UNIT-II

Frontend Development

Javascript basics

Learn JavaScript Tutorial

Our **JavaScript Tutorial** is designed for beginners and professionals both. JavaScript is used to create client-side dynamic pages.

JavaScript is *an object-based scripting language* which is lightweight and cross-platform.

JavaScript is not a compiled language, but it is a translated language. The JavaScript Translator (embedded in the browser) is responsible for translating the JavaScript code for the web browser.

What is JavaScript

JavaScript (js) is a light-weight object-oriented programming language which is used by several websites for scripting the webpages. It is an interpreted, full-fledged programming language that enables dynamic interactivity on websites when applied to an HTML document. It was introduced in the year 1995 for adding programs to the webpages in the Netscape Navigator browser. Since then, it has been adopted by all other graphical web browsers. With JavaScript, users can build modern web applications to interact directly without reloading the page every time. The traditional website uses js to provide several forms of interactivity and simplicity.

Although, JavaScript has no connectivity with Java programming language. The name was suggested and provided in the times when Java was gaining popularity in the market. In addition to web browsers, databases such as CouchDB and MongoDB uses JavaScript as their scripting and query language.

Features of JavaScript

There are following features of JavaScript:

1. All popular web browsers support JavaScript as they provide built-in execution environments.
2. JavaScript follows the syntax and structure of the C programming language. Thus, it is a structured programming language.
3. JavaScript is a weakly typed language, where certain types are implicitly cast (depending on the operation).
4. JavaScript is an object-oriented programming language that uses prototypes rather than using classes for inheritance.
5. It is a light-weighted and interpreted language.
6. It is a case-sensitive language.
7. JavaScript is supportable in several operating systems including, Windows, macOS, etc.
8. It provides good control to the users over the web browsers.

History of JavaScript

In 1993, **Mosaic**, the first popular web browser, came into existence. In the **year 1994**, **Netscape** was founded by **Marc Andreessen**. He realized that the web needed to become more dynamic. Thus, a 'glue language' was believed to be provided to HTML to make web designing easy for designers and part-time programmers. Consequently, in 1995, the company recruited **Brendan Eich** intending to implement and embed Scheme programming language to the browser. But, before Brendan could start, the company merged with **Sun Microsystems** for adding Java into its Navigator so that it could compete with Microsoft over the web technologies and platforms. Now, two languages were there: Java and the scripting language. Further, Netscape decided to give a similar name to the scripting language as Java's. It led to 'Javascript'. Finally, in May 1995, Marc Andreessen coined the first code of Javascript named '**Mocha**'. Later, the marketing team replaced the name with '**LiveScript**'. But, due to trademark reasons and certain other reasons, in December 1995, the language was finally renamed to 'JavaScript'. From then, JavaScript came into existence.

Application of JavaScript

JavaScript is used to create interactive websites. It is mainly used for:

- Client-side validation,
- Dynamic drop-down menus,
- Displaying date and time,
- Displaying pop-up windows and dialog boxes (like an alert dialog box, confirm dialog box and prompt dialog box),
- Displaying clocks etc.

JavaScript Example

```
1. <script>
2. document.write("Hello JavaScript by JavaScript");
3. </script>
```

JavaScript Example

Javascript example is easy to code. JavaScript provides 3 places to put the JavaScript code: within body tag, within head tag and external JavaScript file.

Let’s create the first JavaScript example.

```
1. <script type="text/javascript">
2. document.write("JavaScript is a simple language for javatpoint learners");
3. </script>
```

Test it Now

The **script** tag specifies that we are using JavaScript.

The **text/javascript** is the content type that provides information to the browser about the data.

The **document.write()** function is used to display dynamic content through JavaScript. We will learn about document object in detail later.

---

3 Places to put JavaScript code

1. Between the body tag of html
2. Between the head tag of html
3. In .js file (external javaScript)

---

1) JavaScript Example : code between the body tag

In the above example, we have displayed the dynamic content using JavaScript. Let's see the simple example of JavaScript that displays alert dialog box.

```
<script type="text/javascript">
 alert("Hello Javatpoint");
</script>
```

---

2) JavaScript Example : code between the head tag

Let's see the same example of displaying alert dialog box of JavaScript that is contained inside the head tag.

In this example, we are creating a function msg(). To create function in JavaScript, you need to write function with function_name as given below.

To call function, you need to work on event. Here we are using onclick event to call msg() function.

```
<html>
<head>
<script type="text/javascript">
function msg(){
 alert("Hello Javatpoint");
}
</script>
</head>
<body>
<p>Welcome to JavaScript</p>
<form>
<input type="button" value="click" onclick="msg()"/>
</form>
</body>
</html>
```

External JavaScript file

We can create external JavaScript file and embed it in many html page.

It provides **code re usability** because single JavaScript file can be used in several html pages.

An external JavaScript file must be saved by .js extension. It is recommended to embed all JavaScript files into a single file. It increases the speed of the webpage.

Let's create an external JavaScript file that prints Hello Javatpoint in a alert dialog box.

message.js

```
1. function msg(){
2.  alert("Hello Javatpoint");
3. }
```

Let's include the JavaScript file into html page. It calls the JavaScript function on button click.

index.html

```
1. <html>
2. <head>
3. <script type="text/javascript" src="message.js"></script>
4. </head>
5. <body>
6. <p>Welcome to JavaScript</p>
7. <form>
8. <input type="button" value="click" onclick="msg()"/>
9. </form>
10. </body>
11. </html>
```

Advantages of External JavaScript

There will be following benefits if a user creates an external javascript:

1. It helps in the reusability of code in more than one HTML file.
2. It allows easy code readability.
3. It is time-efficient as web browsers cache the external js files, which further reduces the page loading time.
4. It enables both web designers and coders to work with html and js files parallelly and separately, i.e., without facing any code conflictions.
5. The length of the code reduces as only we need to specify the location of the js file.

Disadvantages of External JavaScript

There are the following disadvantages of external files:

1. The stealer may download the coder's code using the url of the js file.
2. If two js files are dependent on one another, then a failure in one file may affect the execution of the other dependent file.
3. The web browser needs to make an additional http request to get the js code.
4. A tiny to a large change in the js code may cause unexpected results in all its dependent files.
5. We need to check each file that depends on the commonly created external javascript file.
6. If it is a few lines of code, then better to implement the internal javascript code.

JavaScript Comment

The **JavaScript comments** are meaningful way to deliver message. It is used to add information about the code, warnings or suggestions so that end user can easily interpret the code.

The JavaScript comment is ignored by the JavaScript engine i.e. embedded in the browser.

Advantages of JavaScript comments

There are mainly two advantages of JavaScript comments.

1. **To make code easy to understand** It can be used to elaborate the code so that end user can easily understand the code.
2. **To avoid the unnecessary code** It can also be used to avoid the code being executed. Sometimes, we add the code to perform some action. But after sometime, there may be need to disable the code. In such case, it is better to use comments.

Types of JavaScript Comments

There are two types of comments in JavaScript.

1. Single-line Comment
2. Multi-line Comment

JavaScript Single line Comment

It is represented by double forward slashes (//). It can be used before and after the statement.

Let’s see the example of single-line comment i.e. added before the statement.

```
<script>
// It is single line comment
document.write("hello javascript");
</script>
```

Test it Now

Let’s see the example of single-line comment i.e. added after the statement.

```
<script>
var a=10;
var b=20;
var c=a+b;//It adds values of a and b variable
```

```
5. document.write(c);//prints sum of 10 and 20
6. </script>
```

JavaScript Multi line Comment

It can be used to add single as well as multi line comments. So, it is more convenient.

It is represented by forward slash with asterisk then asterisk with forward slash. For example:

```
1. /* your code here  */
```

It can be used before, after and middle of the statement.

```
1. <script>
2. /* It is multi line comment.
3. It will not be displayed */
4. document.write("example of javascript multiline comment");
5. </script>
```

javaScript Variable

A **JavaScript variable** is simply a name of storage location. There are two types of variables in JavaScript : local variable and global variable.

There are some rules while declaring a JavaScript variable (also known as identifiers).

1. Name must start with a letter (a to z or A to Z), underscore(_), or dollar($) sign.
2. After first letter we can use digits (0 to 9), for example value1.
3. JavaScript variables are case sensitive, for example x and X are different variables.

Correct JavaScript variables

```
1. var x = 10;
2. var _value="sonoo";
```

Incorrect JavaScript variables

```
1. var  123=30;
2. var *aa=320;
```

Example of JavaScript variable

Let’s see a simple example of JavaScript variable.

```
<script>
var x = 10;
var y = 20;
var z=x+y;
document.write(z);
</script>
```

Output of the above example
30

JavaScript local variable

A JavaScript local variable is declared inside block or function. It is accessible within the function or block only. For example:

```
<script>
function abc(){
var x=10;//local variable
}
</script>
```

Or,

```
<script>
If(10<13){
var y=20;//JavaScript local variable
}
</script>
```

JavaScript global variable

A **JavaScript global variable** is accessible from any function. A variable i.e. declared outside the function or declared with window object is known as global variable. For example:

```
<script>
var data=200;//gloabal variable
function a(){
document.writeln(data);
}
```

```
6. function b(){
7. document.writeln(data);
8. }
9. a();//calling JavaScript function
10. b();
11. </script>
```

JavaScript Global Variable

A **JavaScript global variable** is declared outside the function or declared with window object. It can be accessed from any function.

Let's see the simple example of global variable in JavaScript.

```
1. <script>
2. var value=50;//global variable
3. function a(){
4. alert(value);
5. }
6. function b(){
7. alert(value);
8. }
9. </script>
```

Declaring JavaScript global variable within function

To declare JavaScript global variables inside function, you need to use **window object**. For example:

```
1. window.value=90;
```

Now it can be declared inside any function and can be accessed from any function. For example:

```
1. function m(){
2. window.value=100;//declaring global variable by window object
3. }
4. function n(){
5. alert(window.value);//accessing global variable from other function
6. }
```

Internals of global variable in JavaScript

When you declare a variable outside the function, it is added in the window object internally. You can access it through window object also. For example:

```
1. var value=50;
```

```
2. function a(){
3. alert(window.value);//accessing global variable
4. }
```

OOPS ASPECTS IN JAVASCRIPT

What Is Object-oriented Programming?

Object-oriented Programming treats data as a crucial element in program development and doesn't allow it to flow freely around the system. It ties data more securely to the function that operates on it and protects it from accidental modification from an outside function. OOP breaks down a problem into several entities called objects and builds data and functions around these objects.

Basic concepts of Object-oriented Programming

Objects

Objects are the basic run-time bodies in an object-oriented framework. They may represent a place, a person, an account, a table of data, or anything that the program needs to handle. Objects can also represent user-defined data such as vectors, time, and lists.

Consider two objects, “customer” and “account” in a program. The customer object may send a message requesting the bank balance.

Classes

We know that objects hold the data and the functions to manipulate the data. However, the two can be bound together in a user-defined data type with the help of classes. Any number of objects can be created in a class. Each object is associated with the data of type class. A class is therefore a collection of objects of similar types.

For example, consider the class “Fruits”. We can create multiple objects for this class -

Fruit Mango;

This will create an object mango belonging to the class fruit.

Encapsulation

Encapsulation is the wrapping up/binding of data and function into a single unit called class. Data encapsulation is the most prominent feature of a class wherein the data is not accessible to the outside world, and only those functions wrapped inside the class can access it. These functions serve as the interface between the object’s data and the program.

Inheritance

The phenomenon where objects of one class acquire the properties of objects of another class is called Inheritance. It supports the concept of hierarchical classification. Consider the object “car” that falls in the class “Vehicles” and “Light Weight Vehicles”.

In OOP, the concept of inheritance ensures reusability. This means that additional features can be added to an existing class without modifying it. This is made possible by deriving a new class from the existing one.

OOP Concepts in JavaScript

Now that you are familiar with OOP concepts, this section will show you how JavaScript implements them.

Creating Objects in JavaScript

- We can create an object using the string literal in JavaScript.

```
var student = {
      name: "pp",
      age: 21,
      studies: "Computer Science",
   };
   document.getElementById("demo").innerHTML = student.name + " of the age " + student.age + " studies " + student.studies;
```

- Creating objects using the new keyword.

```
var student = new Object();
   student.name = "pp",
   student.age=21,
   student.studies = "Computer Science";
   document.getElementById("demo").innerHTML = student.name + " of the age " + student.age + " studies " + student.studies;
```

- Creating an object using the object constructor.

```
function stud(name, age, studies){
      this.name = name;
```

```
        this.age = age;
        this.studies = studies;
    }
    var student = stud("Chris", 21, "Computer Science");
    document.getElementById("demo").innerHTML = student.name + " of the age " +
student.age + " studies " + student.studies;
```

Class Implementation in JavaScript

JavaScript uses the ES6 standard to define classes. Consider the following example.

```
class Cars {
    constructor(name, maker, price) {
      this.name = name;
      this.maker =  maker;
      this.price = price;
    }
    getDetails(){
        return (`The name of the car is ${this.name}.`)
    }
  }
  let car1 = new Cars('Rolls Royce Ghost', 'Rolls Royce', '$315K');
  let car2 = new Cars('Mercedes AMG One', 'Mercedes', '$2700K');
  console.log(car1.name);
  console.log(car2.maker);
  console.log(car1.getDetails());
```

The output of the above code is

Encapsulation in JavaScript

Encapsulation includes wrapping the property and the function within a single unit. Consider the following example:

```
class Emp_details{
    constructor(name,id){
        this.name = name;
        this.id = id;
    }
    add_Address(add){
        this.add = add;
    }
    getDetails(){
        console.log(`Employee Name: ${this.name}, Address: ${this.add}`);
    }
}
let person1 = new Emp_details('Anand',27);
person1.add_Address('Bangalore');
person1.getDetails();
```

Here, the class holds the data variables name and id along with the functions add_Address and getDetails. All are encapsulated within the class Emp_details.

Memory Management in JavaScript

Memory management is an essential task when writing a good and effective program in some programming languages. This article will help you to understand different concepts of memory management in JavaScript. In low-level languages like C and C++, programmers should care about the usage of memory in some manual fashion. On the other hand, Javascript automatically allocates memory when objects are created into the environment and also it cleans the memory when an object is destroyed. JavaScript can manage all of these on its own but this does not imply that the developers do not need to worry about the memory management in JavaScript.

Memory management in any programming language involves three important phases, termed as memory life-cycle −

- Allocating the memory which is required in our program.
- Utilize the allocated memory unit.
- After completion, clear the memory block.

Different Strategies to Allocate Memory in JavaScript

Allocating by value initialization

In JavaScript, we do not need to care about allocating memory for simple variables. We can directly assign values to some variables and it will allocate necessary memory on its own.

Syntax

```
var variable1 = <value>
var variable2 = <value>
```

Example

For simple allocation by values, see the following example.

Source Code

```
<head>
<title>HTML Console</title>
</head>
<body>
<h3> Output Console </h3>
<p> Output:</p>
<div id="output">
</div>
<div id="opError" style="color : #ff0000">
</div>
<script>
var content ="
var error ="
varopDiv=document.querySelector('#output')
varopErrDiv=document.querySelector('#opError')

// actual javascript code
try{
```

```
var number =52;
varst='my_string';
var student ={
        name:'Smith',
        roll:5,
        age:23,
};
vararr=[15,null,'another_string'];
    content +="Allocated memory for number: "+JSON.stringify(number)+'<br>'
    content +="Allocated memory for string: "+JSON.stringify(st)+'<br>'
    content +="Allocated memory for student: "+JSON.stringify(student)+'<br>'
    content +="Allocated memory for array: "+JSON.stringify(arr)+'<br>'
}catch(err){
    error += err
}finally{

// display on output console
opDiv.innerHTML= content
opErrDiv.innerHTML= error
}
</script>
</body>
</html>
```

From the above example, it is clear that numbers and strings are single values, and allocation is also simple. But for objects and arrays, JavaScript can also easily allocate the memory based on their values.

Allocating by Function Call

Like variable value assignment, we can also create some memory blocks by calling some functions. For example, when a function returns a separate object it will automatically assign a new memory block to the system.

Syntax

Memory_reference = <function call which returns any value>

Examples

The following example uses a function that works on an HTML document. So this program will run on a browser or HTML editor.

Source Code

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8"/>
</head>
<body>
<script>
var e =document.createElement('div');
e.innerHTML="<h1> Header from JavaScript </h1>"
document.body.appendChild(e);
</script>
</body>
</html>
```

In this example, the JavaScript code is present inside the <script> tag in HTML. Please notice, in this case, initially, the document does not have any block inside <body>. The JavaScript creates a new component by calling createElement(), and then a new div block is created. This block allocates the memory but only when a function is called. After that, the new component is added as a child of the body tag to use this inside the HTML document.

Using previously Allocated Memory in JavaScript

Using previously allocated memory is just reading or writing values from some variables which are assigned previously. We can update its existing value with some other values. See the following example for a better understanding−

Example

Initially allocating memory for a variable, then reading the value from it. Writing a new value and again reading from it.

Source Code

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Console</title>
</head>
<body>
<h3> Output Console </h3>
<p> Output:</p>
<div id="output">
</div>
<div id="opError" style="color : #ff0000">
</div>
<script>
var content ="
var error ="
opDiv=document.querySelector('#output')
varopErrDiv=document.querySelector('#opError')

// actual javascript code
try{
var a =52;// allocate memory
        content +="Reading value of variable a: "+JSON.stringify(a)+'<br>'
        a =100
        content +="Reading value of variable a: "+JSON.stringify(a)+'<br>'
}
catch(err){
        error += err
}
finally{

// display on output console
opDiv.innerHTML= content
opErrDiv.innerHTML= error
}
</script>
</body>
```

```
</html>
```

Deallocating memory blocks in JavaScript

When our purpose is served, we can remove the allocated memory block. In some low-level languages, this is a necessary step, otherwise, it may occupy memory spaces over time and the total system may crash. JavaScript also has native support of Garbage Collector, which cleans unnecessary memory blocks and cleans up the memory. But sometimes the compiler cannot understand whether a block will be used in later cases or not. In such cases, the Garbage Collector does not clean up that memory. To manually remove allocated locations, we can use the ‘delete’ keyword before the variable name.

Syntax

```
delete <variable_name>
```

The variable must be allocated beforehand, otherwise, it will raise an error while trying to delete that variable. Let us see one example to understand this concept clearly.

Example

Source Code

```
<!DOCTYPE html>
<html>
<head>
<title>HTML Console</title>
</head>
<body>
<h3> Output Console </h3>
<p> Output:</p>
<div id="output">
</div>
<div id="opError" style="color : #ff0000">
</div>
<script>
var content ="
var error ="
varopDiv=document.querySelector('#output')
varopErrDiv=document.querySelector('#opError')

// actual javascript code
try{
        a ="a simple variable";// allocate memory
        content +="Reading value of variable a: "+JSON.stringify(a)+'<br>'
delete a
        content +="Reading value of variable a: "+JSON.stringify(a)+'<br>'
}
catch(err){
        error += err
}
finally{

// display on output console
opDiv.innerHTML= content
opErrDiv.innerHTML= error
}
</script>
```

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

Note − The ‘delete’ keyword will only work when the variable is allocated directly (without using the var or let keyword).

Conclusion

Working with any programming language, the programmer should know the overall concept in depth. Memory management is one of the concerning issues, in which developers should properly manage the memory otherwise it will occupy unnecessary memory blocks and create major problems in the environment. JavaScript provides an additional garbage collector tool that automatically cleans the unused memory blocks. However, we can also deallocate memory by using the ‘delete’ keyword just before the variable name

AJAX for data exchange with server jQuery Framework

Short Description of AJAX

Ajax is only a name given to a set of tools that were previously existing.

The main part is XMLHttpRequest, a server-side object usable in JavaScript, that was implemented in Internet Explorer since the 4.0 version.

To get data on the server, XMLHttpRequestprovides two methods:

1. open: Creates a connection
2. send: Sends a request to the server

Data furnished by the server will be found in the attributes of the XMLHttpRequestobject:

1. responseXmlfor an XML file, or
2. responseTextfor a plain text

Take note that a new XMLHttpRequestobject has to be created for each new data request.

We have to wait for the data to be available to process it, and in this purpose, the state of availability of data is given by the readyStateattribute of XMLHttpRequest.

Attributes of XMLHttpRequest Class

1. *readyState*: The code successively changes value from 0 to 4

   0: Not initialized

1: Connection established
2: Request received
3: Answer in process
4: Finished

2. *status*: 200 is OK

   404 if the page is not found

3. *responseText*: Holds loaded data as a string of characters.
4. *responseXml*: Holds an XML loaded file, DOM's method allows to extract data.
5. *onreadystatechange*: Property that takes a function as value that is invoked when the readystatechangeevent is dispatched.

Methods of XMLHttpRequest Class

1. *open(mode, url,boolean)* : *mode*: type of request, GET or POST

   url: the location of the file, with a path
   boolean: true (asynchronous) / false (synchronous)
   optionally, a login and a password may be added to arguments

2. *send("string")*: string: POST data, null for a GET command
3. *abort()* : Cancels the current HTTP request
4. *getAllResponseHeaders()*: Retrieves the values of all the HTTP headers
5. getResponseHeader(string): Retrieves the value of an HTTP header from the response body
   string: name of http header
6. *setRequestHeader(name,value)*: Adds a new http header in the request
   name: name/identifier of the header
   value: value of the header

Using the Code

Here is a simple function 'AjaxRequest' which is implemented to perform the AJAX requests.

JavaScript

Shrink ▲

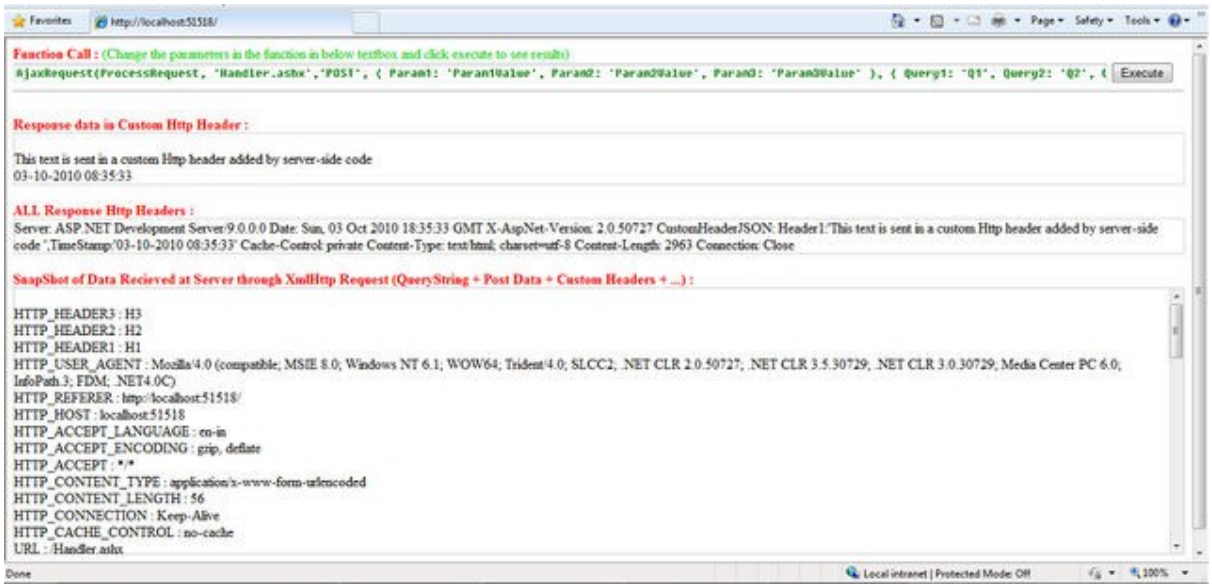
```
functionAjaxRequest(ReadyHandler,URL,Method,Params,QueryString,HttpHeaders) {
if (URL == null) { alert("Request URL is Empty"); }
else {

if (window.XMLHttpRequest) {// code for IE7+, Firefox, Chrome, Opera, Safari
xmlhttp = newXMLHttpRequest();
      }
else {// code for IE6, IE5
xmlhttp = newActiveXObject("Microsoft.XMLHTTP");
      }

//An anonymous function is assigned to the event indicator.
xmlhttp.onreadystatechange = function() {

//200 status means ok, otherwise some error code is returned, 404 for example
```

```
//The 4 state means for the response is ready and sent by the server.
if (xmlhttp.readyState == 4&&xmlhttp.status == 200) {
ResponseText = xmlhttp.responseText;   //get text data in the response
ResponseXML = xmlhttp.responseXML; //get xml data in the response
ResponseHeaderJSON = xmlhttp.getResponseHeader
                        ("CustomHeaderJSON");  // Extract Data in http header
ResponseHeaders = xmlhttp.getAllResponseHeaders();   //Get a string
                                                //containing all http headers returned by server

// Make all the results available in the ReadyHandler via prototyping.
ReadyHandler.prototype.ResponseText = ResponseText;
ReadyHandler.prototype.ResponseHeaderJSON = ResponseHeaderJSON;
ReadyHandler.prototype.ResponseXML = ResponseXML;
ReadyHandler.prototype.ResponseHeaders = ResponseHeaders;
// Execute function passed as ReadyHandelr
ReadyHandler();
        }
    }

//If querystring is provided Attach it to the url
                        if (QueryString != "") {
varQueryStringData = "";
for (QueryStringAttributeinQueryString) {
QueryStringData = QueryStringAttribute + "=" +
                        QueryString[QueryStringAttribute] + "&" + QueryStringData;
        }
QueryStringData = QueryStringData.substring(0,
                                                QueryStringData.lastIndexOf('&'));
        URL = URL + "?" + escape(QueryStringData);      //Here is where the
                                                //query string ia attached to the request url.
    }

//POST or GET URL of the script to execute.true for asynchronous
//(false for synchronous).
xmlhttp.open(Method, URL, true);
xmlhttp.setRequestHeader("Content-type", "application/x-www-form-urlencoded");
if (HttpHeaders != "") {
varHttpHeadersData = "";
for (HttpHeaderNameinHttpHeaders) {
xmlhttp.setRequestHeader(HttpHeaderName,
                        HttpHeaders[HttpHeaderName]);  // Here the custom headers are added
        }
    }

            //Post data provided then assemble it into single string to be posted to server
            if (Params != "") {
varParamsData = "";
for (ParamNamein Params) {
ParamsData = ParamName + "=" + Params[ParamName] + "&" + ParamsData;
        }
ParamsData = ParamsData.substring(0, ParamsData.lastIndexOf('&'));
    }

xmlhttp.send(ParamsData); //Send the request with the post data
  }
}
```

[You can find the complete implementation with sufficient comments in the source code.]

It can give a more clear idea of using AJAX in your applications.

In the demo application, you can test the 'AjaxRequest' function by changing the parameters that are passed to it.



Actually all the code that is typed in the text box is executed as JavaScript code on click of 'Execute' button.This is done using the eval() function.

JavaScript

```
FunctionCall = document.getElementById('FunctionCode').value;
eval(FunctionCall);
```

Function Usage

JavaScript

```
functionAjaxRequest(ReadyHandler,URL,Method,Params,QueryString,HttpHeaders)
```

Description

->*ReadyHandler*: Function to be called after successful completion of the AJAX request

Note: On successful completion of the request, the result of the request will be available in the function passed as ReadyHandler.
The result of request will be in 4 variables, namely:

- ResponseText: Text response from server
- ResponseHeaderJSON: Custom HTTP Header String value

This header string may contain a single string value or a you can also use a JSON format for multiple values which then can be parsed in ReadyHandler(as shown in the example).

- ResponseHeaders: String containing all Response HTTP Headers
- ResponseXML: XML response from server (XML object available only when the Response contains a proper XML)

->URL: This parameter takes the URL to which the request is to be sent

->Method: Method of request "GET"/"POST"

->Params: POST data to be sent to server. Expects a JSON formatted name value pairs

->QueryString: Data to be sent to the server as QueryString. Expects a JSON formatted name value pairs

->HttpHeaders: Data to be sent as HTTP Headers. Expects JSON formatted name value pairs

Note: While sending the data in headers, you have to take care only ASCII characters where charCode ranging from 32 to 126 are sent or you may get unexpected results. See RFC documentation for HTTP.

The ReadyHandlercan contain the code which will dynamically change the contents of the webpage based on the response data.

For example, in the demo application, I have used 'ProcessRequest()' as the Ready handler which sets the response in the respective <Div>.

JavaScript

```
functionProcessRequest() {

// // Assign the content to the form
document.getElementById('ResponseTextDiv').innerHTML = ResponseText;

document.getElementById('ResponseXMLDiv').innerHTML = ResponseHeaders;
eval("var CustomHeaders = { " + ResponseHeaderJSON + "};");
var header;
varallHeaders = "<br/>";
if (CustomHeaders != "") {
for (header inCustomHeaders) {
allHeaders = allHeaders + CustomHeaders[header] + "<br/>"
      }
   }
document.getElementById('ResponseHeadersDiv').innerHTML = allHeaders;
}
```

Example:

JavaScript

```
AjaxRequest(ProcessRequest, 'Handler.ashx','POST',
              { Param1: 'Param1Value', Param2: 'Param2Value', Param3: 'Param3Value' },
              { Query1: 'Q1', Query2: 'Q2', Query3: 'Q3' },
              { Header1: 'H1', Header2: 'H2', Header3: 'H3' }
                );
```

For handling the client request, I have implemented a simple **Generic Handler (.ashx).**

You can access all the data (query string + Post Data + HTTP Headers) that is sent by the client browser in AJAX request.

In the Generic handler, the data is accessible via the context.Requestobject.

Though you can access all the data together in context.Request.Params[], you can access the data separately as follows:

- Query String: context.Request.QueryString[[index/string]]
- Http Headers: context.Request.Headers[[index/string]]

In the example application, what I have done is just echo back the data which is received in the request along with a custom HTTP header added.

JavaScript

```
foreach(string Param incontext.Request.Params)
 {
ParamsData ="<br/>" + Param + " : " +
            context.Request.Params[Param].ToString() + ParamsData;
 }
context.Response.Write(ParamsData);
```

The above lines capture the data in the request and send it back in the response.

For adding an extra custom HTTP header in response:

C#

```
context.Response.AddHeader("CustomHeaderJSON", CustomHeaderJSON);
```

As you see, the context.Responseobject is used to assemble the response which is to be sent back to the browser.

Different methods of *context.Response* can be used to do this.

'*CustomHeaderJSON*' can contain a string , but I have created a JSON format string for supporting multiple values.The values are then parsed at client side using JavaScript.

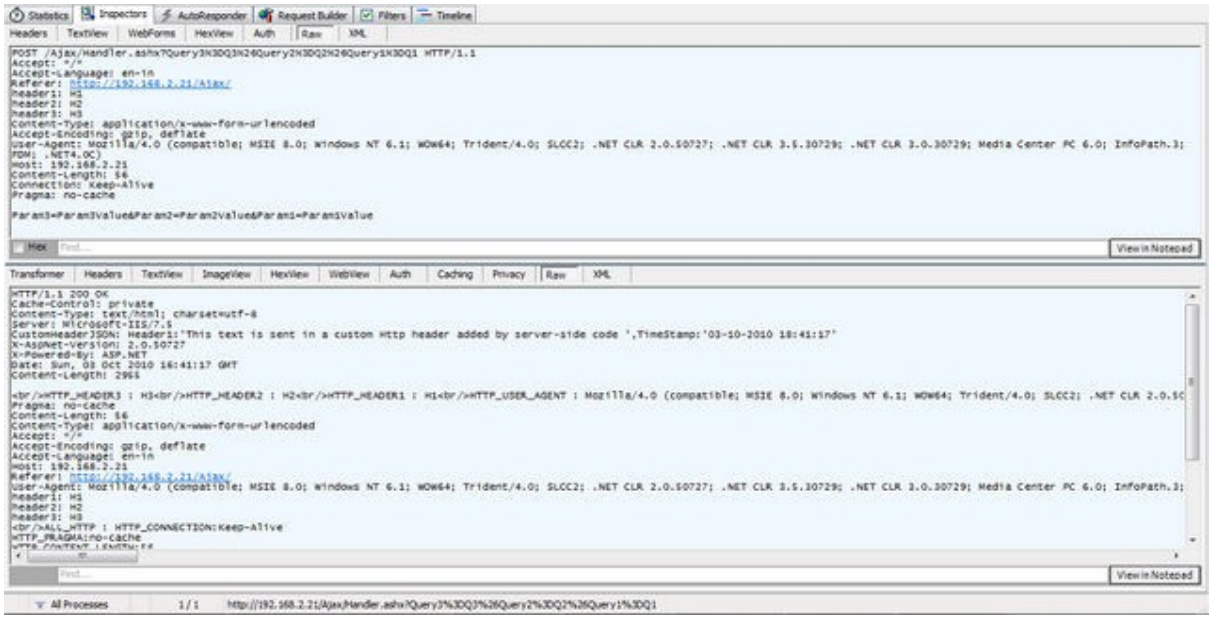
I have just used string concatenate for creating it, but you can also use different JSON parsers/Encoders available at http://www.json.org/.

You can also use JSON strings to exchange data through AJAX. It is sometimes better to use JSON than XML. Using JSON results in less bytes transferred than XML.

Points of Interest

This is a basic implementation of AJAX and the function can be tuned and modified according to needs and reconfigurability.

Here is how the request and response looks like [HTTP request in Fiddler]:



jQuery Events

jQuery events are the actions that can be detected by your web application. They are used to create dynamic web pages. An event shows the exact moment when something happens.

These are some examples of events.

- A mouse click
- An HTML form submission
- A web page loading
- A keystroke on the keyboard
- Scrolling of the web page etc.

These events can be categorized on the basis their types:

Mouse Events

- click
- dblclick
- mouseenter
- mouseleave

Keyboard Events

- keyup
- keydown
- keypress

Form Events

- submit
- change
- blur
- focus

Document/Window Events

- load
- unload
- scroll
- resize

Note: A term "fires" is generally used with events. For example: The click event fires in the moment you press a key.

Syntax for event methods

Most of the DOM events have an equivalent jQuery method. To assign a click events to all paragraph on a page, do this:

1. $("p").click ();

The next step defines what should happen when the event fires. You must pass a function to the event.

UNIT – III

REACT JS

React Introduction

ReactJS is a declarative, efficient, and flexible JavaScript library for building reusable UI components. It is an open-source, component-based front end library responsible only for the view layer of the application. It was created by **Jordan Walke,** who was a software engineer at **Facebook.** It was initially developed and maintained by Facebook and was later used in its products like **WhatsApp**&**Instagram.** Facebook developed ReactJS in **2011** in its newsfeed section, but it was released to the public in the month of **May 2013.**

Today, most of the websites are built using MVC (model view controller) architecture. In MVC architecture, React is the 'V' which stands for view, whereas the architecture is provided by the Redux or Flux.