
JavaScript Tips for ASP.NET – Part 2

Summary: This article provides useful JavaScript tips and tricks for ASP.NET developers. (13 printed pages)

Contents

12. Cache your scripts.....	1
13. Get inner text.....	2
14. Toggle display.....	2
15. Preload your images.....	3
16. Concatenate your strings efficiently.....	4
17. Use hash tables for efficient lookups.....	5
18. Check whether an item exists in the array.....	7
19. Retrieve query string values.....	8
20. Move the focus away from your dropdown after change event.....	9
21. Handle asynchronous operations with minimally growing arrays.....	10
22. Pass data from one page to another using post.....	11

Overview

There is no doubt that ASP.NET server controls make your life easy on server-side. Third-party components like **Karamasoft UISuite™** make it a lot easier on both server-side and client-side. However, you may still need some knowledge on client-side programming to provide your website visitors with a powerful user interface.

JavaScript is the most commonly used language for client-side programming regardless of your server-side development environment. Even though this article may be applied to other languages and frameworks, it is more appropriate for ASP.NET developers because most of the tips are accompanied with sample code in ASPX and C#.

12. Cache your scripts

Using a JavaScript **include file** has several advantages compared to directly embedding JavaScript code inside a script block. The main advantage is that it provides code reuse by keeping the code in a central place. Another advantage is that your page size will be reduced significantly causing the page to be easier to maintain. Probably the most important advantage is caching by the browser. Subsequent requests to your page will retrieve the JavaScript library from the browser cache in order to load a lot faster.

You can place your JavaScript code in a **“.js”** file, and include it in your web pages using the following syntax:

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

13. Get inner text

Internet Explorer provides an **innerText** property to get or set the text between the start and end tags of the object. Firefox does not provide an **innerText** property but a **textContent** property to get or set the inner text. For the browsers that do not provide a property to get or set the inner text, you can clear the HTML tags inside the **innerHTML** property.

```
function GetInnerText(elem) {  
    return (typeof(elem.innerText) != 'undefined') ? elem.innerText :  
    (typeof(elem.textContent) != 'undefined') ? elem.textContent :  
    elem.innerHTML.replace(/<[^\>]+>/g, '');  
}
```

14. Toggle display

You can toggle the display of an HTML element easily by toggling its style display value between 'none' and ''.

JavaScript

```
function ToggleDisplay(elemId) {  
    var elem = document.getElementById(elemId);  
    elem.style.display = (elem.style.display != 'none') ? 'none' : '';  
}
```

ASPX

```
<asp:LinkButton ID="lbToggleDisplay" runat="server"  
OnClick="ToggleDisplay('pnlToggleDisplay'); return false;">Toggle  
Display</asp:LinkButton>  
  
<asp:Panel ID="pnlToggleDisplay" runat="server">
```

Panel content here!

```
</asp:Panel>
```

15. Preload your images

If you want to display your images instantly when the page is loaded, you should preload your images using JavaScript in the **HEAD** section of your web page. This technique also helps when you have rollover effects on your images where you display a different image when the mouse is over the image.

All you need to do is to create a new JavaScript **Image** object and set its **src** attribute to the path of your image.

```
<head runat="server">
<script type="text/javascript">
var img1 = new Image();
img1.src = '/Images/Product/UISuite.gif';
</script>
</head>
```

Note that you need to create new JavaScript Image objects for all the images you need to cache using different local variable names. To overcome this issue, you can use the following JavaScript function that takes the image paths as arguments and stores the JavaScript Image objects inside array elements.

```
<head runat="server">
<script type="text/javascript">
function PreloadImages() {
    var lenArg = arguments.length;
    if (lenArg > 0) {
        var imgArr = new Array();
        for (var i = 0; i < lenArg; i++) {
            imgArr[i] = new Image();
            imgArr[i].src = arguments[i];
        }
    }
}
```

```
    }  
}  
  
PreloadImages('/Images/Product/UltimateEditor.gif',  
'/Images/Product/UltimateSearch.gif',  
'/Images/Product/UltimateSpell.gif');  
  
</script>  
</head>
```

This function takes as many image path arguments as you pass, creates a new JavaScript Image object for each of them and stores them in an array.

16. Concatenate your strings efficiently

Even though it is easy to concatenate strings using the **+** operator, string operations become very costly in terms of CPU cycle when the strings get bigger and there are a lot of strings to concatenate.

The better alternative is to create your own **StringBuilder** JavaScript object that is similar to the StringBuilder object in the .NET Framework. You can define your StringBuilder object with a buffer array property to store the strings to concatenate. Then add an Append method to add a string to the buffer array. The JavaScript Array object provides a join method that puts all the elements of an array into a string separated by a specified delimiter. **ConvertToString** method will join all elements of the buffer array into a string.

```
function StringBuilder() {  
    this.buffer = [];  
}  
  
StringBuilder.prototype.Append = function(str) {  
    this.buffer[this.buffer.length] = str;  
};  
  
StringBuilder.prototype.ConvertToString = function() {  
    return this.buffer.join('');  
}
```

```
};
```

Instead of the following:

```
var str = 'This ' + 'is ' + 'a ' + 'test';
```

You can now use the following:

```
var sb = new StringBuilder;  
sb.Append('This ');  
sb.Append('is ');  
sb.Append('a ');  
sb.Append('test');  
var str = sb.ConvertToString();
```

Our tests indicated that concatenating the string 'test' 50,000 times using the StringBuilder Append method worked 10 times faster than using the + operator in Internet Explorer.

17. Use hash tables for efficient lookups

Hash tables provide a way of efficient addition of new entries and the time spent searching for a specified key is independent of the number of items stored, which is $O(1)$. Before explaining how to create a hash table in JavaScript, let's begin with a quick background about the associative arrays in JavaScript.

An associative array (hash) is an abstract data type composed of a collection of keys and a collection of values, where each key is associated with one value. In JavaScript, all objects are associative arrays. You can reach the object properties as follows:

```
document.forms['Form1']
```

Or

```
document.forms.Form1
```

JavaScript objects do not provide a length property like Array object but you can iterate through the object properties using the following construct:

```
for (var prop in object)
```

For example, **UltimateEditor** component provides an associative array called UltimateEditors in the client-side API. Each UltimateEditor client-side object is defined as the property of UltimateEditors associative array. If you have multiple UltimateEditor components in your page, the following JavaScript code will iterate through all your UltimateEditor objects and display their HTML contents:

```
for (var ueObj in UltimateEditors) {  
    alert(ueObj.GetEditorHTML());  
}
```

We can now create our HashTable object using the associative array functionality. Our HashTable object will have two properties: hashArr property that stores key/value pairs and length property that keeps track of the number of items in the hash table. It will provide four methods: get method to retrieve the value for a given key, put method to add a key/value pair to the hash table, remove method to remove a key/value pair from the hash table and has method to return whether given key value exists in the hash table.

```
function HashTable(){  
    this.hashArr = new Array();  
    this.length = 0;  
}  
  
HashTable.prototype.get = function(key) {  
    return this.hashArr[key];  
};  
  
HashTable.prototype.put = function(key, value) {  
    if (typeof(this.hashArr[key]) == 'undefined') {
```

```
        this.length++;
    }
    this.hashArr[key] = value;
};

HashTable.prototype.remove = function(key) {
    if (typeof(this.hashArr[key]) !== 'undefined') {
        this.length--;
        var value = this.hashArr[key];
        delete this.hashArr[key];
        return value;
    }
};

HashTable.prototype.has = function(key) {
    return (typeof(this.hashArr[key]) !== 'undefined');
};
```

Now let's create a sample hash table for an efficient phone lookup:

```
var phoneLookup = new HashTable();
phoneLookup.put('Jane', '111-222-3333');
phoneLookup.put('John', '444-555-6666');
alert(phoneLookup.get('Jane'));
```

18. Check whether an item exists in the array

JavaScript **Array** object does not provide a built-in method to return whether a given value exists in the array. However, we can add a prototype method to extend Array functionality by adding a method to return whether the array has the given value.

```
Array.prototype.has = function(value) {
```

```
var i;  
for (var i = 0, loopCnt = this.length; i < loopCnt; i++) {  
    if (this[i] === value) {  
        return true;  
    }  
}  
return false;  
};
```

Note that this method uses `===` operator to check if the value is identical (is equal to and is of the same type) to the value in the array. If you want to check only whether they are equal, you should replace `===` with `==` operator (is equal to).

Then, we can use this method as follows:

```
var arr = new Array();  
arr[0] = 'test';  
alert(arr.has('test')); // Should display true  
alert(arr.has('test2')); // Should display false
```

19. Retrieve query string values

ASP.NET provides **Request.QueryString** collection that retrieves the values of the variables in the HTTP query string, but there is no built-in mechanism to retrieve query string collection on the client-side. However, you can parse the window location search property to get the query string values by using the `GetAttributeValue` method that we provided in one of our previous tips to center your window. Note that `window.location.search` property gets or sets the substring of the `href` property that follows the question mark.

```
function GetAttributeValue(attribList, attribName, firstDelim,  
secondDelim) {  
    var attribNameLowerCase = attribName.toLowerCase();  
    if (attribList) {  
        var attribArr = attribList.split(firstDelim);
```

```
for (var i = 0, loopCnt = attribArr.length; i < loopCnt; i++) {
    var nameValueArr = attribArr[i].split(secondDelim);
    for (var j = 0, loopCnt2 = nameValueArr.length; j < loopCnt2;
j++) {
        if (nameValueArr[0].toLowerCase().replace(/\s/g, '') ==
attribNameLowerCase && loopCnt2 > 1) {
            return nameValueArr[1];
        }
    }
}
```

This method takes **three arguments**: a name/value pair list, the attribute name to retrieve its value, the first delimiter and the second delimiter. The first delimiter will be ampersand and the second delimiter would be equal sign to parse query string variables. Then you can use the following method to retrieve the query string value for given query string variable name.

```
function GetQueryStringValue(varName) {
    return GetAttributeValue(window.location.search.substring(1),
varName, '&', '=');
}
```

20. Move the focus away from your dropdown after change event

When the user changes the selection in a **DropDownList**, the focus remains inside the dropdown. If the user presses up/down arrow key or moves the mouse wheel, that action triggers the **onchange** event on the dropdown.

You can prevent this by making the dropdown lose focus in the onchange event.

JavaScript

```
function LoseFocus(ddlElem) {
    ddlElem.blur();
}
```

```
}
```

ASPX

```
<asp:DropDownList id="ddlLoseFocus" runat="server" AutoPostBack="true" />
```

C#

```
protected void Page_Load(object sender, EventArgs e)
{
    ddlLoseFocus.Attributes.Add("onchange", "LoseFocus(this)");
}
```

21. Handle asynchronous operations with minimally growing arrays

When you work with asynchronous AJAX callbacks, you might need to store the callback results in an array to process them later. The tricky part is to keep the array size at minimum. In order to do this, you can empty the array item when you are done with processing it and make it available for future callback results and always insert the new callback result to the first available array item.

```
function InsertActiveCallbackArr(arrayObj, arrayElem) {
    var i = 0;
    for (var loopCnt = arrayObj.length; (i < loopCnt) && arrayObj[i]; i++);
    arrayObj[i] = arrayElem;
}
```

When you are done with the callback result, simply set that array item to null.

```
function ProcessActiveCallbackArr(arrayObj) {
    for (var i = 0, loopCnt = arrayObj.length; i < loopCnt; i++) {
        // Process the callback result here
    }
}
```

```
// Then set the array item to null  
arrayObj[i] = null;  
}  
}
```

22. Pass data from one page to another using post

If you need to pass small amounts of data from one page to another, you can pass it in the query string and the target page can retrieve it through Request.QueryString collection (GET operation). However, if you need to pass large amount of data you cannot pass it in the query string since Internet Explorer has maximum 2048 character limit on the address of the page. In these cases, you can pass data from one page to another using POST operation.

In your source page, define a hidden variable to hold the data to pass and pass only its ID to the target page in the query string.

The target page should first get that ID through Request.QueryString collection. Then it should set its own hidden variable value with the one in the source frame on the client-side during BODY onload event handler. You can use opener object on the client-side to retrieve the DOM elements in the opener page. Then it should do a postback to itself and retrieve the value of its hidden variable through Request.Form collection.

Source Page JavaScript

```
function PassData(ctlID) {  
    window.open('TargetPage.aspx?ctlID=' + ctlID, 'NewWindow',  
'width=400,height=300,location=no,menubar=no,resizable=no,scrollbars=no,  
,status=yes,toolbars=no');  
}
```

Source Page ASPX

```
<asp:TextBox ID="txtPassData" runat="server" Text="Data to pass"  
CssClass="TextBox"></asp:TextBox>  
  
<asp:Button ID="btnPassData" runat="server" Text="Pass Data"  
CssClass="Button" OnClientClick="PassData('txtPassData'); return  
false;" />
```

Target Page JavaScript

```
function Body_Onload() {  
    var pageStatusElem = document.getElementById('hfPageStatus');  
  
    // Page status hidden field value is initially empty string  
    if (pageStatusElem.value == '') {  
        // Set text to spell check hidden field value to the text in  
the opener window  
        document.getElementById('hfTextPassed').value =  
opener.window.document.getElementById('<%=Request.QueryString["ctlID"]%>').value;  
  
        // Set page status hidden field value so that when the page is  
posted back it can process text to spell check retrieved from the  
opener window  
        pageStatusElem.value = '<%=SET_TEXT%>';  
  
        // Post back the page  
        document.form1.submit();  
    }  
}
```

Target Page ASPX

```
<body onload="Body_Onload()">  
    <form id="form1" runat="server">  
        <div>  
            Text passed: <asp:Label ID="lblTextPassed" runat="server"  
Text="Label"></asp:Label>  
        </div>  
        <asp:HiddenField ID="hfTextPassed" runat="server" Value="" />  
        <asp:HiddenField ID="hfPageStatus" runat="server" Value="" />  
    </form>  
</body>
```

Target Page C#

```
public const string SET_TEXT = "SetText";
public const string TEXT_IS_SET = "TextIsSet";

protected void Page_Load(object sender, EventArgs e)
{
    // Initial entry
    if (hfPageStatus.Value == SET_TEXT)
    {
        // Set page status not to enter again
        hfPageStatus.Value = TEXT_IS_SET;
    }
    lblTextPassed.Text = hfTextPassed.Value;
}
```