupation":"character" }'.evalJSON();

d.name;
```

This will produce the following result:

```
"Violet"
```

Using JSON with Ajax

Using JSON with Ajax is very straightforward. Simply invoke String.evalJSON on the transport's responseText property:

```
new Ajax.Request('/some_url', {

  method:'get',

  onSuccess: function(transport){

     var json = transport.responseText.evalJSON();

   }

});
```

If your data comes from an untrusted source, be sure to sanitize it:

```
new Ajax.Request('/some_url', {

  method:'get',
  requestHeaders: {Accept: 'application/json'},

  onSuccess: function(transport){

    var json = transport.responseText.evalJSON(true);

  }

});
```

15. Prototype – AJAX Support

Introduction to AJAX

AJAX stands for **A**synchronous **Ja**vaScript and **X**ML. AJAX is a new technique for creating better, faster, and more interactive web applications with the help of XML, HTML, CSS and Java Script.

For a complete understanding on AJAX, please go through our simple AJAX Tutorial.

Prototype Support for AJAX

Prototype framework enables you to deal with Ajax calls in a very easy and fun way that is also safe (cross-browser). Prototype also deals in a smart way with JavaScript code returned from a server and provides helper classes for polling.

Ajax functionality is contained in the global *Ajax object*. This object provides all the necessary methods to handle AJAX requests and responses in an easy way.

AJAX Request

Actual requests are made by creating instances of the **Ajax.Request()** object.

```
new Ajax.Request('/some_url', { method:'get' });
```

The first parameter is the URL of the request; the second is the options hash. The method option refers to the HTTP method to be used; default method is POST.

This AJAX method initiates and processes an AJAX request. This object is a general-purpose AJAX requester: it handles the life-cycle of the request, handles the boilerplate, and lets you plug in callback functions for your custom needs.

In the optional options hash, you can use any callback function like onComplete and/or onSuccess depending on your custom needs.

Syntax:

```
new Ajax.Request(url[, options]);
```

As soon as the object is created, it initiates the request, then goes on processing it throughout its life-cyle. The defined life-cycle is as follows:

1. Created
2. Initialized

3. Request sent
4. Response being received (can occur many times, as packets come in)
5. Response received, request complete

There is a set of callback functions, defined in **Ajax Options**, which are triggered in the following order:

1. *onCreate* (this is actually a callback reserved to **AJAX global responders**)
2. *onUninitialized* (maps on Created)
3. *onLoading* (maps on Initialized)
4. *onLoaded* (maps on Request sent)
5. *onInteractive* (maps on Response being received)
6. *onXYZ* (numerical response status code), onSuccess or onFailure (see below)
7. *onComplete*

Portability

Depending on how your browser implements *XMLHttpRequest*, one or more callbacks may never be invoked. In particular, *onLoaded* and *onInteractive* are not a 100% safe bet so far. However, the global *onCreate*, *onUninitialized* and the two final steps are very much guaranteed.

Return Value

- new Ajax.Request

Disabling and Enabling a PeriodicalUpdater

You can pull the brake on a running PeriodicalUpdater by simply calling its stop method. If you wish to re-enable it later, just call its start method. Both take no argument.

Example

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>



function SubmitRequest()
```

```
{



  new Ajax.Request('/cgi-bin/ajax.cgi', {

  method: 'get',

  onSuccess: successFunc,

  onFailure:  failureFunc

  });


}


function successFunc(response){


     if (200 == response.status){

        alert("Call is success");

    }

    var container = $('notice');

    var content = response.responseText;

    container.update(content);

}


function failureFunc(response){


     alert("Call is failed" );


}

</script>
```

```
</head>


<body>


  <p>Click submit button see how current notice changes.</p>

  <br />


  <div id="notice">Current Notice</div>

  <br />

  <br />

  <input type="button" value="Submit" onclick="SubmitRequest();"/>


</body>

</html>
```

Here is the content of **ajax.cgi**.

```
#!/usr/bin/perl


print "Content-type: text/html\n\n";


print "This content is returned by AJAX cgi <br />";

print "Current Time " . localtime;
```

Parameters and the HTTP Method

You can pass the parameters for the request as the parameters property in options:

```
new Ajax.Request('/some_url', {

  method: 'get',
  parameters: {company: 'example', limit: 12}
});
```

AJAX Response Callbacks

Ajax requests are by default asynchronous, which means you must have callbacks that will handle the data from a response. Callback methods are passed in the options hash when making a request:

```
new Ajax.Request('/some_url',
{
 method:'get',
 onSuccess: function(transport){
   var response = transport.responseText || "no response text";
   alert("Success! \n\n" + response);
   },
 onFailure: function(){ alert('Something went wrong...') }
});
```

Here, two callbacks are passed in the hash:

- onSuccess
- onFailure

Any of the above two call is called accordingly based on the status of the response. The first parameter passed to both is the native *xmlHttpRequest* object from which you can use its *responseText* and *responseXML* properties, respectively.

You can specify both callbacks, one or none - it's up to you. Other available callbacks are:

- onUninitialized
- onLoading
- onLoaded
- onInteractive
- onComplete
- onException

They all match a certain state of the *xmlHttpRequest* transport, except for onException, which fires when there was an exception while dispatching other callbacks.

NOTE: The onUninitialized, onLoading, onLoaded, and onInteractive callbacks are not implemented consistently by all browsers. In general, it's best to avoid using these.

Prototype AJAX Methods

Ajax object provides all the necessary methods to handle AJAX requests and responses in an easy way. Here is a complete list of all the methods related to AJAX.

NOTE: Make sure you at least have the version 1.6 of prototype.js.

Methods	Description
Ajax Options	This is not a method but details all core options shared by all AJAX requesters and callbacks.
Ajax.PeriodicalUpdater()	Periodically performs an AJAX request and updates a container's contents based on the response text.
Ajax.Request()	Initiates and processes an AJAX request.
Ajax.Responders()	A repository of global listeners notified about every step of Prototype-based AJAX requests.
Ajax.Response()	The object passed as the first argument of all Ajax requests callbacks.
Ajax.Updater()	Performs an AJAX request and updates a container's contents based on the response text.

AJAX Options

This is *Ajax* namespace which share a common set of *options* and callbacks.

Callbacks are called at various points in the life-cycle of a request, and always feature the same list of arguments. They are passed to requesters right along with their other options.

Common options

- As an **object**, with properties representing headers.

- As an **array**, with even-index (0, 2...) elements being header names, and odd-index (1, 3...) elements being values.

Prototype automatically provides a set of default headers, that this option can override and augment:

- ***X-Requested-With*** is set to 'XMLHttpRequest'.
- ***X-Prototype-Version*** provides Prototype's current version (e.g. 1.5.0).
- ***Accept*** defaults to 'text/javascript, text/html, application/xml, text/xml, */*'
- ***Content-type*** is built based on the contentType and encoding options.

Option	Description
asynchronous	**Default value is *true*.** Determines whether XMLHttpRequest is used asynchronously or not.
contentType	**Default value is *application/x-www-form-urlencoded*.** The Content-Type header for your request.
encoding	**Default value is *UTF-8*.** The Content-Type header for your request.
method	**Default value is *post*.** The HTTP method to use for the request.
parameters	**Default value is *' '*.** The parameters for the request, which will be encoded into the URL for a 'get' method, or into the request body for the other methods. This can be provided either as a URL-encoded string or as any Hash-compatible object with properties representing parameters.
postBody	**Default value is *None*.** Specific contents for the request body on a 'post' method. If it is not provided, the contents of the parameters option will be used instead.
requestHeaders	**See description below.** Request headers can be passed under two forms: • As an **object**, with properties representing headers.

	• As an **array**, with even-index (0, 2...) elements being header names, and odd-index (1, 3...) elements being values. Prototype automatically provides a set of default headers, that this option can override and augment: • **X-Requested-With** is set to 'XMLHttpRequest'. • **X-Prototype-Version** provides Prototype's current version (e.g. 1.5.0). • **Accept** defaults to 'text/javascript, text/html, application/xml, text/xml, */*' • **Content-type** is built based on the contentType and encoding options.
evalJS	**Default value is *true*.** Automatically evals the content of *Ajax.Response#responseText* if the content-type returned by the server is one of the following: application/ecmascript, application/javascript, application/x-ecmascript, application/x-javascript, text/ecmascript, text/javascript, text/x-ecmascript, or text/x-javascript and the request obeys SOP, (Simple Origin Policy). If you need to force evalutation, pass 'force'. To prevent it altogether, pass false.
evalJSON	**Default value is *true*.** Automatically evals the content of *Ajax.Response#responseText* and populates *Ajax.Response#responseJSON* with it if the content-type returned by the server is set to *application/json*. If the request doesn't obey SOP, the content is sanitized before evaluation. If you need to force evalutation, pass 'force'. To prevent it altogether, pass false.
sanitizeJSON	***false* for local requests, *true* otherwise..** Sanitizes the content of *Ajax.Response#responseText* before evaluating it.

Common Callbacks:

Callback	Description
onCreate	Triggered when the Ajax.Request object is initialized. This is after the parameters and the URL have been processed, but before first using the methods of the XHR object.
onComplete	Triggered at the very end of a request's life-cycle, once the request completed, status-specific callbacks were called, and possible automatic behaviors were processed.
onException	Triggered whenever an XHR error arises. Has a custom signature: the first argument is the requester (i.e., an Ajax.Request instance), the second is the exception object.
onFailure	Invoked when a request completes and its status code exists but is not in the 2xy family. This is skipped if a code-specific callback is defined, and happens before onComplete.
onInteractive	*Not guaranteed* but triggered whenever the requester receives a part of the response (but not the final part), should it be sent in several packets.
onLoaded	*Not guaranteed* but triggered once the underlying XHR object is setup, the connection open, and ready to send its actual request.
onLoading	*Not guaranteed* but triggered when the underlying XHR object is being setup, and its connection opened.
onSuccess	Invoked when a request completes and its status code is undefined or belongs in the 2xy family. This is skipped if a code-specific callback is defined, and happens before *onComplete*
onUninitialized	*Not guaranteed* but invoked when the XHR object was just created.
onXYZ	With XYZ being an HTTP status code for the response. Invoked when the response just completed, and the status code is exactly the one we used in t he callback name. Prevents execution of onSuccess / onFailure. Happens before onComplete.

Responder Callbacks

Callback	Description
onCreate	Triggered whenever a requester object from the Ajax namespace is created, after its parameters where adjusted and its before its XHR connection is opened. This takes two arguments: the requester *object*and the underlying *XHR object.*
onComplete	Triggered at the very end of a request's life-cycle, once the request completed, status-specific callbacks were called, and possible automatic behaviors were processed.
onException	Triggered whenever an XHR error arises. Has a custom signature: the first argument is the requester (i.e., an Ajax.Request instance), the second is the exception object.
onInteractive	*Not guaranteed* but riggered whenever the requester receives a part of the response (but not the final part), should it be sent in several packets.
onLoaded	*Not guaranteed* but triggered once the underlying XHR object is setup, the connection open, and ready to send its actual request.
onLoading	*Not guaranteed* but triggered when the underlying XHR object is being setup, and its connection opened.
onUninitialized	*Not guaranteed* but invoked when the XHR object was just created.

AJAX PeriodicalUpdater() Method

This AJAX method periodically performs an AJAX request and updates a container's contents based on the response text.

Containers are specified by giving IDs of the HTML elements like division or paragraphs. See example below.

Callbacks are called at various points in the life-cycle of a request, and always feature the same list of arguments. They are passed to requesters right along with their other options.

Syntax

```
new Ajax.PeriodicalUpdater(container, url[, options]);
```

Ajax.PeriodicalUpdater features all the Common Options and callbacks, plus those added by Ajax.Updater().

There are two more options specific to this method:

Option	Description
frequency	**Default value is 2**. This is the minimum interval at which AJAX requests are made.
decay	**Default value is 1**. This controls the rate at which the request interval grows when the response is unchanged.

Return Value

- Returns AJAX PeriodicalUpdater object.

Disabling and Enabling a PeriodicalUpdater

You can pull the brake on a running PeriodicalUpdater by simply calling its stop method. If you wish to re-enable it later, just call its start method. Both take no argument.

Example

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>
```

```
function startTimer()

{


   new Ajax.PeriodicalUpdater('datetime', '/cgi-bin/timer.cgi', {

     method: 'get', frequency: 3, decay: 2

   });


}


</script>

</head>


<body>


  <p>Click start button to see how Current Time changes.</p>

  <p>This example may not work in IE.</p>

  <br />


  <div id="datetime">Current Time</div>

  <br />

  <br />

  <input type="button" value="Start" onclick="startTimer();"/>


</body>

</html>
```

Here is the content of **timer.cgi** script:

```
#!/usr/bin/perl


print "Content-type: text/html\n\n";


$datetime = localtime;

print $datetime;

print "<br />";
```

AJAX Request() Method

This AJAX method initiates and processes an AJAX request. This object is a general-purpose AJAX requester: it handles the life-cycle of the request, handles the boilerplate, and lets you plug in callback functions for your custom needs.

In the optional options hash, you can use any callback function like onComplete and/or onSuccess depending on your custom needs.

Syntax

```
new Ajax.Request(url[, options]);
```

As soon as the object is created, it initiates the request, then goes on processing it throughout its life-cyle. The defined life-cycle is as follows:

- Created
- Initialized
- Request sent
- Response being received (can occur many times, as packets come in)
- Response received, request complete

There is a set of callback functions, defined in Ajax Options, which are triggered in the following order:

- *onCreate* (this is actually a callback reserved to **AJAX global responders**)
- *onUninitialized* (maps on Created)
- *onLoading* (maps on Initialized)
- *onLoaded* (maps on Request sent)
- *onInteractive* (maps on Response being received)

- *onXYZ* (numerical response status code), onSuccess or onFailure (see below)
- *onComplete*

Portability

Depending on how your browser implements *XMLHttpRequest*, one or more callbacks may never be invoked. In particular, *onLoaded* and *onInteractive* are not a 100% safe bet so far. However, the global *onCreate*, *onUninitialized* and the two final steps are very much guaranteed.

Return Value

- new Ajax.Request

Disabling and Enabling a PeriodicalUpdater

You can pull the brake on a running PeriodicalUpdater by simply calling its stop method. If you wish to re-enable it later, just call its start method. Both take no argument.

Example

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function SubmitRequest()

{




  new Ajax.Request('/cgi-bin/ajax.cgi', {

  method: 'get',
  onSuccess: successFunc,
  onFailure:  failureFunc
  });

}

function successFunc(response){

     if (200 == response.status){
        alert("Call is success");
    }
    var container = $('notice');
    var content = response.responseText;
    container.update(content);
}

function failureFunc(response){

     alert("Call is failed" );

}
</script>
</head>

<body>

  <p>Click submit button see how current notice changes.</p>
  <br />


  <div id="notice">Current Notice</div>

  <br />

  <br />

  <input type="button" value="Submit" onclick="SubmitRequest();"/>


</body>

</html>
```

Here is the content of **ajax.cgi**.

```
  #!/usr/bin/perl


  print "Content-type: text/html\n\n";


  print "This content is returned by AJAX cgi <br />";

  print "Current Time " . localtime;
```

Parameters and the HTTP Method

You can pass the parameters for the request as the parameters property in options:

```
  new Ajax.Request('/some_url', {

    method: 'get',

    parameters: {company: 'example', limit: 12}
```

AJAX Responders() Method

The AJAX *Ajax.Responders* let you register global listeners about every step of Prototype-based AJAX requests.

There are two Responders - one is used to register listeners and another can be used to unregister a listener.

Syntax

```
Ajax.Responders.register(responder);


Ajax.Responders.unregister(responder);
```

Return Value

- NA.

Unregister A Responder

If you plan on unregistering a responder, be sure to define it first, then pass the reference to *register*, and finally, when the time comes, to *unregister*.

Example

Following is the example which counts currently active AJAX requests by monitoring their onCreate and onComplete events.

Click the submit button many times and then see the result:

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>



function SubmitRequest()

{
  new Ajax.Request('/cgi-bin/ajax.cgi', {
  method: 'get',
  onSuccess: successFunc  });



}

Ajax.Responders.register({
     onCreate: function() {

      var count = Ajax.activeRequestCount++;
      var container = $('requests');
      container.update(count);
    },

    onComplete: function() {
      var count =  Ajax.activeRequestCount--;
      var container = $('requests');
      container.update(count);
    }
});



function successFunc(response){
    var container = $('notice');
    var content = response.responseText;
    container.update(content);
```

```
}



</script>

</head>


<body>


  <p>Click Submit button many times and see the result.</p>

  <br />


  <div id="notice">Current Notice</div>

  <br />

  <div id="requests">Current Requets</div>

  <br />

  <input type="button" value="Submit" onclick="SubmitRequest();"/>


</body>

</html>
```

Here is the content of **ajax.cgi**.

```
#!/usr/bin/perl


print "Content-type: text/html\n\n";


print "This content is returned by AJAX cgi
";
```

```
   print "Current Time " . localtime;
```

AJAX Response() Method

This AJAX *Ajax.Response* is the object passed as the first argument of all Ajax requests callbacks.

This is a wrapper around the native xmlHttpRequest object. It normalizes cross-browser issues while adding support for JSON via the responseJSON and headerJSON properties.

Properties of the Ajax.Response Object

Property	Type	Description
status	Number	The HTTP status code sent by the server.
statusText	String	The HTTP status text sent by the server.
readyState	Number	The request's current state. 0 corresponds to "Uninitialized", 1 to "Loading", 2 to "Loaded", 3 to "Interactive" and 4 to "Complete".
responseText	String	The text body of the response.
responseXML	document Object or null	The XML body of the response if the content-type of the request is set to application/xml. null otherwise.
responseJSON	Object, Array or null	The JSON body of the response if the content-type of the request is set to application/json. null otherwise.
headerJSON	Object, Array or null	Auto-evaluated content of the X-JSON header if present. null otherwise. This is useful to transfer small amounts of data.
request	Object	The request object itself (an instance of Ajax.Request or Ajax.Updater).

transport	Object	The native xmlHttpRequest object itself.

Example

Following is the example to show the usage of *status* and *responseText* properties:

```
<html>

<head>


<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function SubmitRequest()

{




  new Ajax.Request('/cgi-bin/ajax.cgi', {

  method: 'get',

  onSuccess: successFunc,

  onFailure:  failureFunc

  });


}

function successFunc(response){


     if (200 == response.status){
        alert("Call is success");
    }
    var container = $('notice');
    var content = response.responseText;
    container.update(content);
}
function failureFunc(response){

     alert("Call is failed" );

}
</script>
</head>

<body>

  <p>Click submit button to see how current notice changes.</p>
  <br />

  <div id="notice">Current Notice</div>

  <br />
  <br />
  <input type="button" value="Submit" onclick="SubmitRequest();"/>

</body>
</html>
```

Here is the content of **ajax.cgi**.

```
    #!/usr/bin/perl



    print "Content-type: text/html\n\n";



    print "This content is returned by AJAX cgi
";

    print "Current Time " . localtime;
```

Methods of the Ajax.Response Object

Method	Type	Description
getHeader(name)	String or null	Returns the value of the requested header if present. null otherwise.
getAllHeaders()	String or null	Returns a string containing all headers separated by a line break.
getResponseHeader(name)	String	Returns the value of the requested header if present. Throws an error otherwise. This is just a wrapper around the xmlHttpRequest object.s native method. Prefer it's shorter counterpart getHeader.
getAllResponseHeaders()	String	Returns a string containing all headers separated by a line break. Throws an error otherwise. This is just a wrapper around the xmlHttpRequest object's native method. Prefer it.s shorter counterpart getAllHeaders.

Example

Following is the example to show the usage of *getAllHeaders()* and *getResponseHeader(name)* methods:

```
<html>
<head>

<title>Prototype examples</title>
   <script type="text/javascript"
   src="/javascript/prototype.js">
   </script>
<script>

function SubmitRequest()
{


  new Ajax.Request('/cgi-bin/ajax.cgi', {
  method: 'get',
  onSuccess: successFunc
  });

}

function successFunc(response){

    var content = response.getAllHeaders();
    var container = $(header1);
    container.update(content);
    var content = response.getResponseHeader('Content-Type');

    var container = $(header2);

    container.update(content);


}


</script>

</head>


<body>


  <p>Click submit button to see the result:</p>

  <br />


  <div id="header1">All Headers</div>


  <div id="header2">Content Type</div>


  <br />

  <br />

  <input type="button" value="Submit" onclick="SubmitRequest();"/>


</body>

</html>
```

AJAX Updater() Method

This AJAX *Ajax.Updater* performs an AJAX request and updates a container's contents based on the response text.

Ajax.Updater is a specialization of Ajax.Request.

Syntax

```
new Ajax.Updater(container, url[, options]);
```

Return Value

- An AJAX Ajax.Updater object

Ajax.Updater features all the Common Options and callbacks, plus those added byAjax.Updater().

There are two more options specific to this method:

Option	Description
evalScripts	**Default value is false**. This determines whether <script> elements in the response text are evaluated or not.
insertion	**Default value is None**. By default, Element.update is used, which replaces the whole contents of the container with the response text. You may want to instead insert the response text around existing contents.

In the following example, we assume that creating a new item through AJAX returns an XHTML fragment representing only the new item, which we need to add within our list container, but at the bottom of its existing contents. Here it goes:

```
new Ajax.Updater('items', '/items', {

  parameters: { text: $F('text') },

  insertion: Insertion.Bottom

});
```

Example

Following is the example to show the usage of *Ajax.Updater* to update system time. Each time is getting added at the bottom :

```
<html>

<head>


<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function insertTime()

{


   new Ajax.Updater('datetime', '/cgi-bin/timer.cgi', {

     method: 'get',

     insertion: Insertion.Bottom

   });


}

</script>

</head>


<body>


  <p>Click update button many time to see the result.</p>

  <br />
```

```


    <div id="datetime">Date & Time</div>


    <br />

    <br />

    <input type="button" value="Update" onclick="insertTime();"/>


  </body>

  </html>
```

Here is the content of **timer.cgi**.

```
#!/usr/bin/perl


print "Content-type: text/html\n\n";


$datetime = localtime;

print $datetime;

print "<br />";
```

Single Container, or success/failureAlternative?

Let us assume that in the above example, you're going to update the same container whether your request succeeds or fails. There may very well be times when you don't want that. You may want to update only for successful requests, or update a different container on failed requests.

In the following code, only successful requests get an update:

```
new Ajax.Updater({ success: 'items' }, '/items', {

  parameters: { text: $F('text') },

  insertion: Insertion.Bottom

});
```

The next example assumes failed requests that will feature an error message as response text, and will go on to update another element with it, probably a status zone.

```
new Ajax.Updater({success:'items',failure:'notice' },'/items',

{

  parameters: { text: $F('text') },

  insertion: Insertion.Bottom

});
```

16. Prototype – Expressing Range

Prototype Ranges represent an interval of values. The preferred way to obtain a range is to use the **$R** utility function.

You can create a big range of values using a simple syntax as follows:

```
$R(1, 10).inspect();


$R('a', 'e').inspect();
```

This will produce the following result:

```
['1, 2, 3, 4, 5, 6, 7, 8, 9, 10']


['a', 'b', 'c', 'd', 'e']
```

The include() Method

This method determines whether the value is included in the range:

Syntax

```
Range.include(value);
```

Return Value

- If value is included, then returns a true value otherwise false.

Example

```
<html>

<head>

<title>Prototype examples</title>

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

```
      src="/javascript/prototype.js">
      </script>
   <script>

   function showResult()
   {
      alert ( "Test 1 : " + $R(1, 10).include(5));
      // Returns true

      alert ( "Test 2 : " + $R('a', 'h').include('x'));
      // Returns flase

   }
   </script>
   </head>
   <body>

     <p>Click the button to see the result.</p>
     <br />
     <br />
     <input type="button" value="Result" onclick="showResult();"/>
   </body>
```

17. Prototype – Periodic Expression

Many times it is required to execute a function many times after a certain period of time. For example, you may want to refresh your screen after a given time. Prototype provides a simple mechanism to implement it using *PeriodicalExecuter* object.

The advantage provided by *PeriodicalExecuter* is that it shields you against multiple parallel executions of the callback function.

Creating a PeriodicalExecuter

The constructor takes two arguments:

- The callback function.
- The interval (in seconds) between executions.

Once launched, a PeriodicalExecuter triggers indefinitely, until the page unloads or the executer is stopped using *stop()* method.

Example

Following is the example which will pop up a dialogue box after every 5 seconds untill you will stop it by pressing "cancel' button.

```
<html>

<head>

<title>Prototype examples</title>

   <script type="text/javascript"

   src="/javascript/prototype.js">

   </script>

<script>


function startExec()

{

  new PeriodicalExecuter(function(pe) {

  if (!confirm('Want me to annoy you again later?'))
```

```
        pe.stop();
      }, 5);

    }
    </script>
    </head>

    <body>

      <p>Click start button to start periodic executer:</p>
      <br />

      <br />
      <input type="button" value="start" onclick="startExec();"/>

    </body>
    </html>
```