JavaScript Standards and Guidelines

## Getting Started:

## Scripting Benefits

* The purpose of ‘scripting’ is to **allow user interaction** within an otherwise **static interface**.
* JavaScript provides us with just that. Using **unobtrusive calls and smart degradation**, JavaScript can be used to improve and favor user experience.
* JQuery and other frameworks **make selecting nodes and DOM manipulation easy**.
* JavaScript not only provides access to HTML objects, it also gives access to browser and platform-specific objects like browser plug-ins (e.g. Adobe Acrobat, Media Player).
* JavaScript will **allow client-side user form validation**. As a result the user of website gets faster feedback than waiting for response from the server.
* While supporting all of the above, JavaScript can have adverse effects if it is not used carefully following simple best practices and guidelines.

**Usage of JavaScript:**

## Use External JavaScript file

* Do not **use inline or internal scripts**. This **will increase the page size** and the script **cannot** be **reused**.
* Placing inline scripts also **affects maintainability** factor. Any change has to be searched upon and implemented in every single file it is used.
* Obtrusive elements are **scripts** that are **inlaid** into the **xhtml** source. They make the source more difficult to read **and often lead to validation errors**.
* With obtrusive JavaScript, you also **sacrifice** the **advantages** of **separating behavior from structure**. An example of obtrusive JavaScript is the **document.write() method**. This method requires you to place a behavioral script tag inside your structural xhtml. This will confuse most, if not all, screen readers and cause them to **skip the inlaid content altogether**.

Example of Inline Java Script

*<body onload=”document.body.style.backgroundColor=’blue’”></body>*

*document.write(“Corporate Facilities<br />1234 Ln<br />Minneapolis*

* There are several break tags (<br />) inside of our script tag which, in an xhtml document, is not recommended. This will cause the page to fail validation and render incorrectly on some browsers.
* **Unobtrusive JavaScript** suggests **placing all your scripts in an external JavaScript file and accessing your xhtml with the Document Object Model or dom**.
* Now when a screen reader or disabled device comes across this block of code it will understand the content and relay it to the user uninterrupted. The second idea of unobtrusive JavaScript is to keep your source outside the structural xhtml and in an external file. Doing this and relying on the dom in JavaScript allows pages to degrade gracefully.
* Place all **the** JavaScript functions in a separate JavaScript file and include it in the **html page at the bottom of the body section** because script block parallel downloads.

Example of an external javascript file included in the html file

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

## Always Use 'var'

* **Variables** have scope, either **global scope or function scope** and using the **'var' keyword is vital**. When declaring a variable for use either as a global variable or as a function-level variable, always prefix the declaration with the 'var' keyword.

Problem Caused by Not Using Var

*var i=0; // This is right- creates a global variable*

*function test() {*

*for (i=0; i<10; i++) {*

*alert("Welcome to XBank!");*

*} }*

*test();*

*alert(i); // Value of the global variable i is now 10!*

* Since the variable i inside the function was not declared as a function-level variable by using the 'var' keyword, it references the global variable in this example. It is **a good idea to always declare global variables using 'var', but it is vital to declare function-scoped variables using 'var'**.  
    
   Fixed Function

*function test() {*

*var i=0;*

*for (i=0; i<10; i++) {*

*alert("Welcome to XBank!");*

*}}*

or

Fixed Function

*function test() {*

*for (var i=0; i<10; i++) {*

*alert("Welcome to XBank!");*

*} }*

# Use Feature detection instead of Brower detection

* It is **against** the spirit of **web standards**. We should write code that adheres to established standards and software in charge of displaying our code.
* Browser detection relies on the **browser user-agent string** which is easily **spoofed**.
* Perform **Capability testing** instead which **tests if the environment** we are in is capable of what we want to do. Feature detecting **seeks to match an attempt to execute as script** (or a part of a script) with the execution environment by **seeking to test features of that environment** where the results of the test have a direct one-to-one relationship with the features **that need to be supported in the environment** for the code to successfully execute.

Example of Browser detection using JavaScript. Navigator is a JavaScript object which is used to detect the browser.

*txt = "<p>Browser CodeName: " + navigator.appCodeName + "</p>";*

*txt+= "<p>Browser Name: " + navigator.appName + "</p>";*

*txt+= "<p>Browser Version: " + navigator.appVersion + "</p>";*

*txt+= "<p>Cookies Enabled: " + navigator.cookieEnabled + "</p>";*

*txt+= "<p>Platform: " + navigator.platform + "</p>";*

*txt+= "<p>User-agent header: " + navigator.userAgent + "</p>";*

Feature Detection

*function copyToClip(myString){*

*if((typeof clipboardData != 'undefined')&&*

*(clipboardData.setData)){*

*clipboardData.setData("text",myString);*

*}*

*}*

*if (document.getElementById) {*

*var element = document.getElementById('MyId');*

*}*

*else {*

*alert('Your browser lacks the capabilities required to run this script!');*

*}*

Here the clipboardData.setData method is called have a **direct one-to-one relationship with the browser's support for the feature**. It is not necessary to be interested in whether the browser is the expected windows IE that is known to implement the feature, or whether it is some other browser that has decided to copy IE's implementation and provide the feature itself.

The above feature detecting tests are done **using two operations**. The first employs the **typeof operator**, which **returns a string** depending on the **type of its operand**. That string is one of **"undefined", "object", "function", "boolean" "string" and "number"** and the test compares the returned string with the string "undefined". The clipboardData object is not used unless typeof does not return "undefined".

The second test is **a type-converting test**. The logical AND (&&) operator internally converts its operands to boolean in order to make its decision about what value it will return. If clipboardData.setData is undefined it will type-convert to boolean false, while if it is an object or a function the result of the conversion will be boolean true.

**Use Square Bracket Notation**

* It is considered as best practice to use **square bracket notation instead of dot notation** while **accessing object properties at run time.**
* In dot notation, generally the property name is hardcoded and cannot be changed at run time
* This can be better achieved by square Bracket notation since the property name is string which can be variable or hardcoded or even a function that can return value.

Dot Notation Example- *MyObject.property*

Square Bracket Notation Example- *MyObject["property"]*

With **dot notation**, the **property name is hard-coded** and **cannot** be **changed at run-time**. With bracket notation, the **property name is a string which is evaluated to resolve the property name.** The string can be hard-coded, or a variable, or even a function call which returns a string property name.

If a property name is being generated at run-time, the bracket notation is required. For example, if you have properties "value1", "value2", and "value3", and want to access the property using a variable i=2:

This sample will not work:

*MyObject.value+i;*

Correct Sample:

*MyObject[“value”+i]*

**Reference Forms and Form Elements Correctly**

* All forms in HTML should **have name attribute**
* It is generally considered as **bad practices to refer forms using indexes like document.forms[0]**

Incorrect Sample:

*document.forms[0].elements[1].value*

*document.formname.inputname.value*

Correct Sample

*document.forms[“formname”].elements[“inputname”].value*

* It is also considered as **best practice** to **store the form object in variable** and there by accessing the **form elements using that variable** which avoids further confusions.

Sample:

*var formElements = document.forms["mainForm"].elements;*

*formElements["input1"].value="a";*

*formElements["input2"].value="b";*

**Avoid eval()**

* The “eval” function gives us **access to JavaScript’s compiler**.
* It **decrease script’s** **performance substantially** and also **poses a huge security risk** because it grants far too **much power** to the **passed in text**.

**Double and Single Quotes**

* Double quotes are generally used for HTML attributes

Sample:

<img src=”sample.gif” width=”300” height=”200” alt=”samplecontent” title=”samplecontent” />

* Single quotes are generally used for JavaScript string literals

Sample:

document.write(‘<p>’);

* use the escape key(“\”) for nested quotes

document.write(‘Don\’t put sample content’);

**Semicolon**

* JavaScript allows any expression to be used as a statement and uses semi-colons to mark the end of a statement. However, it attempts to make this optional with "semi-colon insertion", which can mask some errors and will also cause JS aggregation to fail. All statements should be followed by ; except for the following*: for, function, if, switch, try, while*.

**NOSCRIPT tag**

* It is always considered as best practices to use NOSCRIPT tag for providing alternate text for JavaScript disabled browsers
* HTML within the NOSCRIPT tag is rendered if the browser fails to recognize JavaScript files.

Sample:

*<script type="text/JavaScript">*

*<!--*

*document.write('<P>Last modified: ' + document.lastModified + '<\/P>);*

*//-->*

*</script>*

*<nocript>*

*<P>Last modified: February 1999</P>*

*</noscript>*

**Server side Includes**

* The **web page should be like a skeleton** and building block such **as header, footer, left navigation, top navigation, right navigation** and content should be included through **server side includes**.
* This will increase the **maintainability**. If any changes needs to be done to any of the building block, the particular include file will be updated in just one place and all the pages will be updated automatically.

**Object Oriented JavaScript**

* In JavaScript a Object is a construct with properties that can be either variables or another objects.
* Objects may be pre-defined and can be created by user.

## Object Properties

**Syntax**

*objectName.propertyName*

## Object Methods

**Syntax**

*object.methodname = function\_name;*

## Creating Objects

1. Creating an object with Object Literal

**Syntax- *var obj = {property\_1: value1, property\_2: value2,….. }***

1. Creating an object with Constructor

**Syntax-**

***function payments(card,cash,cheque){***

***this.card = card;***

***this.cash = cash;***

***this.cheque = cheque;***

***}***

**New object is created using new keyword**

***Var myPayments = new payments(“Master card”, “”,””);***

|  |  |  |  |
| --- | --- | --- | --- |
| **Hierarchy Objects** | | | |
| **Object** | **Properties** | **Methods** | **Event Handlers** |
| Window | defaultStatus frames opener parent scroll self status top window | alert blur close confirm focus open prompt clearTimeout setTimeout | onLoad onUnload onBlur onFocus |
| Frame | defaultStatus frames opener parent scroll self status top window | alert blur close confirm focus open prompt clearTimeout setTimeout | none (The onLoad and onUnload event handlers belong to the Window object) |
| Location | hash host hostname href pathname por protocol search | reload replace | none |
| History | length forward go | back | none |
| Navigator | appCodeName appName appVersion mimeTypes plugins userAgent | javaEnabled | none |
| document | alinkColor anchors applets area bgColor cookie fgColor forms images lastModified linkColor links location referrer title vlinkColor | clear close open write writeln | none (the onLoad and onUnload event handlers belong to the Window object. |
| image | border complete height hspace lowsrc name src vspace width | none | none |
| form | action elements encoding FileUpload method name target | submit reset | onSubmit onReset |
| text | defaultValue name type value | focus blur select | onBlur onCharge onFocus onSelect |

|  |  |  |  |
| --- | --- | --- | --- |
| **Built-in Objects** | | | |
| Array | length | join reverse sort xx | none |
| Date | none | getDate getDay getHours getMinutes getMonth getSeconds getTime getTimeZoneoffset getYear parse prototype setDate setHours setMinutes setMonth setSeconds setTime setYear toGMTString toLocaleString UTC | none |
| String | length prototype | anchor big blink bold charAt fixed fontColor fontSize indexOf italics lastIndexOf link small split strike sub substring sup toLowerCase toUpperCase | Window |

**Recommendations**

1. Keyword *this* should be used instead of a name and mapping the name as the object  
   example: <input type=’text’ name=’portal’ onclick=”validate(this)” />.
2. Using Javascript in an object orientedway, provides better **reusability of the code** and enables your **objects to be better organized**.
3. When writing components **use objects and object hierarchies** to organize related objects and prevent naming collision.

Example:

*function Cart() {*

*this.items = [];*

*}*

*function Item (id,name,desc,price)) {*

*this.id = id;*

*this.name = name;*

*this.desc = desc;*

*this.price = price;*

*}*

*// Creating an instance of the cart and adding an item to the cart*

*var cart = new Cart();*

*cart.items.push(new Item("id-1","book","something that you read",5));*

*cart.items.push(new Item("id-1","Pen", "Something you write with", 3);*

*var total;*

*while (var l; l < cart.items.length; l++) {*

*total = total + cart.items[l].price;*

*}*

1. Use Prototype Property

Example:

*function Cart() {*

*this.items = [ ];*

*}*

*function Item (id,name,desc,price)) {*

*this.id = id;*

*this.name = name;*

*this.desc = desc;*

*this.price = price;*

*}*

*// SmartCart extends the Cart object inheriting its properties and adds a total property*

*function SmartCart() {*

*this.total = 0;*

*}*

*SmartCart.prototype = new Cart();*

# JSON & XML with JavaScript

* **JSON** (JavaScript Object Notation) and **XML** (Extensible Mark-up Language) are the two popular ways we can access data.
* For **JSON**, we can access data using **objects** via. JavaScript. JSON is a lot leaner than XML for Ajax request. JSON can be parsed more efficiently because it can be parsed as JavaScript, which the built-in eval() function
* For **XML**, we can access data using **nodes** via. JavaScript

JSON is a subset of the object literal notation of JavaScript.

Example:

JSON object.

*{*

*"name": "xx (\"yy\") zz",*

*"format": {*

*"type": "rect",*

*"width": 1920,*

*"height": 1080,*

*"interlace": false,*

*"frame rate": 24*

*}*

*}*

*var myJSONObject = {"bindings": [*

*{"ircEvent": "PRIVMSG", "method": "newURI", "regex": "^http://.\*"},*

*{"ircEvent": "PRIVMSG", "method": "deleteURI", "regex": "^delete.\*"},*

*{"ircEvent": "PRIVMSG", "method": "randomURI", "regex": "^random.\*"}*

*]*

*};*

In the above example, the object is created containing a single member "bindings", which contains an array containing three objects, each containing "ircEvent", "method", and "regex" members.

Members can be retrieved using dot or subscript operators.

*myJSONObject.bindings[0].method // "newURI"*

To convert a JSON text into an object, you can use the eval() function.

A JSON parser will recognize only JSON text, rejecting all scripts.

*myData = JSON.parse(text, function (key, value) {*

*var type;*

*if (value && typeof value === 'object') {*

*type = value.type;*

*if (typeof type === 'string' && typeof window[type] === 'function') {*

*return new (window[type])(value);*

*}*

*}*

*return value;*

*});*

## XML using JavaScript

The XMLHttpRequest object is used to exchange data with a server behind the scenes.This object can

* Update a web page without reloading the page
* Request data from a server after the page has loaded
* Receive data from a server after the page has loaded
* Send data to a server in the background

Syntax:

// All modern browsers (IE7+, Firefox, Chrome, Safari) have a built-in XMLHttpRequest object.

*xmlhttp=new XMLHttpRequest();*

//Old versions of Internet Explorer (IE5 and IE6) use an ActiveX Object:

*xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");*

The following code fragment parses an XML document into an XML DOM object:

Example:

*if (window.XMLHttpRequest)*

*{// code for IE7+, Firefox, Chrome, Opera, Safari*

*xmlhttp=new XMLHttpRequest();*

*}*

*else*

*{// code for IE6, IE5*

*xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");*

*}*

*xmlhttp.open("GET","creditcards.xml",false);*

*xmlhttp.send();*

*xmlDoc=xmlhttp.responseXML;*

**Usage of Framework:**

**window.onload Vs document. ready:**

* The **ready** event occurs after the **HTML document** has been **loaded** even if all the graphics haven’t loaded yet. If you want to hook up events for certain elements before the window loads, then $(document).ready is the right place, while **window load** event executes a bit later when the **complete page is fully loaded, including all frames, objects and images**.
* The script can be run as soon as the DOM hierarchy has been fully constructed where as the ready event can be executed once the DOM ready.
* The **ready** method is generally incompatible with **<body onload=””>** , if load must be used either use onload event or ready event to attach load event handlers to the window or more specific items like images.

Sample Code:

JavaScript:

*window.onload=function(){*

*//Our Code Here  
  }*

JQuery:

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

*//Our Code Here*

*});*

**Callback functions**:

* A **callback** is a function that **is passed as an argument to another function** and is **executed after its parent function has completed**.
* The special thing about a **callback** is that functions that appear after the "parent" can execute before the **callback** executes.
* In JavaScript, with **andThen** method, it becomes very easy to create an object that allows several other objects to register callbacks.

*var a=new Alerter('Hello, world!');*

*var b=new Alerter('Bye, world!');*

*function Manager() {*

*this.callback=function () {}; // do nothing*

*this.registerCallback=function(callbackFunction) {*

*this.callback=(this.callback).andThen(callbackFunction);*

*}*

*}*

*var manager=new Manager();*

*manager.registerCallback(a.invoke);*

*manager.registerCallback(b.invoke);*

*manager.callback();*

**Recommendations**

* JavaScript statements are executed line by line. However, with animations, the next line of code can be run even though the animation is not finished. This can create errors. To prevent this type of issues, we can create a **callback** function.

**Ways of accessing callbacks**:

*function Manager() {*

*this.callback=function () {}; // do nothing*

*this.registerCallback=function(callbackFunction) {*

*this.callback=(this.callback).****andThen****(callbackFunction);*

*}*

*}*

*JQuery:*

*$('#myButton').****bind****('click', function() {*

*// our code to handle the click event goes*

*// here, in the callback function*

*});*

**Event Delegation**:

* The general idea with **event delegation** is to have just one (or a few) events on elements far higher up in the HTML hierarchy, and from there, trace on what target element the event actually occurred, and then take appropriate action.
* The major benefit of event delegation was **bind-once event handling and it is significantly faster than event handler as the handler is bound to just one element.**
* **JQuery** makes event delegation quick and easy and we can achieve this by three ways .live() , .delegate() and do it our self.
* **.live() is** simple way to implement event delegation and is suitable for simple scenarios.
* .**delegate**() is more focused way of doing event delegation and it makes it easy to specify which element we want to delegate from and which element to filter the event .
* The last technique is more advanced use-cases where you want to have more flexibility than .**delegate**() and .**live**().
* Using a method called .**closest**() we are going to look at the event. **target** (which element the event happened on) and see if it or any of its parents are the element we want to filter the event on.
* Event capturing phase is when event is being sent from the container all the way down to the target element. Event bubbling refers to the way events are propagated upward to its ancestors after it reaches its target element.

Sample Code:

*$('tr').live('click', function(event) {*

*// this == tr element*

*});*

*$('table').delegate('tr', 'click', function(event) {*

*// this == tr element*

*});*

*$('table').bind('click', function(event) {*

*// this == table element*

*var $tr = $(event.target).closest('tr');*

*});*

**Event Bubbling and Target Element:**

Event Delegation makes use of two features of JavaScript events and they are Event Bubbling and target Element.

**Event Bubbling:**

* When an event is triggered on an element, for example a mouse click on a button, the same event is also triggered on all of that element’s ancestors. This process is known as event bubbling.
* The event bubbles up from the originating element to the top of the DOM tree.

.live( eventType, handler ) - Attach a handler to the event for all elements which match the current selector, now and in the future.

*$('.hoverme').live('mouseover mouseout', function(event) {*

*if (event.type == 'mouseover') {*

*// do something on mouseover*

*} else {*

*// do something on mouseout*

*}*

*});*

**Target Element:**

* The target element of any event is the originating element, the button in our example, and is stored in a property of the event object.
* **this** keyword can be used in JavaScript whereas in JQuery **event. target** was used.
* Note also that **'this'** will also work, but that it is not a **JQuery** object, so if you wish to use a **JQuery** function on it then you must refer to it as '**$(this)**'.

Sample Code:

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

*$("a").click(function(event) {*

*alert(event.target.id); //it alerts the triggered event id*

*});*

*});*

**Chaining**:

* One of the coolest ability in JQuery is to chain method calls together. end() method can be used to end the most recent operation and returns to its previous state. Chain methods should be used properly.

Sample Code:

*$('myDiv').removeClass('off').addClass('on'); //to switch between the class*

**Input field Masking**:

* It is a technique used to the get the particular information from the user in a **particular** **format**.
* A mask is defined by a format made up of **mask literals and mask definitions**.
* By masking input of a particular textbox, you can change its **behavior** so that it accepts input only according to specified format.
* In JQuery, Plug-in will be available for masking of input fields like date, credit card ,SSN etc.
* We can also pass a **placeholder** as an argument to the maskedInput method.

Sample Code:

*jQuery(function($){  
    $("#date").mask("99/99/9999");  
   $("#phone").mask("(999) 999-9999",{placeholder:" "});  
    $("#tin").mask("99-9999999");  
   $("#ssn").mask("999-99-9999");  
});*

## Regular Expressions

* Regular expressions are patterns used to match character combinations in strings.
* In JavaScript, regular expressions are also objects.
* Regular expression can be constructed in two ways:
  + Using a **regular expression literal**, this provides compilation of the regular expression when the script is evaluated.

Example:

*var reg=/ab+c/; //Use this for better performance*

* + Calling the constructor function of the **RegExp** object,

Example:

*var re=new RegExp(“ab+c”);*

* Use regular expression for validation.
* match selectors can be used for regular expressions in JQuery and also there is some plug-ins available for regular expressions.

Example:

*var rege = /^([A-Za-z0-9\_\-\.])+\@([A-Za-z0-9\_\-\.])+\.([A-Za-z]{2,4})$/;*

*//Javascript*

$('p:regexp(/\\d+/)') //JQuery

*if(rege.test($('#email').val())){ //do something }*

Example of Username and Password field validations

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

*$.validator.addMethod("username",function(value,element)*

*{*

*return this.optional(element) || /^[a-zA-Z0-9.\_-]{3,16}$/i.test(value);*

*},"Username are 3-15 characters");*

*$.validator.addMethod("password",function(value,element)*

*{*

*return this.optional(element) || /^[A-Za-z0-9!@#$%^&\*()\_]{6,16}$/i.test(value);*

*},"Passwords are 6-16 characters");*

*$("#signup").validate({*

*rules: {*

*username: "required username",*

*password: "required password",*

*},*

*});*

*});*

**Form Validation using JQuery**:

* JQuery validation is very easy to use and has the good look and feel for the users in the webpage forms and it’s a powerful client side validation.
* Even though validation plug-in is available, it is better to customize validate using regEx and use validation CSS from plug-in.
* **.validate()** method is used for validation using JQuery plug-in.
* The **validate method returns a validator object that** has a few public methods that you can use **trigger validation programmatically or change the contents of the form**.

Sample Code:

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

*$("#commentForm").validate(); //commentForm is Form Name*

*});*

**JQuery :odd and :even methods**

* **jquery :odd and :even methods should be used** to select odd and even child elements rather **than traversing the whole child elements using a for loop** and based on the index value using if else condition to carry out the functionality.
* jQuery :odd and :even methods will achieve this in just one line of code.

Example:

Apply background colors for alternative rows in a table.

Normal JavaScript:

*rowArray=tableobj.getElementsByTagName(“tr”);*

*for (i=0;i<rowArray.length;i++)*

*{ If(i%2==0)*

*rowArray[i].className=”evenRow”;*

*else*

*rowArray[i].className=”oddRow”;}*

jQuery :

*$(#tableID tr:odd). addClass('oddRow’);*

*$(#tableID tr:even). addClass('evenRow’);*

## Wise Use of Selectors

* Jquery **ID selector** is **faster than jQuery Class selector**. Instead of searching the whole document for a specific class or id it is better to search the child of a specific container in a document.

Example

*$(#parentdivid).find(Selector)*

* **Choosing between the class selector and the ID selector** has to be done appropriately. For example**, for unique occurrences** using **ID selector is much faster**. But for **generic validations** like **date masking** etc, which occurs in **multiple pages**, **class selectors** should be used and **any changes can be done in one common js file** rather than modifying the same functionality in individual js files all over again.

*$(".startDate"). bind("click",function(){*

*$(this).show();*

*});*

* #id selectors- Matches single element with the given id attribute.

Example

*$('#uniqueValue)*

This selector selects element with id *uniqueValue*.

* .Class selector- Matches all elements with the given name.

Example:

*$(".class-a")*

This selector selects all paragraph (p) elements.

*<p id="SelectMe" style="display:none">This paragraph will be selected</p>*

*<p id="NotSelectMe">This paragraph will not be selected</p>*

**For loop Vs jQuery each()**

* In normal for loop we have to find the length of the array, and then traverse through the array. This can be simplified using each () jQuery method.

For loop

*li\_array=document.getElementBy(“elementID”).getElementsByTagName(“li”);*

*for(i=0;i<li\_array.lenght;i++) {*

*<Do something>}*

JQuery each()

*$(“ul#elementID li”).each(function(){*

*<Do something>});*

**Browser Detection using jQuery**

$.browser – used to detect the browser and $.support is also available to detect

Sample code in core jQuery

*jQuery.each(jQuery.browser, function(i, val) {*

*$("<div>" + i + " : <span>" + val + "</span>")*

*.appendTo(document.body);*

*});*

Result

*mozilla : true version : 1.9.2.13*

*$.browser.version = 1.9.2.13*

*$.support.boxModel = true*

### Recommendations:

* It is best practice to test the detection support using $.support before using $.browser properties
* It is always best to avoid browser-specific code entirely where possible.
* Browser detection is a wrong practice since it belongs to the category **Browser Spoofing**

**JQuery-UI**

* Jquery-UI framework can be used for all UI related functionalities. It has many inbuilt components such as **Accordion, Toggle container, Tab container etc.**
* Jquery Accordion has to be used in forms where accordion type of structure exists. The content of each tab should be loaded using an Ajax call instead of having all the tabs pre-populated in the page and keeping them hidden which will increase the page size.
* Also user should not be allowed to move to the next tab unless the user fills in the first tab. Once the first tab is completed without any errors, Next or continue button should be enables on clicking of the button an ajax call can be made to load the content of next tab. This approach will ensure the user to complete the form step by step and also the page will not be dumped with all the unwanted form fields which need not be visible the page is loaded.

Properties for Accordion UI:

**active, header, navigation, animated, alwaysOpen, event**,etc. They all have default values.

*jQuery('.firstLevel’).accordion({*

*active: 'a.selected',*

*header: 'a.head',*

*alwaysOpen: false,*

*animated: true,*

*showSpeed: "fast",*

*hideSpeed: "fast",*

*navigation: true*

*});*

Sample Code:

The markup of your accordion container needs pairs of headers and content panels:

*<div id="accordion">*

*<h3><a href="#">First header</a></h3>*

*<div>First content</div>*

*<h3><a href="#">Second header</a></h3>*

*<div>Second content</div>*

*</div>*

**JQuery accordion**:

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

*$("#accordion").accordion (); //*It will initialize an accordion effect

*});*

*$('# accordion ').accordion ({*

*autoheight: false,*

*event: click //*default value is click and it can be changed to mouseover etc.  
*});*

Usually, **to open multiple sections in an accordion,** effect can be written with a few lines of JQuery code in JQuery UI as follows:

*jQuery(document).ready(function(){*

*$('.accordion .head').click(function() {*  //.accordion - class selector also used to access

*$(this).next().toggle();*

*return false;*

*}).next().hide();*

*});*

Or animated:

*jQuery(document).ready(function(){*

*$('.accordion .head').click(function() {*

*$(this).next().toggle('slow');*

*return false;*

*}).next().hide();*

*});*

* Event Abstraction jquery should be used for all JavaScript functionalities. Avoid using normal JavaScript and use jQuery. Event handing has to be done through jQuery.

# JQuery Date picker:

* The jQuery UI Datepicker is a highly configurable plugin that adds datepicker functionality to your pages.
* The **datePicker** plugin is a popup calendar widget and then some. It allows you to add popup calendars to your forms to make it easier for users of your website to enter dates.
* By default, the **datepicker** calendar opens in a small overlay **onFocus** and closes automatically **onBlur** or when a date is selected.
* We can customize the date format and language, restrict the selectable date ranges and add in buttons and other navigation options easily by setting the corresponding attributes.

Sample Code:

*$( "#dob" ).datepicker();* //It will popup the calendar while onclick where dob was the ID of text box.

**Date Picker-UI**  
  
 *$(function() {*

*$("#datepicker").datepicker();*

*});*

*formatDate(format, date, settings)*

The format can be combinations of the following:

*d - day of month (no leading zero)*

*dd - day of month (two digit)*

*o - day of the year (no leading zeros)*

*oo - day of the year (three digit)*

*D - day name short*

*DD - day name long*

*m - month of year (no leading zero)*

*mm - month of year (two digit)*

*M - month name short*

*MM - month name long*

*y - year (two digit)*

*yy - year (four digit)*

**Datepicker** provides support for **localizing** its content to **function for various languages and date formats**. Every localization is contained within its own file with the language code appended to the name, e.g. **jquery.ui.datepicker-fr.js**

# JQuery Light box:

* Light box JS is a simple, unobtrusive script used to overlay images on the current page. It's a snap to setup and works on all modern browsers.
* It keeps users on the same page while doing another task.

Sample Code:

Include the Lightbox CSS file :

*<link rel="stylesheet" href="css/lightbox.css" type="text/css" media="screen" />*

Include the JS file:

*<script type="text/javascript" src="jquery.js"></script><script type=”text/javascript” src=”lightbox.js”></script>*

*$(function() {*

*$('#gallery a').lightBox({fixedNavigation:true});*

*});*

*Example of Calling Jquery lightbox*

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

*$(function() {*

*// Use this example, or...*

*$('a[@rel\*=lightbox]').lightBox(); // Select all links that contains lightbox in the attribute rel*

*// This, or...*

*$('#gallery a').lightBox(); // Select all links in object with gallery ID*

*// This, or...*

*$('a.lightbox').lightBox(); // Select all links with lightbox class*

*// This, or...*

*$('a').lightBox(); // Select all links in the page*

*// ... The possibility are many. Use your creative or choose one in the examples above*

*});*

*</script>*

**Jquery Effects**:

* The jQuery library provides several techniques for adding animation to a web page.
* Mostly used effect methods are animate(),fadeIn(),fadeOut(),fadeTo(),toggle().
* While using special effects it is recommended to use event.preventDefault() to not trigger the default action when the particular event was called.

Sample Code:

*$('#clickme').click(function() {*

*event.preventDefault(); //this method is used to stop the default action of the event*

*$('#book').fadeOut('slow', function() {*

*// Animation complete.*

*});*

*});*

**jQuery ajax() method Vs Normal JavaScript Ajax code**

* JQuery Ajax API is very much simpler for making an Ajax request.
* It has many inbuilt attributes and call back function which makes it simple.
* jQuery Ajax API is compatible with all browser . In normal javascript, we need to have activeXobject created and need to send request.
* Activexobject creation has to be done in different way for different browsers. We need to identify the browser and need to create ActiveXobject based on the browser.

Normal JavaScript Ajax method:

*function showHint(str)*

*{*

*var xmlhttp;*

*if (window.XMLHttpRequest)*

*{// code for IE7+, Firefox, Chrome, Opera, Safari*

*xmlhttp=new XMLHttpRequest();*

*}*

*else*

*{// code for IE6, IE5*

*xmlhttp=new ActiveXObject("Microsoft.XMLHTTP");*

*}*

*xmlhttp.onreadystatechange=function()*

*{*

*if (xmlhttp.readyState==4 && xmlhttp.status==200)*

*{*

*document.getElementById("containerID").innerHTML=xmlhttp.responseText;*

*}*

*}*

*xmlhttp.open("GET","url” ,true);*

*xmlhttp.send();*

*}*

jQuery Ajax method

*$.ajax({*

*type: "GET",*

*url: "some.php",*

*data: "name=XX&location=Boston",*

*success: function(msg){*

*$(“#container”).HTML=msg;*

*} });*

**JQuery Guidelines**

* **Always descend from an #id**

The fastest selector in jQuery is the ID selector ($('#someid')). This is because it maps directly to a native JavaScript method, getElementById().

Selecting Single Elements:

*<div id="content">*

*<form method="post" action="/">*

*<h2> Are you an Existing US Bank Customer? </h2>*

*<ul id="validity">*

*<li><input type="radio" class="on" name="customer" value="Yes" /> Yes</li>*

*<li><input type="radio" class="off" name="customer" value="No /> No</li> </ul>*

*<input class="button" id=" customerGo" type="submit" value="Go" />*

*</form>*

*</div>  
  
Selecting the button like this is slower:*

*var customer = $('#content .button'); //Instead, select the button directly:*

*var customer = $('# customerGo);*

Once we start talking about selecting multiple elements, we are really talking about DOM traversal and looping, something that is slow. To minimize the performance hit, always descend from the closest parent ID:

*var customer = $('# validity input'); // Selecting Multiple Elements*

* **Use Tags Before Classes**

The second fastest selector in jQuery is the **Tag selector** ($('head')). Again, this is because it maps to a native JavaScript method, getElementsByTagName()

*<div id="content">*

*<form method="post" action="/">*

*<h2> Are you an Existing US Bank Customer? </h2>*

*<ul id="validity">*

*<li><input type="radio" class="on" name="customer" value="Yes" /> Yes</li>*

*<li><input type="radio" class="off" name="customer" value="No /> No</li> </ul>*

*<input class="button" id=" customer" type="submit" value="Go" />*

*</form>*

*</div>*

Always **prefix a class with a tag name** (and remember to descend from an ID):

*var active\_Customer = $('#validity input.on');*

The **class selector** is among the **slowest selectors in jQuery**; in IE it loops through the entire DOM**. Avoid using it whenever possible**.

Never prefix an ID with a tag name. For example, this is slow because it will loop through all <div> elements looking for the ‘content’ ID:

*var content = $('div#content'); Similarly, it is redundant to descend from multiple IDs:*

*var customer = $('#content #validity');*

* **Use Global variables when an object is fetched more than once or twice**

Global variables need to be used when a particular object is fetched more than once or twice inside functions. If that object changes id, purpose, or is removed altogether then we will just have to one reference at the top of the page.

Example

*var v1; // this variable has global scope*

*function func1(){*

*var v2; // this variable is only visible inside func1*

*// v1 is still visible here*

*}*

Variables and functions in JavaScript should be lower Camel Cased. The first letter of each variable or function should be lowercase, while the first letter of subsequent words should be capitalized. There should be no underscores between the words.

* **Use the Find Selector wherever possible**

Use the find() method, when possible. The key is to avoid jQuery to use its Sizzle engine, if it’s not necessary. Certainly, there will be times when this is not possible. But, if you don’t require the extra overhead, don’t go looking for it.

// Jquery initiates Sizzle

*$('#someDiv p.someClass').hide();*

// Better for all browsers, and Sizzle never inits.

*$('#someDiv').find('p.someClass').hide();*

More complicated selectors trigger jQuery’s full Sizzle engine, though brilliant, come along with a bit more overhead. Sizzle, **first takes your selector and turns it into an “array”** composed of each component of your selector.  
  
Recommendations:

* + Keep your **selectors simple**
  + Utilize the **find()** method. This way is better than using Sizzle and we can continue using the **browser’s native functions**.
  + When using Sizzle, **optimize the right-most part of your selector** as much as possible. It begins deciphering **from right to left, each item with regular expressions**.

It is also **possible to add some context to the selector** hence limiting the DOM traversal.

Example  
  
Syntax: *$(expr, context) (Using context)*

Which is equivalent to

*$(context).find(expr) (Using find())*

*$(“.logoImage”,”#logoContainer”) // Searches the parent with ID logoContainer for child elements with class .logoImage.*

* **Cache Jquery Objects**

It is good to get the habit of saving jQuery objects to a variable.

*$('#customer input.on).bind('click', function(){...});*

*$('# customer input.on).css('border', '3px dashed yellow');*

*$('#customer input.on).css('background-color', 'orange');*

*$('# customer input.on).fadeIn('slow');*

Instead, save the object to a local variable, and continue your operations:

*var $active\_employee = $('#validity input.on');*

*$active\_employee.bind('click', function(){...});*

*$active\_employee.css('border', '3px dashed yellow');*

*$active\_employee.css('background-color', 'orange');*

*$active\_employee.fadeIn('slow');*

Since our **local variable** is a jQuery wrapped set, we are using **$ as a prefix**. Never repeat a jQuery selection operation more than once in your application.

*var headerDiv = $('#headerDiv ');*

*headerDiv.hide();*

The reason why we cache the location of the headerDiv element is to limit the number of times that we have to traverse the DOM for this element to once. Doing this save a lot of time and improves performance.

The above example can be easily combined into one line, while achieving the same outcome.

*var headerDiv = $('#headerDiv').hide();*

In the above example, we still hide the headerDiv element, but the method also returns the JQuery object which is then referenced via the headerDiv variable.

* **Storing Jquery results**

If you would use **the jQuery result object(s) in another part of your program**, or if your **function execute more than once**, **cache it in an object with a global scope**. By defining a global container with jQuery results, we can reference them from within other functions:

// Define an object in the global scope (i.e. the window object)

*window.$my =*

*{*

*// Initialize all the queries you want to use more than once*

*head : $('head'),*

*personal\_banking : $('#p\_bank'),*

*cust\_banking : $('#c\_bank')*

*};*

*function do\_something()*

*{*

*// Now you can reference the stored results and manipulate them*

*var script = document.createElement('script');*

*$my.head.append(script);*

*// When working inside functions, continue to save jQuery results*

*// to your global container.*

*$my.c\_results = $('#some\_ul li');*

*$my.other\_results = $('#some\_table td');*

*// Use the global functions as you would a normal jQuery result*

*$my.other\_results.css('border-color', 'red');*

*$my.cusotmerGo.css('border-color', 'green');}*

* **Harness the power of Chaining**

The previous example can also be accomplished like this:

*var $active\_customer = $('#validity input.on');$active\_customer.bind('click', function(){...})*

*.css('border', '3px dashed yellow')*

*.css('background-color', 'orange')*

*.fadeIn('slow');This allows us to write less code, making our JavaScript more lightweight.*

* **Use Sub-queries**

jQuery allows us to run additional selector operations on a wrapped set. This reduces performance overhead on subsequent selections since we already grabbed and stored the parent object in a local variable.

*<div id="content">*

*<form method="post" action="/">*

*<h2> Are you an Existing US Bank Customer? </h2>*

*<ul id="validity">*

*<li><input type="radio" class="on" name="customer" value="Yes" /> Yes</li>*

*<li><input type="radio" class="off" name="customer" value="No /> No</li> </ul>*

*<input class="button" id=" customer" type="submit" value="Go" />*

*</form>*

*</div>*

For example, we can leverage sub-queries to grab the active and inactive values and cache them for later manipulation.

*var $validity = $('#customer'),*

*$active\_customer = $validity.find('input.on'),*

*$inactive\_customer = $validity.find('input.off');*

Tip: You can declare multiple local variables by separating them with commas

DOM Scripting

**Limit Direct DOM Manipulation**

The basic idea here is to create exactly what you need in memory, and then update the DOM. This is not a jQuery best practice, but a must for efficient JavaScript. Direct DOM manipulation is slow. For example, if you need to dynamically create a list of elements, do not do this:

*var top\_100\_list = [...], // assume this has 100 unique strings*

*$mylist = $('#mylist'); // jQuery selects our <ul> element*

*for (var i=0, l=top\_100\_list.length; i<l; i++)*

*{ $mylist.append('<li>' + top\_100\_list[i] + '</li>');*

*}*Instead, we want to create the entire set of elements in a string before inserting into the DOM:

*var top\_100\_list = [...], // assume this has 100 unique strings*

*$mylist = $('#mylist'), // jQuery selects our <ul> element*

*top\_100\_li = ""; // This will store our list items*

*for (var i=0, l=top\_100\_list.length; i<l; i++)*

*{*

*top\_100\_li += '<li>' + top\_100\_list[i] + '</li>';*

*}*

*$mylist.html(top\_100\_li);*

Even faster, we should always wrap many elements in a single parent node before insertion:

*var top\_100\_list = [...], // assume this has 100 unique strings*

*$mylist = $('#mylist'), // jQuery selects our <ul> element*

*top\_100\_ul = '<ul id="#mylist">'; // This will store our entire unordered list*

*for (var i=0, l=top\_100\_list.length; i<l; i++)*

*{*

*top\_100\_ul += '<li>' + top\_100\_list[i] + '</li>';*

*}*

*top\_100\_ul += '</ul>'; // Close our unordered list*

$mylist.replaceWith(top\_100\_ul);If you do the above and are still concerned about performance:

Give **jQuery’s clone() method** a try. This creates a copy of the node tree, which you can manipulate “off-line” and then insert back in when you are ready.

Use DOM DocumentFragments. As the creator of jQuery points out, they perform much better than direct DOM manipulation. The idea would be to create what you need (similar to what we did above with a string), and use the jQuery insert or replace methods.

**Leverage Event Delegation (a.k.a. Bubbling)**

Normally, **every event** (e.g. click, mouseover, etc.) in **JavaScript “bubbles” up the DOM tree to parent elements**. This is incredibly **useful** when we want **many elements (nodes) to call the same function**. Instead of binding an event listener function to many nodes—very inefficient—you can bind it once to their parent, and have it figure out which node triggered the event. For example, say we are developing a large form with many inputs, and want to toggle a class name when selected. A binding like this is inefficient:

*$('#myList li).bind('click', function(){*

*$(this).addClass('clicked');*

*// do stuff*

*});Instead, we should listen for the click event at the parent level:*

*$('#myList).bind('click', function(e){*

*var target = e.target, // e.target grabs the node that triggered the event.*

*$target = $(target); // wraps the node in a jQuery object*

*if (target.nodeName == 'LI') {*

*$target.addClass('clicked');*

*// do stuff*

*}*

*});*

The parent node acts as a dispatcher and can then do work based on what target element triggered the event. If you find yourself binding one event listener to many elements, you are doing something wrong (and slow).

**Document.all**

It is always considered as bad practices to use “document.all” and it is not a standard javascript DOM feature. There are certain rules for using document.all and please find those rules below,

* Always try other standard methods first
* Only fall back to using document.all as a last option
* Only use it if you need to support IE versions earlier than 5.0
* Always check that it is supported with "if (document.all) { }" around the block where you use it

Sample:

*if (document.getElementById) {*

*var obj = document.getElementById("myId");*

*}*

*else if (document.all) {*

*var obj = document.all("myId");*

*}*

**Existence of DOM object**

* It is always considered as **best practices to check for the existence of any or DOM objects before accessing** the same or else **unnecessary error will be thrown if the accessing objects are not available.**

*If(document.obj)*

*{  
 document.obj.style.display=’block’;*

*}*

Best Practices:

## Scripts at the bottom of the page:

* It is always advisable to use scripts at the bottom of the page since the primary goal will always be the page to load faster than and as quickly as possible or else the user will be forced to wait until the entire file has been loaded.
* Moreover the functionality based script files can also be placed at the bottom of the page, just before closing the body tag.

## Avoid Heavy Nesting

* **Nesting** code **explains its logic and makes it much easier to read**, however nesting it too far can also make it **hard to follow what you are trying to do**. Readers of your code shouldn’t have to scroll horizontally, or suffer confusion when their code editors wrap long lines.
* **Nesting loops will reduce the performance** and so avoid using nested loops**. Instead “and” and “or” operators can be used to combine conditions in a single loop**.
* Other way is multiple selectors can be used in a single statement.

Incorrect example**:**

*if ($(this).hasClass("classname1")) {*

*if ($(this).hasClass("classname2")) {*

*if ($($(this)).hasClass("classname3")){*

*if ($($(this)).hasClass("classname4")){*

*} } } } }*

Correct Example**:**

*If ($(this).has (“classname1 classname2 classname3 classname4”))*

*{*

*}*

* Different Jquery functions related to a particular object can be combined in a chained way using the concept of “**Chainability**”

*$(“#id”).attr(‘class’,’test’).show();*

## Lost Features

* Java script should be used appropriately keeping in mind the other features like accessibility, simple language, pop-ups enabled/disabled options and provide an alternate to address all the above.
* For example, the below code may seem to be right but for those who have javascript disabled, this screen will never be available.

Example of inaccessible pop-up window due to incorrect use of href attributes

*<a href=”javascript:window.open(‘terms\_conditions.html’, popup, ‘width=300’);”>Terms and*

*Conditions</a>*

This has to be coded like the below example

*<a href=”terms\_conditions.html” onClick=”window.open(‘terms\_conditions.html’, popup, ‘width=300’); return false;”>Terms and Conditions</a>*

Incorrect Sample:

*<a onclick="bank()" href="#">Click!</a>*

Correct Sample:

*<a href="home.html" class="bank">Click!</a>*

JS file

|  |  |  |
| --- | --- | --- |
| *$('a.bank').click(function(){* | | |
| *alert('you are in US bank now!');* |  |
| *});* |  |

* This ensures that the page is accessible without java script enabled too.

**Defer to $(window).load**

* There is an interest among jQuery developers to hook everything into the $(document).ready pseudo event. After all, it is used in most examples you will find. **Although $(document).ready is incredibly useful, it occurs during page render while objects are still downloading.** If you notice your page stalling while loading, all those $(document).ready functions could be the reason why.
* You can reduce CPU utilization during the page load by binding your jQuery functions to the $(window).load event, which occurs after all objects called by the HTML (including <iframe> content) have downloaded.

*$(window).load(function(){*

*// jQuery functions to initialize after the page has loaded.*

*});*

Functionality such as drag and drop, binding visual effects and animations, pre-fetching hidden images, etc., are all good candidates for this technique.

**Avoid “With” Statement**

* Initially “with” statement was considered as best practices since it enables shorthand for accessing deeply nested objects.

Incorrect Sample:

*with (being.bank.customer.creditcards) {*

*studentcard = true;*

*visacard = true;*

*}*

Instead of

being. *bank.customer.creditcards*. *studentcard* = true;

being. *bank.customer.creditcards*.visacard= true;

* But later it was recognized that “with” statement had bad performance while setting new members

Correct Sample:

*var o = bank.customer.creditcards*.*;*

*o. studentcard = true;*

*o. visacard = true;*

**Avoid using void()**

* The void() function is **not supported by all browsers**
* The inbuilt void() function is supported since JavaScript 1.1, therefore it is considered as best practices to define your own void()

*<script type="text/javaScript">*

*<!--*

*function myVoid() { } // create a void function*

*//-->*

*</script>*

*<a href="#" onclick="this.href='javascript:myVoid()'">non functional text link</a>*

**Pass functions, not strings, to setTimeout() and setInterval()**

* If you pass a string into setTimeout() or setInterval() the string will be evaluated the same way as with eval which is **slow**. **Wrap your code into an anonymous function** instead so that it can be interpreted and optimized during compilation.

Slow*:*

*setInterval('doSomethingPeriodically()', 1000);*

*setTimeOut('doSomethingAfterFiveSeconds()', 5000);*

Faster*:*

*setInterval(doSomethingPeriodically, 1000);*

*setTimeOut(doSomethingAfterFiveSeconds, 5000);*

**Analytics Implementation**

* If web analytics needs to be implemented in page the web analytics code should not added in to any of the other js files. It should be kept as separate file and needs to be added to page as server side include.
* These web analytics data fetching code should be made independent from the pages so that when some other new vendor is recommended by business we can make the change only to the particular include file which will get reflected in all the pages.
* Also the web analytics code and js should be loaded at the bottom of the page to get the scope of all the available DOM element’s references and not scattered across the document.

Example:

*The web trends codes are added to global.js file which is not an efficient way*

**Logical Grouping of JavaScript**

* Logical grouping of JavaScript function should be done and placed in separate js files so that based on the page requirement we can include js files. This will reduce the page loading time and thus the performance is improved.

Example:

*The form validation js can be placed in a separate file and can be included only in the form based pages where validation is required.*

**Comment Your Code**

* It is always considered as best practice to comment your code as it will be of very ease during maintenance
* Always comment important section of the code

**JavaScript Optimization**

* All JavaScript, CSS files has to be minified.
* Maintain one minified version and one normal version. Any Development can be done using normal version and once changes or development is completed minify the normal version file and use it. This way we don’t have to uncompress the files.
* From CSS perspective, use of CSS sprites for images which will reduce the no of server requests and the page will load faster. Images should be optimized for web using Photoshop. Again we can optimize the image using some tools like http://www.smushit.com/ysmush.it

**Optimize LOOPS:**

* The most common mistake in loops is to read **the length attribute of an array at every iteration**

Incorrect Sample:

*var names = ['George','Ringo','Paul','John'];*

*for(var i=0;i<names.length;i++){*

*doSomeThingWith(names[i]);*

*}*

* In the above example, the script needs to read the length of the array every time when the loop runs. This can be better optimized by storing the length in variable.

*var names = ['George','Ringo','Paul','John'];*

*var all = names.length;*

*for(var i=0;i<all;i++){*

*doSomeThingWith(names[i]);*

*}*

* Caching math functions in variables before executing calculations within a loop

Wrong Way:

*var d=35;*

*for (var i=0; i<1000; i++) {*

*y += Math.sin(d)\*10;*

*}*

Better Approach:

*var d = 55;*

*var math\_sind = Math.sin(d)\*10;*

*for (var i=0; i<1000; i++) {*

*y += math\_sind;*

*}*

**Eliminate Query Waste**

Only run functions that are applicable to the page. The most efficient way to do this is to use inline initialization functions so your template has full control over when and where JavaScript executes.

*<script type="text/javascript>*

*mylib.article.init();*

*</script>*

</body>

If your page template includes any variety of modules that may or may not be on the page, or for visual reasons you need them to initialize sooner, you could place the initialization function immediately after the module.

*<ul id="validity">*

*<li><input type="radio" class="on" name="customer" value="Yes" /> Yes</li>*

*<li><input type="radio" class="off" name="customer" value="No /> Yellow</li> </ul>*

*<script type="text/javascript>*

*mylib.customer.init();*

*</script>*

*Your Global JS library would look something like this:*

*var mylib =*

*{*

*article\_page :*

*{*

*init : function()*

*{*

*// Article page specific jQuery functions.*

*}*

*},*

*customer:*

*{*

*init : function()*

*{*

*// customer jQuery functions.*

*}*

*}*

*}*

## Quick JavaScript Checklist for developers

* **Are there any functions defined internally in the page?**
* **Can any JavaScript files be logically merged thereby reducing the number of external JavaScript files?**
* **Is JavaScript type used to include the external file instead of language attribute which is deprecated?**
* **Are any JavaScript keywords used for custom identifiers or variables?**
* **Are comments given at the necessary places?**
* **Are frameworks used efficiently for enhancing UI aspects?**
* **Are the functions defined properly enhancing reusability?**
* **Are there any duplications of functions or code?**
* **Is the syntax efficiently and correctly used?**
* **Is <noscript> tag used to provide an alternate content for users that have disabled scripts in their browser or have a browser that doesn’t support client-side scripting?**
* **Is your JavaScript files compressed?**
* **Is your JavaScript Validated?**