jQuery Introduction

jQuery is a fast and concise JavaScript Library created by John Resig in 2006 with a nice motto : **Write less, do more**.

jQuery simplifies HTML document traversing, event handling, animating, and Ajax interactions for rapid web development.

jQuery is a JavaScript toolkit designed to simplify various tasks by writing less code. Here is the list of important core features supported by jQuery:

- **DOM manipulation:** The jQuery made it easy to select DOM elements, traverse them and modifying their content by using cross-browser open source selector engine called**Sizzle**.
- **Event handling:** The jQuery offers an elegant way to capture a wide variety of events, such as a user clicking on a link, without the need to clutter the HTML code itself with event handlers.
- **AJAX Support:** The jQuery helps you a lot to develop a responsive and feature-rich site using AJAX technology.
- **Animations:** The jQuery comes with plenty of built-in animation effects which you can use in your websites.
- **Lightweight:** The jQuery is very lightweight library - about 19KB in size (Minified and gzipped).
- **Cross Browser Support:** The jQuery has cross-browser support, and works well in IE 6.0+, FF 2.0+, Safari 3.0+, Chrome and Opera 9.0+
- **Latest Technology:** The jQuery supports CSS3 selectors and basic XPath syntax.

How to install jQuery ?

This is very simple to do require setup to use jQuery library. You have to carry two simple steps:

1. Go to the download page to grab the latest version available.
2. Now put downloaded **jquery-1.3.2.min.js** file in a directory of your website, e.g. /jquery.

The downloaded file name jquery-1.3.2.min.js may vary for your version. Your minified version would be kind of unreadable which would not have any new line or unnecessary words in it.

The jQuery does not require any special installation and very similar to JavaScript, we do not need any compilation or build phase to use jQuery.

How to use jQuery library?

Now you can include *jquery* library in your HTML file as follows:

```
<html>
<head>
<title>The jQuery Example</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript">
      // you can add our javascript code here
   </script>
</head>
<body>
........
</body>
</html>
```

How to call a jQuery library functions?

As almost everything we do when using jQuery reads or manipulates the document object model (DOM), we need to make sure that we start adding events etc. as soon as the DOM is ready.

If you want an event to work on your page, you should call it inside the $(document).ready() function. Everything inside it will load as soon as the DOM is loaded and before the page contents are loaded.

To do this, we register a ready event for the document as follows:

```
$(document).ready(function() {
   // do stuff when DOM is ready
 });
```

To call upon any jQuery library function, use HTML script tags as shown below:

```
<html>
<head>
<title>The jQuery Example</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>

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

   $(document).ready(function()
   {
      $("div").click(function()
      {
        alert("Hello world!");
      });
   });

   </script>

</head>
<body>
<div id="newdiv">
Click on this to see a dialogue box.
</div>
</body>
</html>
```

How to use Custom Scripts?

It is better to write our custom code in the custom JavaScript file : **custom.js**, as follows:

```
/* Filename: custom.js */
$(document).ready(function() {
  $("div").click(function() {
         alert("Hello world!");
  });
});
```

Now we can include **custom.js** file in our HTML file as follows:

```
<html>
<head>
<title>The jQuery Example</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript"
```

```
   src="/jquery/custom.js"></script>
</head>
<body>
<div id="newdiv">
Click on this to see a dialogue box.
</div>
</body>
</html>
```

jQuery is a framework built using JavaScript capabilities. So you can use all the functions and other capabilities available in JavaScript.

This chapter would explain most basic concepts but frequently used in jQuery.

String:

A string in JavaScript is an immutable object that contains none, one or many characters.

Following are the valid examples of a JavaScript String:

```
"This is JavaScript String"
'This is JavaScript String'
'This is "really" a JavaScript String'
"This is 'really' a JavaScript String"
```

Numbers:

Numbers in JavaScript are double-precision 64-bit format IEEE 754 values. They are immutable, just as strings.

Following are the valid examples of a JavaScript Numbers:

```
5350
120.27
0.26
```

Boolean:

A boolean in JavaScript can be either **true** or **false**. If a number is zero, it defaults to false. If an empty string defaults to false:

Following are the valid examples of a JavaScript Boolean:

```
true      // true
false     // false
0         // false
1         // true
""        // false
"hello"   // true
```

Objects:

JavaScript supports Object concept very well. You can create an object using the object literal as follows:

```
var emp = {
   name: "Zara",
   age: 10
};
```

You can write and read properties of an object using the dot notation as follows:

```
// Getting object properties
emp.name  // ==> Zara
emp.age   // ==> 10

// Setting object properties
emp.name = "Daisy"  // <== Daisy
emp.age  =  20      // <== 20
```

Arrays:

You can define arrays using the array literal as follows:

```
var x = [];
var y = [1, 2, 3, 4, 5];
```

An array has a **length** property that is useful for iteration:

```
var x = [1, 2, 3, 4, 5];
for (var i = 0; i < x.length; i++) {
   // Do something with x[i]
 }
```

Functions:

A function in JavaScript can be either named or anonymous. A named function can be defined using *function* keyword as follows:

```
function named(){
  // do some stuff here
}
```

An anonymous function can be defined in similar way as a normal function but it would not have any name.

A anonymous function can be assigned to a variable or passed to a method as shown below.

```
var handler = function (){
  // do some stuff here
}
```

JQuery makes a use of anonymous functions very frequently as follows:

```
$(document).ready(function(){
  // do some stuff here
});
```

Arguments:

JavaScript variable *arguments* is a kind of array which has *length* property. Following example explains it very well:

```
function func(x){
  console.log(typeof x, arguments.length);
}
func();                 //==> "undefined", 0
func(1);                //==> "number", 1
func("1", "2", "3");    //==> "string", 3
```

The arguments object also has a *callee* property, which refers to the function you're inside of. For example:

```
function func() {
   return arguments.callee;
}
func();                 // ==> func
```

Context:

JavaScript famous keyword **this** always refers to the current context. Within a function **this**context can change, depending on how the function is called:

```
$(document).ready(function() {
  // this refers to window.document
});

$("div").click(function() {
  // this refers to a div DOM element
});
```

You can specify the context for a function call using the function-built-in methods **call()** and**apply()** methods.

The difference between them is how they pass arguments. Call passes all arguments through as arguments to the function, while apply accepts an array as the arguments.

```
function scope() {
  console.log(this, arguments.length);
}

scope() // window, 0
scope.call("foobar", [1,2]);  //==> "foobar", 1
scope.apply("foobar", [1,2]); //==> "foobar", 2
```

Scope:

The scope of a variable is the region of your program in which it is defined. JavaScript variable will have only two scopes.

- **Global Variables:** A global variable has global scope which means it is defined everywhere in your JavaScript code.
- **Local Variables:** A local variable will be visible only within a function where it is defined. Function parameters are always local to that function.

Within the body of a function, a local variable takes precedence over a global variable with the same name:

```
var myVar = "global";     // ==> Declare a global variable

function ( ) {
   var myVar = "local";   // ==> Declare a local variable
   document.write(myVar); // ==> local
}
```

Callback:

A callback is a plain JavaScript function passed to some method as an argument or option. Some callbacks are just events, called to give the user a chance to react when a certain state is triggered.

jQuery's event system uses such callbacks everywhere for example:

```
$("body").click(function(event) {
   console.log("clicked: " + event.target);
 });
```

Most callbacks provide arguments and a context. In the event-handler example, the callback is called with one argument, an Event.

Some callbacks are required to return something, others make that return value optional. To prevent a form submission, a submit event handler can return false as follows:

```
$("#myform").submit(function() {
   return false;
 });
```

Closures:

Closures are created whenever a variable that is defined outside the current scope is accessed from within some inner scope.

Following example shows how the variable **counter** is visible within the create, increment, and print functions, but not outside of them:

```
function create() {
  var counter = 0;
  return {
    increment: function() {
      counter++;
    },
    print: function() {
      console.log(counter);
    }
  }
}
var c = create();
c.increment();
c.print();     // ==> 1
```

This pattern allows you to create objects with methods that operate on data that isn't visible to the outside world. It should be noted that **data hiding** is the very basis of object-oriented programming.

Proxy Pattern:

A proxy is an object that can be used to control access to another object. It implements the same interface as this other object and passes on any method invocations to it. This other object is often called the real subject.

A proxy can be instantiated in place of this real subject and allow it to be accessed remotely. We can saves jQuery's setArray method in a closure and overwrites it as follows:

```
(function() {
  // log all calls to setArray
  var proxied = jQuery.fn.setArray;

  jQuery.fn.setArray = function() {
    console.log(this, arguments);
    return proxied.apply(this, arguments);
  };
})();
```

The above wraps its code in a function to hide the *proxied* variable. The proxy then logs all calls to the method and delegates the call to the original method. Using *apply(this, arguments)*guarantees that the caller won't be able to notice the difference between the original and the proxied method.

Built-in Functions:

JavaScript comes along with a useful set of built-in functions. These methods can be used to manipulate Strings, Numbers and Dates.

Following are important JavaScript functions:

Method	Description
charAt()	Returns the character at the specified index.
concat()	Combines the text of two strings and returns a new string.
forEach()	Calls a function for each element in the array.
indexOf()	Returns the index within the calling String object of the first occurrence of the specified value, or -1 if not found.
length()	Returns the length of the string.
pop()	Removes the last element from an array and returns that element.
push()	Adds one or more elements to the end of an array and returns the new length of the array.
reverse()	Reverses the order of the elements of an array -- the first becomes the last, and the last becomes the first.
sort()	Sorts the elements of an array.
substr()	Returns the characters in a string beginning at the specified location through the specified number of characters.
toLowerCase()	Returns the calling string value converted to lower case.
toString()	Returns the string representation of the number's value.
toUpperCase()	Returns the calling string value converted to uppercase.

A complete list of built-in function is available here : Built-in Functions.

The Document Object Model:

The Document Object Model is a tree structure of various elements of HTML as follows:

```
<html>
<head>
   <title>the title</title>
</head>
<body>
   <div>
      <p>This is a paragraph.</p>
      <p>This is second paragraph.</p>
      <p>This is third paragraph.</p>
   </div>
</body>
</html>
```

Following are the important points about the above tree structure:

- The <html> is the ancestor of all the other elements; in other words, all the other elements are descendants of <html>.
- The <head> and <body> elements are not only descendants, but children of <html>, as well.
- Likewise, in addition to being the ancestor of <head> and <body>, <html> is also their parent.
- The <p> elements are children (and descendants) of , descendants of <body> and <html>, and siblings of each other <p> elements.

The jQuery library harnesses the power of Cascading Style Sheets (CSS) selectors to let us quickly and easily access elements or groups of elements in the Document Object Model (DOM).

A jQuery Selector is a function which makes use of expressions to find out matching elements from a DOM based on the given criteria.

The $() factory function:

All type of selectors available in jQuery, always start with the dollar sign and parentheses: **$()**.

The factory function **$()** makes use of following three building blocks while selecting elements in a given document:

jQuery	Description
Tag Name:	Represents a tag name available in the DOM. For example **$('p')**selects all paragraphs in the document.
Tag ID:	Represents a tag available with the given ID in the DOM. For example**$('#some-id')** selects the single element in the document that has an ID of some-id.
Tag Class:	Represents a tag available with the given class in the DOM. For example **$('.some-class')** selects all elements in the document that have a class of some-class.

All the above items can be used either on their own or in combination with other selectors. All the jQuery selectors are based on the same principle except some tweaking.

NOTE: The factory function **$()** is a synonym of **jQuery()** function. So in case you are using any other JavaScript library where **$** sign is conflicting with some thing else then you can replace**$** sign by **jQuery** name and you can use function **jQuery()** instead of **$()**.

Example:

Following is a simple example which makes use of Tag Selector. This would select all the elements with a tag name **p**.

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>

   <script type="text/javascript" language="javascript">
   $(document).ready(function() {
      var pars = $("p");
      for( i=0; i<pars.length; i++ ){
         alert("Found paragraph: " + pars[i].innerHTML);
      }
   });
   </script>
</head>
<body>
   <div>
      <p class="myclass">This is a paragraph.</p>
      <p id="myid">This is second paragraph.</p>
      <p>This is third paragraph.</p>
   </div>
</body>
</html>
```

How to use Selectors?

The selectors are very useful and would be required at every step while using jQuery. They get the exact element that you want from your HTML document.

Following table lists down few basic selectors and explains them with examples.

Selector	Description
Name	Selects all elements which match with the given element**Name**.
#ID	Selects a single element which matches with the given**ID**
.Class	Selects all elements which match with the given **Class**.
Universal (*)	Selects all elements available in a DOM.
Multiple Elements E, F, G	Selects the combined results of all the specified selectors **E, F** or **G**.

Similar to above syntax and examples, following examples would give you understanding on using different type of other useful selectors:

- **$('*'):** This selector selects all elements in the document.
- **$("p > *"):** This selector selects all elements that are children of a paragraph element.
- **$("#specialID"):** This selector function gets the element with id="specialID".
- **$(".specialClass"):** This selector gets all the elements that have the class of*specialClass*.
- **$("li:not(.myclass)"):** Selects all elements matched by <li> that do not have class="myclass".
- **$("a#specialID.specialClass"):** This selector matches links with an id of *specialID* and a class of *specialClass*.
- **$("p a.specialClass"):** This selector matches links with a class of *specialClass* declared within <p> elements.
- **$("ul li:first"):** This selector gets only the first <li> element of the <ul>.
- **$("#container p"):** Selects all elements matched by <p> that are descendants of an element that has an id of *container*.
- **$("li > ul"):** Selects all elements matched by <ul> that are children of an element matched by

- **$("strong + em"):** Selects all elements matched by that immediately follow a sibling element matched by .
- **$("p ~ ul"):** Selects all elements matched by <ul> that follow a sibling element matched by <p>.
- **$("code, em, strong"):** Selects all elements matched by <code> or or .
- **$("p strong, .myclass"):** Selects all elements matched by that are descendants of an element matched by <p> as well as all elements that have a class of*myclass*.
- **$(":empty"):** Selects all elements that have no children.
- **$("p:empty"):** Selects all elements matched by <p> that have no children.
- **$("div[p]"):** Selects all elements matched by that contain an element matched by <p>.
- **$("p[.myclass]"):** Selects all elements matched by <p> that contain an element with a class of *myclass*.
- **$("a[@rel]"):** Selects all elements matched by <a> that have a rel attribute.
- **$("input[@name=myname]"):** Selects all elements matched by <input> that have a name value exactly equal to *myname.*
- **$("input[@name^=myname]"):** Selects all elements matched by <input> that have a name value beginning with *myname*.
- **$("a[@rel$=self]"):** Selects all elements matched by <a> that have **rel** attribute value ending with *self*
- **$("a[@href*=domain.com]"):** Selects all elements matched by <a> that have an href value containing domain.com.
- **$("li:even"):** Selects all elements matched by <li> that have an even index value.
- **$("tr:odd"):** Selects all elements matched by <tr> that have an odd index value.
- **$("li:first"):** Selects the first <li> element.
- **$("li:last"):** Selects the last <li> element.
- **$("li:visible"):** Selects all elements matched by <li> that are visible.
- **$("li:hidden"):** Selects all elements matched by <li> that are hidden.
- **$(":radio"):** Selects all radio buttons in the form.
- **$(":checked"):** Selects all checked boxex in the form.
- **$(":input"):** Selects only form elements (input, select, textarea, button).
- **$(":text"):** Selects only text elements (input[type=text]).
- **$("li:eq(2)"):** Selects the third <li> element
- **$("li:eq(4)"):** Selects the fifth <li> element
- **$("li:lt(2)"):** Selects all elements matched by <li> element before the third one; in other words, the first two <li> elements.
- **$("p:lt(3)"):** selects all elements matched by <p> elements before the fourth one; in other words the first three <p> elements.
- **$("li:gt(1)"):** Selects all elements matched by <li> after the second one.
- **$("p:gt(2)"):** Selects all elements matched by <p> after the third one.
- **$("div/p"):** Selects all elements matched by <p> that are children of an element matched by .
- **$("div//code"):** Selects all elements matched by <code>that are descendants of an element matched by .
- **$("//p//a"):** Selects all elements matched by <a> that are descendants of an element matched by <p>
- **$("li:first-child"):** Selects all elements matched by <li> that are the first child of their parent.
- **$("li:last-child"):** Selects all elements matched by <li> that are the last child of their parent.
- **$(":parent"):** Selects all elements that are the parent of another element, including text.
- **$("li:contains(second)"):** Selects all elements matched by <li> that contain the text second.

You can use all the above selectors with any HTML/XML element in generic way. For example if selector **$("li:first")** works for <li> element then **$("p:first")** would also work for <p> element.

Some of the most basic components we can manipulate when it comes to DOM elements are the properties and attributes assigned to those elements.

Most of these attributes are available through JavaScript as DOM node properties. Some of the more common properties are:

- className
- tagName
- id

- href
- title
- rel
- src

Consider the following HTML markup for an image element:

```
<img id="myImage" src="image.gif" alt="An image"
class="someClass" title="This is an image"/>
```

In this element's markup, the tag name is img, and the markup for id, src, alt, class, and title represents the element's attributes, each of which consists of a name and a value.

jQuery gives us the means to easily manipulate an element's attributes and gives us access to the element so that we can also change its properties.

Get Attribute Value:

The **attr()** method can be used to either fetch the value of an attribute from the first element in the matched set or set attribute values onto all matched elements.

Example:

Following is a simple example which fetches title attribute of tag and set value with the same value:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {
      var title = $("em").attr("title");
      $("#divid").text(title);
   });

   </script>
</head>
<body>
   <div>
      <em title="Bold and Brave">This is first paragraph.</em>
      <p id="myid">This is second paragraph.</p>
      <div id="divid"></div>
   </div>
</body>
</html>
```

Set Attribute Value:

The **attr(name, value)** method can be used to set the named attribute onto all elements in the wrapped set using the passed value.

Example:

Following is a simple example which set **src** attribute of an image tag to a correct location:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {
      $("#myimg").attr("src", "/images/jquery.jpg");
   });

   </script>
</head>
<body>
   <div>
      <img id="myimg" src="/wongpath.jpg" alt="Sample image" />
   </div>
</body>
</html>
```

Applying Styles:

The **addClass(classes)** method can be used to apply defined style sheets onto all the matched elements. You can specify multiple classes separated by space.

Example:

Following is a simple example which sets **class** attribute of a para <p> tag:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {
      $("em").addClass("selected");
      $("#myid").addClass("highlight");
   });

   </script>
   <style>
      .selected { color:red; }
      .highlight { background:yellow; }
  </style>
</head>
<body>
   <em title="Bold and Brave">This is first paragraph.</em>
   <p id="myid">This is second paragraph.</p>
</body>
</html>
```

Useful Attribute Methods:

Following table lists down few useful methods which you can use to manipulate attributes and properties:

Methods	Description
attr(properties)	Set a key/value object as properties to all matched elements.
attr(key, fn)	Set a single property to a computed value, on all matched elements.
removeAttr(name)	Remove an attribute from each of the matched elements.
hasClass(class)	Returns true if the specified class is present on at least one of the set of matched elements.
removeClass(class)	Removes all or the specified class(es) from the set of matched elements.
toggleClass(class)	Adds the specified class if it is not present, removes the specified class if it is present.
html( )	Get the html contents (innerHTML) of the first matched element.
html(val)	Set the html contents of every matched element.
text( )	Get the combined text contents of all matched elements.
text(val)	Set the text contents of all matched elements.
val( )	Get the input value of the first matched element.
val(val)	Set the value attribute of every matched element if it is called on <input> but if it is called on <select> with the passed <option> value then passed option would be selected, if it is called on check box or radio box then all the matching check box and radiobox would be checked.

Similar to above syntax and examples, following examples would give you understanding on using various attribute methods in different situation:

- **$("#myID").attr("custom") :** This would return value of attribute *custom* for the first element matching with ID myID.
- **$("img").attr("alt", "Sample Image"):** This sets the **alt** attribute of all the images to a new value "Sample Image".
- **$("input").attr({ value: "", title: "Please enter a value" }); :** Sets the value of all <input> elements to the empty string, as well as sets the title to the string *Please enter a value*.
- **$("a[href^=http://]").attr("target","_blank"):** Selects all links with an href attribute starting with *http://* and set its target attribute to *_blank*
- **$("a").removeAttr("target") :** This would remove *target* attribute of all the links.
- **$("form").submit(function() {$(":submit",this).attr("disabled", "disabled");}); :** This would modify the disabled attribute to the value "disabled" while clicking Submit button.
- **$("p:last").hasClass("selected"):** This return true if last <p> tag has associated class*selected*.
- **$("p").text():** Returns string that contains the combined text contents of all matched <p> elements.
- **$("p").text("Hello World"):** This would set "<I>Hello World</I>" as text content of the matching <p> elements
- **$("p").html() :** This returns the HTML content of the all matching paragraphs.
- **$("div").html("Hello World") :** This would set the HTML content of all matching to *Hello World*.
- **$("input:checkbox:checked").val() :** Get the first value from a checked checkbox
- **$("input:radio[name=bar]:checked").val():** Get the first value from a set of radio buttons
- **$("button").val("Hello") :** Sets the value attribute of every matched element <button>.
- **$("input").val("on") :** This would check all the radio or check box button whose value is "on".
- **$("select").val("Orange") :** This would select Orange option in a dropdown box with options Orange, Mango and Banana.

- **$("select").val("Orange", "Mango") :** This would select Orange and Mango options in a dropdown box with options Orange, Mango and Banana.

jQuery is a very powerful tool which provides a variety of DOM traversal methods to help us select elements in a document randomly as well as in sequential method.

Most of the DOM Traversal Methods do not modify the jQuery object and they are used to filter out elements from a document based on given conditions.

Find Elements by index:

Consider a simple document with the following HTML content:

```
<html>
<head>
<title>the title</title>
</head>
<body>
   <div>
   <ul>
     <li>list item 1</li>
     <li>list item 2</li>
     <li>list item 3</li>
     <li>list item 4</li>
     <li>list item 5</li>
     <li>list item 6</li>
   </ul>
   </div>
</body>
</html>
```

- Above every list has its own index, and can be located directly by using **eq(index)**method as below example.
- Every child element starts its index from zero, thus, *list item 2* would be accessed by using **$("li").eq(1)** and so on.

Example:

Following is a simple example which adds the color to second list item.

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {
      $("li").eq(2).addClass("selected");
   });

   </script>
   <style>
      .selected { color:red; }
  </style>
</head>
<body>
   <div>
   <ul>
     <li>list item 1</li>
     <li>list item 2</li>
```

```
     <li>list item 3</li>
     <li>list item 4</li>
     <li>list item 5</li>
     <li>list item 6</li>
   </ul>
   </div>
</body>
</html>
```

Filtering out Elements:

The **filter(selector)** method can be used to filter out all elements from the set of matched elements that do not match the specified selector(s). The *selector* can be written using any selector syntax.

Example:

Following is a simple example which applies color to the lists associated with middle class:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {
      $("li").filter(".middle").addClass("selected");
   });

   </script>
   <style>
      .selected { color:red; }
  </style>
</head>
<body>
   <div>
   <ul>
     <li class="top">list item 1</li>
     <li class="top">list item 2</li>
     <li class="middle">list item 3</li>
     <li class="middle">list item 4</li>
     <li class="bottom">list item 5</li>
     <li class="bottom">list item 6</li>
   </ul>
   </div>
</body>
</html>
```

Locating Descendent Elements :

The **find(selector)** method can be used to locate all the descendent elements of a particular type of elements. The *selector* can be written using any selector syntax.

Example:

Following is an example which selects all the elements available inside different <p> elements:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {
      $("p").find("span").addClass("selected");
   });

   </script>
   <style>
      .selected { color:red; }
  </style>
</head>
<body>
   <p>This is 1st paragraph and <span>THIS IS RED</span></p>
   <p>This is 2nd paragraph and <span>THIS IS ALSO RED</span></p>
</body>
</html>
```

JQuery DOM Traversing Methods:

Following table lists down useful methods which you can use to filter out various elements from a list of DOM elements:

Selector	Description
eq(index)	Reduce the set of matched elements to a single element.
filter(selector)	Removes all elements from the set of matched elements that do not match the specified selector(s).
filter(fn)	Removes all elements from the set of matched elements that do not match the specified function.
is(selector)	Checks the current selection against an expression and returns true, if at least one element of the selection fits the given selector.
map(callback)	Translate a set of elements in the jQuery object into another set of values in a jQuery array (which may, or may not contain elements).
not(selector)	Removes elements matching the specified selector from the set of matched elements.
slice(start, [end])	Selects a subset of the matched elements.

Following table lists down other useful methods which you can use to locate various elements in a DOM:

Selector	Description
add(selector)	Adds more elements, matched by the given selector, to the set of matched elements.
andSelf( )	Add the previous selection to the current selection.
children([selector])	Get a set of elements containing all of the unique immediate children of each of the matched set of elements.
closest(selector)	Get a set of elements containing the closest parent element that

	matches the specified selector, the starting element included.
contents( )	Find all the child nodes inside the matched elements (including text nodes), or the content document, if the element is an iframe.
end( )	Revert the most recent 'destructive' operation, changing the set of matched elements to its previous state .
find(selector)	Searches for descendent elements that match the specified selectors.
next([selector])	Get a set of elements containing the unique next siblings of each of the given set of elements.
nextAll([selector])	Find all sibling elements after the current element.
offsetParent( )	Returns a jQuery collection with the positioned parent of the first matched element.
parent([selector])	Get the direct parent of an element. If called on a set of elements, parent returns a set of their unique direct parent elements.
parents([selector])	Get a set of elements containing the unique ancestors of the matched set of elements (except for the root element).
prev([selector])	Get a set of elements containing the unique previous siblings of each of the matched set of elements.
prevAll([selector])	Find all sibling elements in front of the current element.
siblings([selector])	Get a set of elements containing all of the unique siblings of each of the matched set of elements.

jQuery - CSS Methods

The jQuery library supports nearly all of the selectors included in Cascading Style Sheet (CSS) specifications 1 through 3, as outlined on the World Wide Web Consortium's site.

Using JQuery library developers can enhance their websites without worrying about browsers and their versions as long as the browsers have JavaScript enabled.

Most of the JQuery CSS Methods do not modify the content of the jQuery object and they are used to apply CSS properties on DOM elements.

Apply CSS Properties:

This is very simple to apply any CSS property using JQuery method **css(PropertyName, PropertyValue)**.

Here is the syntax for the method:

```
selector.css( PropertyName, PropertyValue );
```

Here you can pass *PropertyName* as a javascript string and based on its value, *PropertyValue*could be string or integer.

Example:

Following is an example which adds font color to the second list item.

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {
      $("li").eq(2).css("color", "red");
   });

   </script>
</head>
<body>
   <div>
   <ul>
     <li>list item 1</li>
     <li>list item 2</li>
     <li>list item 3</li>
     <li>list item 4</li>
     <li>list item 5</li>
     <li>list item 6</li>
   </ul>
   </div>
</body>
</html>
```

Apply Multiple CSS Properties:

You can apply multiple CSS properties using a single JQuery method **CSS({key1:val1, key2:val2....)**. You can apply as many properties as you like in a single call.

Here is the syntax for the method:

```
selector.css( {key1:val1, key2:val2....keyN:valN})
```

Here you can pass key as property and val as its value as described above.

Example:

Following is an example which adds font color as well as background color to the second list item.

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {
      $("li").eq(2).css({"color":"red",
                         "background-color":"green"});
   });

   </script>
</head>
<body>
   <div>
```

```
   <ul>
     <li>list item 1</li>
     <li>list item 2</li>
     <li>list item 3</li>
     <li>list item 4</li>
     <li>list item 5</li>
     <li>list item 6</li>
   </ul>
   </div>
</body>
</html>
```

Setting Element Width & Height:

The **width(val)** and **height(val)** method can be used to set the width and hieght respectively of any element.

Example:

Following is a simple example which sets the width of first division element where as rest of the elements have width set by style sheet:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {
      $("div:first").width(100);
      $("div:first").css("background-color", "blue");
   });

   </script>
   <style>
   div{ width:70px; height:50px; float:left; margin:5px;
      background:red; cursor:pointer; }
  </style>
</head>
<body>
  <div></div>
  <div>d</div>
  <div>d</div>
  <div>d</div>
  <div>d</div>
</body>
</html>
```

JQuery CSS Methods:

Following table lists down all the methods which you can use to play with CSS properties:

Method	Description
css(name)	Return a style property on the first matched element.

css(name, value)	Set a single style property to a value on all matched elements.
css(properties)	Set a key/value object as style properties to all matched elements.
height(val)	Set the CSS height of every matched element.
height( )	Get the current computed, pixel, height of the first matched element.
innerHeight( )	Gets the inner height (excludes the border and includes the padding) for the first matched element.
innerWidth( )	Gets the inner width (excludes the border and includes the padding) for the first matched element.
offset( )	Get the current offset of the first matched element, in pixels, relative to the document
offsetParent( )	Returns a jQuery collection with the positioned parent of the first matched element.
outerHeight([margin])	Gets the outer height (includes the border and padding by default) for the first matched element.
outerWidth([margin])	Get the outer width (includes the border and padding by default) for the first matched element.
position( )	Gets the top and left position of an element relative to its offset parent.
scrollLeft(val)	When a value is passed in, the scroll left offset is set to that value on all matched elements.
scrollLeft( )	Gets the scroll left offset of the first matched element.
scrollTop(val)	When a value is passed in, the scroll top offset is set to that value on all matched elements.
scrollTop( )	Gets the scroll top offset of the first matched element.
width(val)	Set the CSS width of every matched element.
width( )	Get the current computed, pixel, width of the first matched element.

jQuery - DOM Manipulation Methods

JQuery provides methods to manipulate DOM in efficient way. You do not need to write big code to modify the value of any element's attribute or to extract HTML code from a paragraph or division.

JQuery provides methods such as .attr(), .html(), and .val() which act as getters, retrieving information from DOM elements for later use.

Content Manipulation:

The **html( )** method gets the html contents (innerHTML) of the first matched element.

Here is the syntax for the method:

```
selector.html( )
```

Example:

Following is an example which makes use of .html() and .text(val) methods. Here .html() retrieves HTML content from the object and then .text(val) method sets value of the object using passed parameter:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {

     $("div").click(function () {
      var content = $(this).html();
      $("#result").text( content );
    });

   });

   </script>
   <style>
      #division{ margin:10px;padding:12px;
                 border:2px solid #666;
                 width:60px;
               }
  </style>
</head>
<body>
   <p>Click on the square below:</p>
   <span id="result"> </span>
   <div id="division" style="background-color:blue;">
     This is Blue Square!!
   </div>
</body>
</html>
```

DOM Element Replacement:

You can replace a complete DOM element with the specified HTML or DOM elements. The**replaceWith(content)** method serves this purpose very well.

Here is the syntax for the method:

```
selector.replaceWith( content )
```

Here content is what you want to have instead of original element. This could be HTML or simple text.

Example:

Following is an example which would replace division element with "<h1>JQuery is Great</h1>":

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

```
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {

     $("div").click(function () {
       $(this).replaceWith("<h1>JQuery is Great</h1>");
    });

   });

   </script>
   <style>
      #division{ margin:10px;padding:12px;
                 border:2px solid #666;
                 width:60px;
               }
  </style>
</head>
<body>
   <p>Click on the square below:</p>
   <span id="result"> </span>
   <div id="division" style="background-color:blue;">
     This is Blue Square!!
   </div>
</body>
</html>
```

Removing DOM Elements:

There may be a situation when you would like to remove one or more DOM elements from the document. JQuery provides two methods to handle the situation.

The **empty( )** method remove all child nodes from the set of matched elements where as the method **remove(expr)** method removes all matched elements from the DOM.

Here is the syntax for the method:

```
selector.remove( [ expr ])

or

selector.empty( )
```

You can pass optional paramter *expr* to filter the set of elements to be removed.

Example:

Following is an example where elements are being removed as soon as they are clicked:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {
```

```
    $("div").click(function () {
      $(this).remove( );
    });

   });

   </script>
   <style>
      .div{ margin:10px;padding:12px;
            border:2px solid #666;
            width:60px;
          }
  </style>
</head>
<body>
   <p>Click on any square below:</p>
   <span id="result"> </span>
   <div class="div" style="background-color:blue;"></div>
   <div class="div" style="background-color:green;"></div>
   <div class="div" style="background-color:red;"></div>
</body>
</html>
```

Inserting DOM elements:

There may be a situation when you would like to insert new one or more DOM elements in your existing document. JQuery provides various methods to insert elements at various locations.

The **after(content)** method insert content after each of the matched elements where as the method **before(content)** method inserts content before each of the matched elements.

Here is the syntax for the method:

```
selector.after( content )

or

selector.before( content )
```

Here content is what you want to insert. This could be HTML or simple text.

Example:

Following is an example where elements are being inserted just before the clicked element:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {

     $("div").click(function () {
       $(this).before('<div class="div"></div>' );
    });
```

```
   });

   </script>
   <style>
      .div{ margin:10px;padding:12px;
            border:2px solid #666;
            width:60px;
          }
  </style>
</head>
<body>
   <p>Click on any square below:</p>
   <span id="result"> </span>
   <div class="div" style="background-color:blue;"></div>
   <div class="div" style="background-color:green;"></div>
   <div class="div" style="background-color:red;"></div>
</body>
</html>
```

DOM Manipulation Methods:

Following table lists down all the methods which you can use to manipulate DOM elements:

Method	Description
after(content)	Insert content after each of the matched elements.
append(content)	Append content to the inside of every matched element.
appendTo(selector)	Append all of the matched elements to another, specified, set of elements.
before(content)	Insert content before each of the matched elements.
clone(bool)	Clone matched DOM Elements, and all their event handlers, and select the clones.
clone( )	Clone matched DOM Elements and select the clones.
empty( )	Remove all child nodes from the set of matched elements.
html(val)	Set the html contents of every matched element.
html( )	Get the html contents (innerHTML) of the first matched element.
insertAfter(selector)	Insert all of the matched elements after another, specified, set of elements.
insertBefore(selector)	Insert all of the matched elements before another, specified, set of elements.
prepend(content)	Prepend content to the inside of every matched element.
prependTo(selector)	Prepend all of the matched elements to another, specified, set of elements.
remove(expr)	Removes all matched elements from the DOM.
replaceAll(selector)	Replaces the elements matched by the specified selector with the matched elements.
replaceWith(content)	Replaces all matched elements with the specified HTML or DOM elements.

text(val)	Set the text contents of all matched elements.
text( )	Get the combined text contents of all matched elements.
wrap(elem)	Wrap each matched element with the specified element.
wrap(html)	Wrap each matched element with the specified HTML content.
wrapAll(elem)	Wrap all the elements in the matched set into a single wrapper element.
wrapAll(html)	Wrap all the elements in the matched set into a single wrapper element.
wrapInner(elem)	Wrap the inner child contents of each matched element (including text nodes) with a DOM element.
wrapInner(html)	Wrap the inner child contents of each matched element (including text nodes) with an HTML structure.

jQuery - Events Handling

We have the ability to create dynamic web pages by using events. Events are actions that can be detected by your Web Application.

Following are the examples events:

- A mouse click
- A web page loading
- Taking mouse over an element
- Submitting an HTML form
- A keystroke on your keyboard
- etc.

When these events are triggered you can then use a custom function to do pretty much whatever you want with the event. These custom functions call Event Handlers.

Binding event handlers:

Using the jQuery Event Model, we can establish event handlers on DOM elements with the **bind()**method as follows:

```
$('div').bind('click', function( event ){
   alert('Hi there!');
});
```

This code will cause the division element to respond to the click event; when a user clicks inside this division thereafter, the alert will be shown. Try it yourself.

The full syntax of the bind() command is as follows:

```
selector.bind( eventType[, eventData], handler)
```

Following is the description of the parameters:

- **eventType:** A string containing a JavaScript event type, such as click or submit. Refer to the next section for a complete list of event types.
- **eventData:** This is optional parameter is a map of data that will be passed to the event handler.
- **handler:** A function to execute each time the event is triggered.

Removing event handlers:

Typically, once an event handler is established, it remains in effect for the remainder of the life of the page. There may be a need when you would like to remove event handler.

jQuery provides the **unbind()** command to remove an exiting event handler. The syntax of unbind() is as follows:

```
selector.unbind(eventType, handler)

or

selector.unbind(eventType)
```

Following is the description of the parameters:

- **eventType:** A string containing a JavaScript event type, such as click or submit. Refer to the next section for a complete list of event types.
- **handler:** If provided, identifies the specific listener that.s to be removed.

Event Types:

The following are cross platform and recommended event types which you can bind using JQuery:

Event Type	Description
blur	Occurs when the element loses focus
change	Occurs when the element changes
click	Occurs when a mouse click
dblclick	Occurs when a mouse double-click
error	Occurs when there is an error in loading or unloading etc.
focus	Occurs when the element gets focus
keydown	Occurs when key is pressed
keypress	Occurs when key is pressed and released
keyup	Occurs when key is released

load	Occurs when document is loaded
mousedown	Occurs when mouse button is pressed
mouseenter	Occurs when mouse enters in an element region
mouseleave	Occurs when mouse leaves an element region
mousemove	Occurs when mouse pointer moves
mouseout	Occurs when mouse pointer moves out of an element
mouseover	Occurs when mouse pointer moves over an element
mouseup	Occurs when mouse button is released
resize	Occurs when window is resized
scroll	Occurs when window is scrolled
select	Occurs when a text is selected
submit	Occurs when form is submitted
unload	Occurs when documents is unloaded

The Event Object:

The callback function takes a single parameter; when the handler is called the JavaScript event object will be passed through it.

The event object is often unneccessary and the parameter is omitted, as sufficient context is usually available when the handler is bound to know exactly what needs to be done when the handler is triggered, however there are certail attributes which you would need to be accessed.

The Event Attributes:

The following event properties/attributes are available and safe to access in a platform independent manner:

Property	Description
altKey	Set to true if the Alt key was pressed when the event was triggered, false if not. The Alt key is labeled Option on most Mac keyboards.
ctrlKey	Set to true if the Ctrl key was pressed when the event was triggered, false if not.

data	The value, if any, passed as the second parameter to the bind() command when the handler was established.
keyCode	For keyup and keydown events, this returns the key that was pressed.
metaKey	Set to true if the Meta key was pressed when the event was triggered, false if not. The Meta key is the Ctrl key on PCs and the Command key on Macs.
pageX	For mouse events, specifies the horizontal coordinate of the event relative from the page origin.
pageY	For mouse events, specifies the vertical coordinate of the event relative from the page origin.
relatedTarget	For some mouse events, identifies the element that the cursor left or entered when the event was triggered.
screenX	For mouse events, specifies the horizontal coordinate of the event relative from the screen origin.
screenY	For mouse events, specifies the vertical coordinate of the event relative from the screen origin.
shiftKey	Set to true if the Shift key was pressed when the event was triggered, false if not.
target	Identifies the element for which the event was triggered.
timeStamp	The timestamp (in milliseconds) when the event was created.
type	For all events, specifies the type of event that was triggered (for example, click).
which	For keyboard events, specifies the numeric code for the key that caused the event, and for mouse events, specifies which button was pressed (1 for left, 2 for middle, 3 for right)

The Event Methods:

There is a list of methods which can be called on an Event Object:

Method	Description
preventDefault()	Prevents the browser from executing the default action.
isDefaultPrevented()	Returns whether event.preventDefault() was ever called on this event object.

stopPropagation()	Stops the bubbling of an event to parent elements, preventing any parent handlers from being notified of the event.
isPropagationStopped()	Returns whether event.stopPropagation() was ever called on this event object.
stopImmediatePropagation()	Stops the rest of the handlers from being executed.
isImmediatePropagationStopped()	Returns whether event.stopImmediatePropagation() was ever called on this event object.

Event Manipulation Methods:

Following table lists down important event-related methods:

Method	Description
bind(type, [data], fn)	Binds a handler to one or more events (like click) for each matched element. Can also bind custom events.
die(type, fn)	This does the opposite of live, it removes a bound live event.
hover(over, out)	Simulates hovering for example moving the mouse on, and off, an object.
live(type, fn)	Binds a handler to an event (like click) for all current - and future - matched element. Can also bind custom events.
one(type, [data], fn)	Binds a handler to one or more events to be executed once for each matched element.
ready(fn)	Binds a function to be executed whenever the DOM is ready to be traversed and manipulated.
toggle(fn, fn2, fn3,...)	Toggle among two or more function calls every other click.
trigger(event, [data])	Trigger an event on every matched element.
triggerHandler(event, [data])	Triggers all bound event handlers on an element .
unbind([type], [fn])	This does the opposite of bind, it removes bound events from each of the matched elements.

Event Helper Methods:

jQuery also provides a set of event helper functions which can be used either to trigger an event to bind any event types mentioned above.

Trigger Methods:

Following is an example which would triggers the blur event on all paragraphs:

```
$("p").blur();
```

Binding Methods:

Following is an example which would bind a **click** event on all the :

```
$("div").click( function () {
   // do something here
});
```

Here is a complete list of all the support methods provided by jQuery:

Method	Description
blur( )	Triggers the blur event of each matched element.
blur(fn)	Bind a function to the blur event of each matched element.
change( )	Triggers the change event of each matched element.
change(fn)	Binds a function to the change event of each matched element.
click( )	Triggers the click event of each matched element.
click(fn)	Binds a function to the click event of each matched element.
dblclick( )	Triggers the dblclick event of each matched element.
dblclick(fn)	Binds a function to the dblclick event of each matched element.
error( )	Triggers the error event of each matched element.
error(fn)	Binds a function to the error event of each matched element.
focus( )	Triggers the focus event of each matched element.
focus(fn)	Binds a function to the focus event of each matched element.
keydown( )	Triggers the keydown event of each matched element.
keydown(fn)	Bind a function to the keydown event of each matched element.
keypress( )	Triggers the keypress event of each matched element.
keypress(fn)	Binds a function to the keypress event of each matched element.
keyup( )	Triggers the keyup event of each matched element.
keyup(fn)	Bind a function to the keyup event of each matched element.
load(fn)	Binds a function to the load event of each matched element.
mousedown(fn)	Binds a function to the mousedown event of each matched element.
mouseenter(fn)	Bind a function to the mouseenter event of each matched

	element.
mouseleave(fn)	Bind a function to the mouseleave event of each matched element.
mousemove(fn)	Bind a function to the mousemove event of each matched element.
mouseout(fn)	Bind a function to the mouseout event of each matched element.
mouseover(fn)	Bind a function to the mouseover event of each matched element.
mouseup(fn)	Bind a function to the mouseup event of each matched element.
resize(fn)	Bind a function to the resize event of each matched element.
scroll(fn)	Bind a function to the scroll event of each matched element.
select( )	Trigger the select event of each matched element.
select(fn)	Bind a function to the select event of each matched element.
submit( )	Trigger the submit event of each matched element.
submit(fn)	Bind a function to the submit event of each matched element.
unload(fn)	Binds a function to the unload event of each matched element.

jQuery - AJAX

AJAX is an acronym standing for Asynchronous JavaScript and XML and this technology help us to load data from the server without a browser page refresh.

If you are new with AJAX, I would recommend you go through our Ajax Tutorial before proceeding further.

JQuery is a great tool which provides a rich set of AJAX methods to develope next generation web application.

Loading simple data:

This is very easy to load any static or dynamic data using JQuery AJAX. JQuery provides **load()**method to do the job:

Syntax:

Here is the simple syntax for **load()** method:

```
[selector].load( URL, [data], [callback] );
```

Here is the description of all the parameters:

- **URL:** The URL of the server-side resource to which the request is sent. It could be a CGI, ASP, JSP, or PHP script which generates data dynamically or out of a database.
- **data:** This optional parameter represents an object whose properties are serialized into properly encoded parameters to be passed to the request. If specified, the request is made using the **POST** method. If omitted, the **GET** method is used.

- **callback:** A callback function invoked after the response data has been loaded into the elements of the matched set. The first parameter passed to this function is the response text recieved from the server and second parameter is the status code.

Example:

Consider the following HTML file with a small JQuery coding:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">
   $(document).ready(function() {
      $("#driver").click(function(event){
          $('#stage').load('/jquery/result.html');
      });
   });
   </script>
</head>
<body>
   <p>Click on the button to load result.html file:</p>
   <div id="stage" style="background-color:blue;">
          STAGE
   </div>
   <input type="button" id="driver" value="Load Data" />
</body>
</html>
```

Here **load()** initiates an Ajax request to the specified URL **/jquery/result.html** file. After loading this file, all the content would be populated inside tagged with ID *stage*. Assuming, our /jquery/result.html file has just one HTML line:

```
   <h1>THIS IS RESULT...</h1>
```

Getting JSON data:

There would be a situation when server would return JSON string against your request. JQuery utility function **getJSON()** parses the returned JSON string and makes the resulting string available to the callback function as first parameter to take further action.

Syntax:

Here is the simple syntax for **getJSON()** method:

```
[selector].getJSON( URL, [data], [callback] );
```

Here is the description of all the parameters:

- **URL:** The URL of the server-side resource contacted via the GET method.
- **data:** An object whose properties serve as the name/value pairs used to construct a query string to be appended to the URL, or a preformatted and encoded query string.
- **callback:** A function invoked when the request completes. The data value resulting from digesting the response body as a JSON string is passed as the first parameter to this callback, and the status as the second.

Example:

Consider the following HTML file with a small JQuery coding:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">
   $(document).ready(function() {
      $("#driver").click(function(event){
          $.getJSON('/jquery/result.json', function(jd) {
             $('#stage').html('<p> Name: ' + jd.name + '</p>');
             $('#stage').append('<p>Age : ' + jd.age+ '</p>');
             $('#stage').append('<p> Sex: ' + jd.sex+ '</p>');
          });
      });
   });
   </script>
</head>
<body>
   <p>Click on the button to load result.html file:</p>
   <div id="stage" style="background-color:blue;">
          STAGE
   </div>
   <input type="button" id="driver" value="Load Data" />
</body>
</html>
```

Here JQuery utility method **getJSON()** initiates an Ajax request to the specified URL**/jquery/result.json** file. After loading this file, all the content would be passed to the callback function which finally would be populated inside tagged with ID *stage*. Assuming, our /jquery/result.json file has following json formatted content:

```
{
"name": "Zara Ali",
"age" : "67",
"sex": "female"
}
```

When you click the given button, then result.json file gets loaded.

Passing data to the Server:

Many times you collect input from the user and you pass that input to the server for further processing. JQuery AJAX made it easy enough to pass collected data to the server using **data**parameter of any available Ajax method.

Example:

This example demonstrate how can pass user input to a web server script which would send the same result back and we would print it:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">
```

```
   $(document).ready(function() {
      $("#driver").click(function(event){
          var name = $("#name").val();
          $("#stage").load('/jquery/result.php', {"name":name} );
      });
   });
   </script>
</head>
<body>
   <p>Enter your name and click on the button:</p>
   <input type="input" id="name" size="40" /><br />
   <div id="stage" style="background-color:blue;">
          STAGE
   </div>
   <input type="button" id="driver" value="Show Result" />
</body>
</html>
```

Here is the code written in **result.php** script:

```
<?php
if( $_REQUEST["name"] )
{
   $name = $_REQUEST['name'];
   echo "Welcome ". $name;
}
?>
```

Now you can enter any text in the given input box and then click "Show Result" button to see what you have entered in the input box. To understand it in better way you can Try it yourself.

JQuery AJAX Methods:

You have seen basic concept of AJAX using JQuery. Following table lists down all important JQuery AJAX methods which you can use based your programming need:

Methods and Description
jQuery.ajax(options) Load a remote page using an HTTP request.
jQuery.ajaxSetup(options) Setup global settings for AJAX requests.
jQuery.get(url, [data], [callback], [type]) Load a remote page using an HTTP GET request.
jQuery.getJSON(url, [data], [callback]) Load JSON data using an HTTP GET request.
jQuery.getScript(url, [callback]) Loads and executes a JavaScript file using an HTTP GET request.
jQuery.post(url, [data], [callback], [type]) Load a remote page using an HTTP POST request.
load(url, [data], [callback]) Load HTML from a remote file and inject it into the DOM.
serialize( ) Serializes a set of input elements into a string of data.

serializeArray( ) Serializes all forms and form elements like the .serialize() method but returns a JSON data structure for you to work with.

JQuery AJAX Events:

You can call various JQuery methods during the life cycle of AJAX call progress. Based on different events/stages following methods are available:

You can go through all the AJAX Events.

Methods and Description
ajaxComplete(callback) Attach a function to be executed whenever an AJAX request completes.
ajaxStart(callback) Attach a function to be executed whenever an AJAX request begins and there is none already active.
ajaxError(callback) Attach a function to be executed whenever an AJAX request fails.
ajaxSend(callback) Attach a function to be executed before an AJAX request is sent.
ajaxStop(callback) Attach a function to be executed whenever all AJAX requests have ended.
ajaxSuccess(callback) Attach a function to be executed whenever an AJAX request completes successfully.

jQuery - Effects

jQuery provides a trivially simple interface for doing various kind of amazing effects. jQuery methods allow us to quickly apply commonly used effects with a minimum configuration.

This tutorial covers all the important jQuery methods to create visual effects.

Showing and Hiding elements:

The commands for showing and hiding elements are pretty much what we would expect: **show()**to show the elements in a wrapped set and **hide()** to hide them.

Syntax:

Here is the simple syntax for **show()** method:

```
[selector].show( speed, [callback] );
```

Here is the description of all the parameters:

- **speed:** A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).

- **callback:** This optional parameter represents a function to be executed whenever the animation completes; executes once for each element animated against.

Following is the simple syntax for **hide()** method:

```
[selector].hide( speed, [callback] );
```

Here is the description of all the parameters:

- **speed:** A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).
- **callback:** This optional parameter represents a function to be executed whenever the animation completes; executes once for each element animated against.

Example:

Consider the following HTML file with a small JQuery coding:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {

     $("#show").click(function () {
        $(".mydiv").show( 1000 );
     });

     $("#hide").click(function () {
        $(".mydiv").hide( 1000 );
     });

   });

   </script>
   <style>
   .mydiv{ margin:10px;padding:12px;
      border:2px solid #666;
      width:100px;
      height:100px;
    }
  </style>
</head>
<body>
   <div class="mydiv">
      This is  SQUAR
   </div>

   <input id="hide" type="button" value="Hide" />
   <input id="show" type="button" value="Show" />

</body>
</html>
```

Toggling the elements:

jQuery provides methods to toggle the display state of elements between revealed or hidden. If the element is initially displayed, it will be hidden; if hidden, it will be shown.

Syntax:

Here is the simple syntax for one of the **toggle()** methods:

```
[selector]..toggle([speed][, callback]);
```

Here is the description of all the parameters:

- **speed:** A string representing one of the three predefined speeds ("slow", "normal", or "fast") or the number of milliseconds to run the animation (e.g. 1000).
- **callback:** This optional parameter represents a function to be executed whenever the animation completes; executes once for each element animated against.

Example:

We can animate any element, such as a simple containing an image:

```
<html>
<head>
<title>the title</title>
   <script type="text/javascript"
   src="/jquery/jquery-1.3.2.min.js"></script>
   <script type="text/javascript" language="javascript">

   $(document).ready(function() {
      $(".clickme").click(function(event){
          $(".target").toggle('slow', function(){
             $(".log").text('Transition Complete');
          });
      });

   });
   </script>
   <style>
   .clickme{ margin:10px;padding:12px;
      border:2px solid #666;
      width:100px;
      height:50px;
    }
   </style>
</head>
<body>
   <div class="content">
      <div class="clickme">Click Me</div>
      <div class="target">
         <img src="/images/jquery.jpg" alt="jQuery" />
      </div>
      <div class="log"></div>
</body>
</html>
```

JQuery Effect Methods:

You have seen basic concept of jQuery Effects. Following table lists down all the important methods to create different kind of effects:

Methods and Description
animate(params, [duration, easing, callback]) A function for making custom animations.
fadeIn(speed, [callback]) Fade in all matched elements by adjusting their opacity and firing an optional callback after completion.
fadeOut(speed, [callback]) Fade out all matched elements by adjusting their opacity to 0, then setting display to "none" and firing an optional callback after completion.
fadeTo(speed, opacity, callback) Fade the opacity of all matched elements to a specified opacity and firing an optional callback after completion.
hide( ) Hides each of the set of matched elements if they are shown.
hide(speed, [callback]) Hide all matched elements using a graceful animation and firing an optional callback after completion.
show( ) Displays each of the set of matched elements if they are hidden.
show(speed, [callback]) Show all matched elements using a graceful animation and firing an optional callback after completion.
slideDown(speed, [callback]) Reveal all matched elements by adjusting their height and firing an optional callback after completion.
slideToggle(speed, [callback]) Toggle the visibility of all matched elements by adjusting their height and firing an optional callback after completion.
slideUp(speed, [callback]) Hide all matched elements by adjusting their height and firing an optional callback after completion.
stop([clearQueue, gotoEnd]) Stops all the currently running animations on all the specified elements.
toggle( ) Toggle displaying each of the set of matched elements.
toggle(speed, [callback]) Toggle displaying each of the set of matched elements using a graceful animation and firing an optional callback after completion.
toggle(switch) Toggle displaying each of the set of matched elements based upon the switch (true shows all elements, false hides all elements).
jQuery.fx.off Globally disable all animations.

UI Library Based Effects:

To use these effects you would have to download jQuery UI Library **jquery-ui-1.7.2.custom.min.js** or latest version of this UI library from jQuery UI Library.

After extracting jquery-ui-1.7.2.custom.min.js file from the download, you would include this file in similar way as you include core jQuery Library file.

Methods and Description
Blind Blinds the element away or shows it by blinding it in.
Bounce Bounces the element vertically or horizontally n-times.
Clip Clips the element on or off, vertically or horizontally.
Drop Drops the element away or shows it by dropping it in.
Explode Explodes the element into multiple pieces.
Fold Folds the element like a piece of paper.
Highlight Highlights the background with a defined color.
Puff Scale and fade out animations create the puff effect.
Pulsate Pulsates the opacity of the element multiple times.
Scale Shrink or grow an element by a percentage factor.
Shake Shakes the element vertically or horizontally n-times.
Size Resize an element to a specified width and height.
Slide Slides the element out of the viewport.
Transfer Transfers the outline of an element to another.