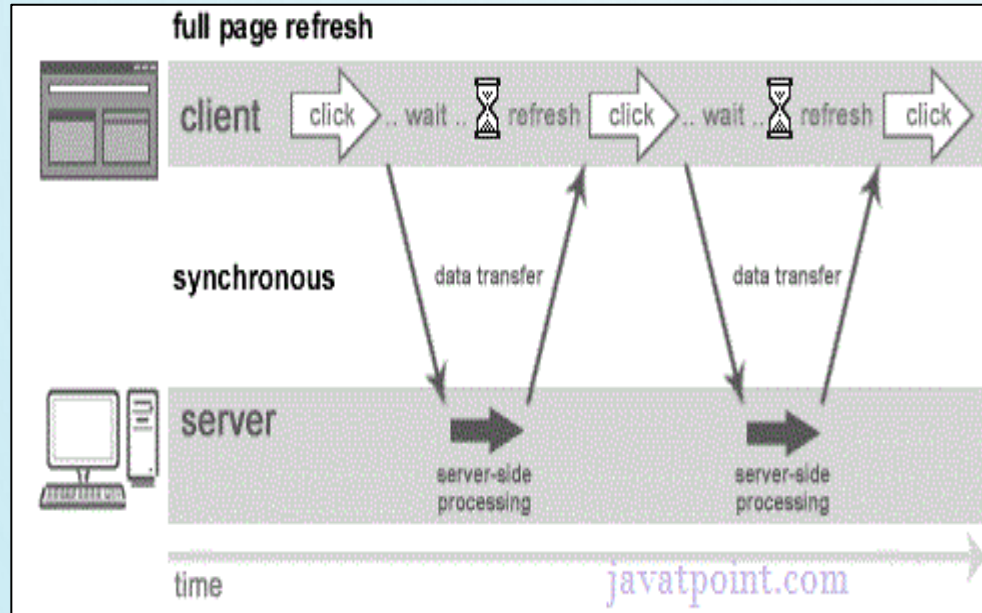


AJAX – INTRODUCTION

- AJAX is an acronym for **Asynchronous JavaScript and XML**.
- It is a group of inter-related technologies like **JavaScript, DOM, XML, HTML/XHTML, CSS, XMLHttpRequest** etc.
- AJAX allows to send and receive data asynchronously without reloading the web page. So it is fast.
- AJAX allows to send only **important information** to the server **not** the **entire page**. So only valuable data from the client side is routed to the server side. It makes the application interactive and faster.

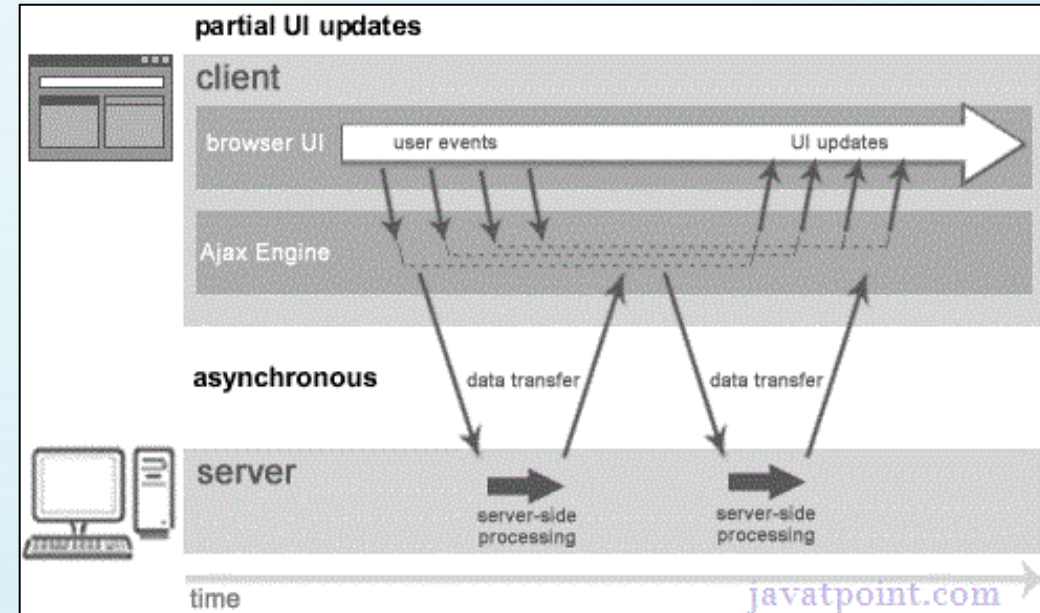
Synchronous and Asynchronous

A **synchronous** request blocks the client until operation completes i.e. browser is unresponsive. In such case, javascript engine of the browser is blocked.



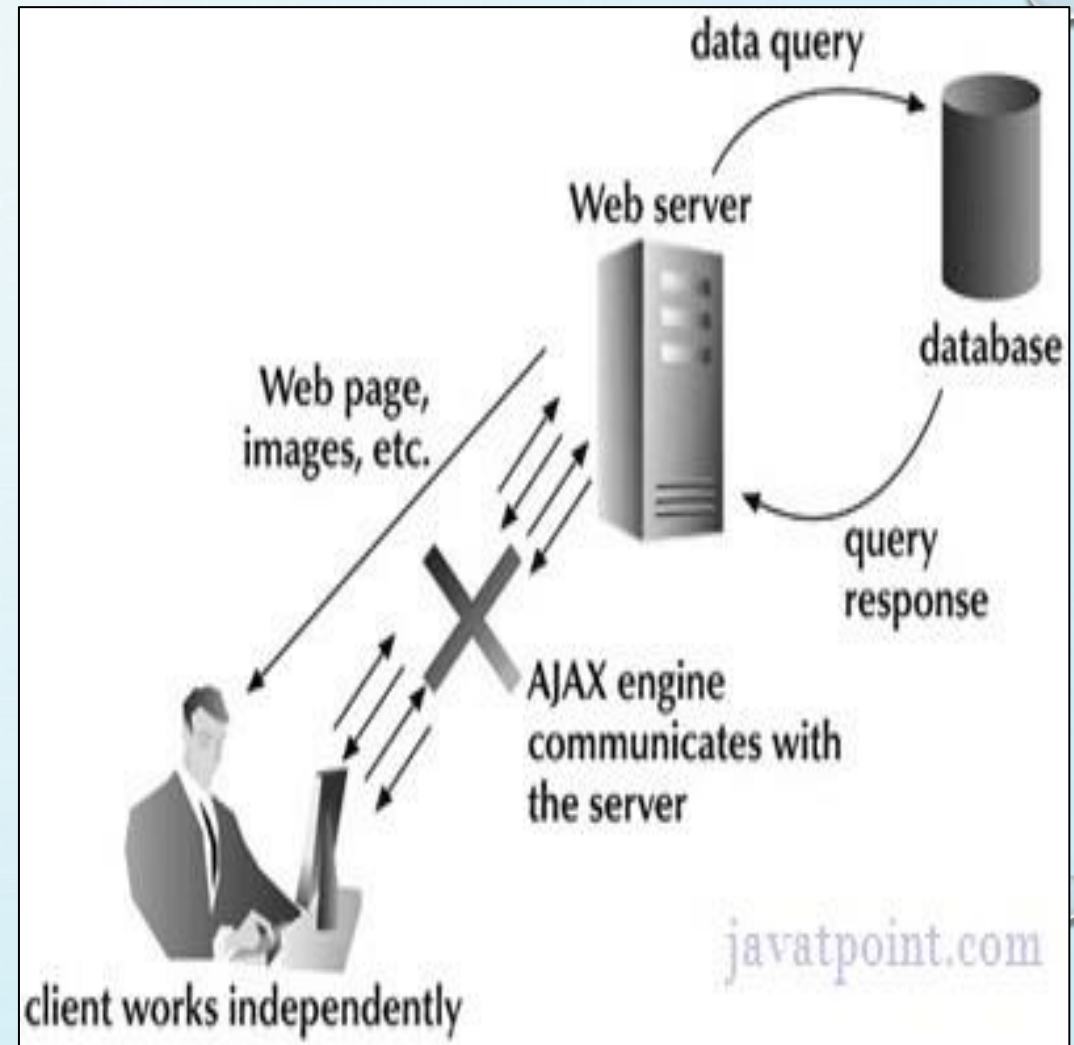
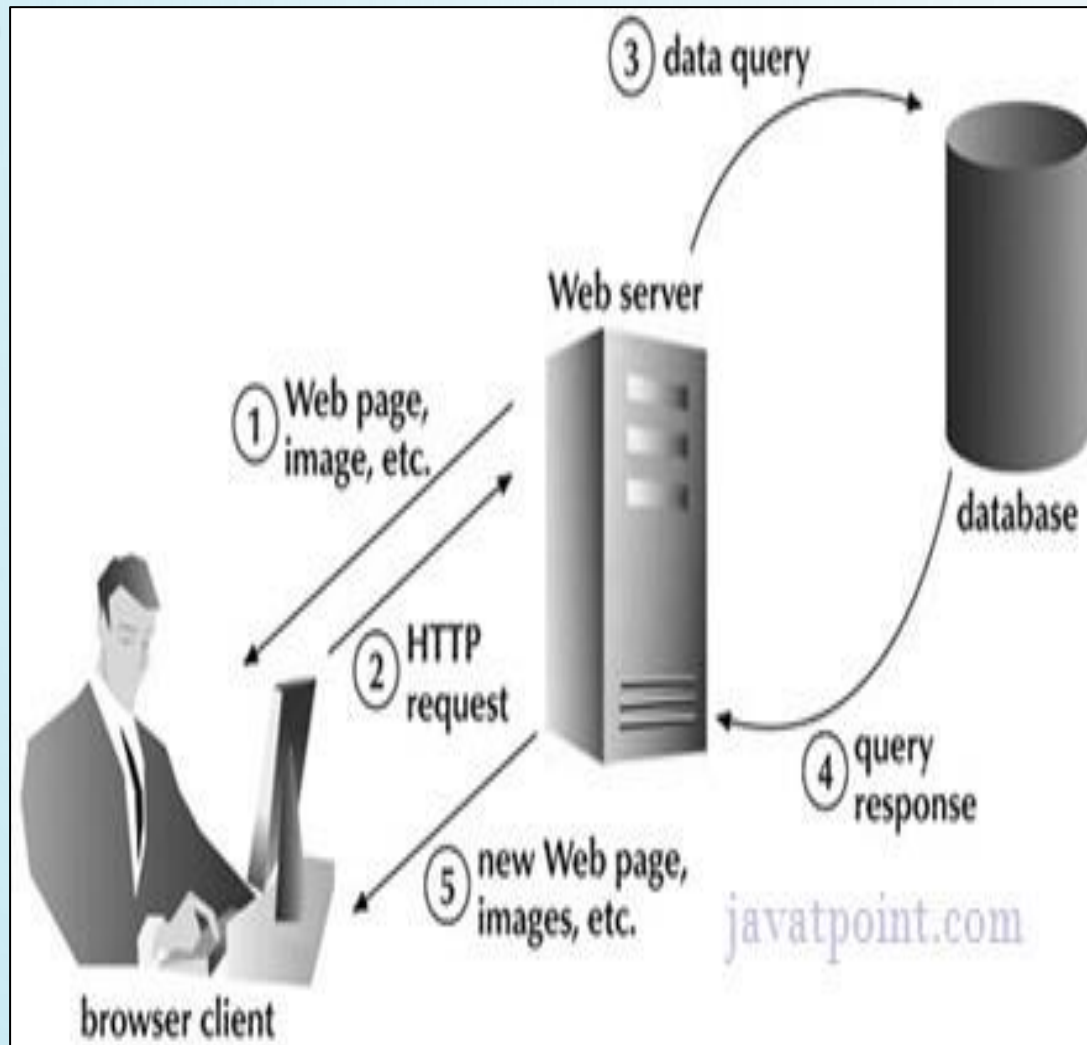
Full page is refreshed at request time and user is blocked until request completes.

An **asynchronous** request doesn't block the client i.e. browser is responsive. At that time, user can perform another operations also. In such case, javascript engine of the browser is not blocked.



Full page is not refreshed at request time and user gets response from the ajax engine.

Synchronous and Asynchronous



AJAX – Technologies

As describe earlier, ajax is not a technology but group of inter-related technologies. AJAX technologies includes:




- **HTML/XHTML** and **CSS** - used for displaying content and style. It is mainly used for presentation.
- **DOM** - used for dynamic display and interaction with data.
- **XML** or **JSON** - For carrying data to and from server. JSON (Javascript Object Notation) is like XML but short and faster than XML.
- **XMLHttpRequest** - For asynchronous communication between client and server. For more visit next page
- **JavaScript** - it is used mainly for client-side validation and to bring the above technologies together.

XML

simple.xml

```
<breakfast_menu>
  <food>
    <name>Belgian Waffles</name>
    <price>$5.95</price>
    <description>Two of our famous Belgian Waffles with plenty of real maple syrup</description>
    <calories>650</calories> </food>
  <food>
    <name>Strawberry Belgian Waffles</name>
    <price>$7.95</price>
    <description>Light Belgian waffles covered with strawberries and whipped cream</description>
    <calories>900</calories> </food>
  <food>
    <name>Berry-Berry Belgian Waffles</name>
    <price>$8.95</price>
    <description>Light Belgian waffles covered with an assortment of fresh berries and whipped cream</description>
    <calories>900</calories> </food>
</breakfast_menu>
```

← → ↻ w3schools.com/xml/simplexml.xml

 Gmail  YouTube  Maps

Belgian Waffles - \$5.95

Two of our famous Belgian Waffles with plenty of real maple syrup *(650 calories per serving)*

Strawberry Belgian Waffles - \$7.95

Light Belgian waffles covered with strawberries and whipped cream *(900 calories per serving)*

Berry-Berry Belgian Waffles - \$8.95

Light Belgian waffles covered with an assortment of fresh berries and whipped cream *(900 calories per serving)*

AJAX – XMLHttpRequest

- An **object** of **XMLHttpRequest** is used for asynchronous communication between client and server, which
 - Sends data from the client in the background
 - Receives the data from the server
 - Updates the webpage without reloading it.

All modern browsers (Chrome, Firefox, IE, Edge, Safari, Opera) have a built-in **XMLHttpRequest** object.

The keystone of AJAX is the XMLHttpRequest object.

Create an XMLHttpRequest object

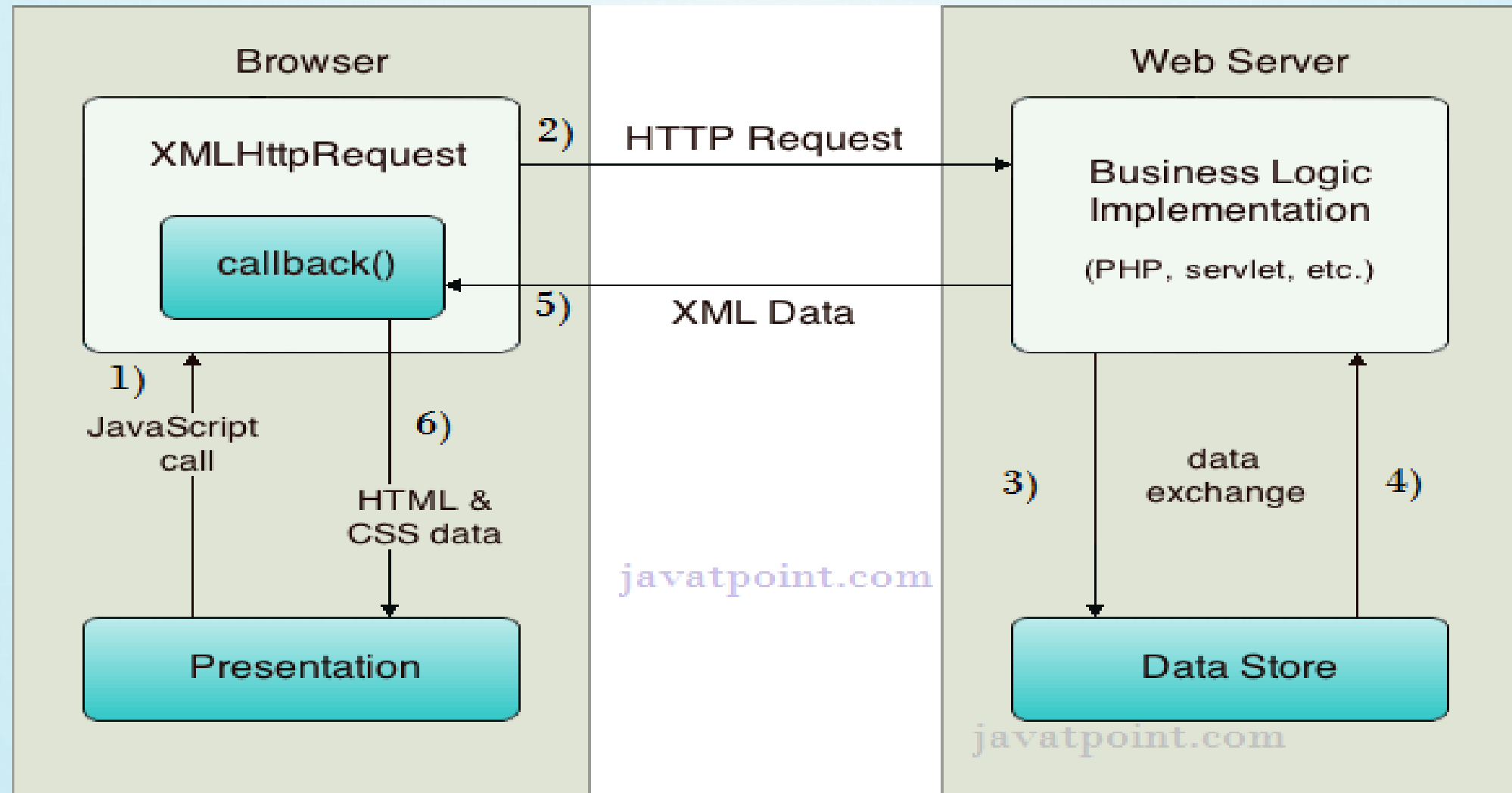
Define a callback function

Open the XMLHttpRequest object

Send a Request to a server

```
variable = new XMLHttpRequest();  
xhr.onload = function() {  
    // What to do when the response is ready  
}  
  
xhr.open("GET", "ajax_info.txt");  
xhr.send();
```

AJAX – XMLHttpRequest Object



AJAX – XMLHttpRequest Object

The **common properties** of XMLHttpRequest object

Property	Description
onReadyStateChange	It is called whenever readystate attribute changes. It must not be used with synchronous requests.
readyState	represents the state of the request. It ranges from 0 to 4. 0 UNOPENED open() is not called. 1 OPENED open is called but send() is not called. 2 HEADERS_RECEIVED send() is called, and headers and status are available. 3 LOADING Downloading data; responseText holds the data. 4 DONE The operation is completed fully.
responseText	returns response as text.
responseXML	returns response as XML

AJAX – XMLHttpRequest Object

The **methods** of XMLHttpRequest object

Method	Description
<code>void open(method, URL)</code>	opens the request specifying get or post method and url.
<code>void open(method, URL, async)</code>	same as above but specifies asynchronous or not.
<code>void open(method, URL, async, username, password)</code>	same as above but specifies username and password.
<code>void send()</code>	sends get request.
<code>void send(string)</code>	send post request.
<code>setRequestHeader(header,value)</code>	it adds request headers.

AJAX – Example1 (onload)

```
<!DOCTYPE html>
<html>
<body>

<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Request data</button>
<p id="demo"></p>

<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("demo").innerHTML = this.responseText;
  }
  xhttp.open("GET", "demo_get.asp");
  xhttp.send();
}
</script>
```

The XMLHttpRequest Object

Request data

The XMLHttpRequest Object

Request data

This content was requested using the GET method.

Requested at: 3/6/2023 2:23:33 AM

AJAX – Example2 (Onload with Id)

```
<!DOCTYPE html>
<html>
<body>

<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Request data</button>

<p id="demo"></p>

<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("demo").innerHTML = this.responseText;
  }
  xhttp.open("GET", "demo_get.asp?t=" + Math.random());
  xhttp.send();
}
</script>
```

The XMLHttpRequest Object

Request data

The XMLHttpRequest Object

Request data

This content was requested using the GET method.

Requested at: 3/6/2023 2:26:31 AM

Here, the request with ID. In the previous example, you may get a cached result. To avoid this, add a unique ID to the URL:

AJAX – onload with query string

```
<!DOCTYPE html>
<html>
<body>

<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Request data</button>

<p id="demo"></p>

<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onload = function() {
    document.getElementById("demo").innerHTML = this.responseText;
  }
  xhttp.open("GET", "demo_get2.asp?fname=Henry&lname=Ford");
  xhttp.send();
}
</script>
```

The XMLHttpRequest Object

Request data

The XMLHttpRequest Object

Request data

Hello Henry Ford

THE ONREADystateCHANGE PROPERTY

Property	Description
onreadystatechange	Defines a function to be called when the readyState property changes
readyState	Holds the status of the XMLHttpRequest. 0: request not initialized 1: server connection established 2: request received 3: processing request 4: request finished and response is ready
status	200: "OK" 403: "Forbidden" 404: "Page not found" For a complete list go to the Http Messages Reference
statusText	Returns the status-text (e.g. "OK" or "Not Found")

The `readyState` property holds the status of the XMLHttpRequest.

The `onreadystatechange` property defines a callback function to be executed when the readyState

The `status` property and the `statusText` properties hold the status of the XMLHttpRequest object

The `onreadystatechange` function is called every time the readyState changes.

When `readyState` is 4 and status is 200, the response is ready:

EXAMPLE

```
<!DOCTYPE html>
<html>
<body>

<div id="demo">
<h2>The XMLHttpRequest Object</h2>
<button type="button" onclick="loadDoc()">Change Content</button>
</div>

<script>
function loadDoc() {
  const xhttp = new XMLHttpRequest();
  xhttp.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML =
        this.responseText;
    }
  };
  xhttp.open("GET", "ajax_info.txt");
  xhttp.send();
}
</script>

</body>
</html>
```

The XMLHttpRequest Object

Change Content

AJAX

AJAX is not a programming language.

AJAX is a technique for accessing web servers from a web page.

AJAX stands for Asynchronous JavaScript And XML.

JSON

- JSON stands for **JavaScript Object Notation**.
- It is an open standard **data-interchange format**.
- It is lightweight and **self describing**
- It is originated from JavaScript.
- **JSON is text**, written with JavaScript object notation.
- It is **easy to read and write than XML**. For AJAX applications, JSON is faster and easier than XML.
- It is language independent (**interoperability**).
- It supports array, object, string, number and values.
- The format was specified by Douglas Crockford.
- The JSON file must be save with **.json extension**.
- The MIME type for JSON text is "**application/ json**"

JSON

- JSON stands for JavaScript Object Notation.
- JSON is lightweight data-interchange format.
- JSON is easy to read and write than XML.
- JSON is language independent.
- JSON supports array, object, string, number and values.

Data Type	Description	Example
String	A string is always written in double-quotes. It may consist of numbers, alphanumeric and special characters.	"student", "name", "1234", "Ver_1"
Number	Number represents the numeric characters.	121, 899
Boolean	It can be either True or False.	true
Null	It is an empty value.	

JSON

```
{"name" : "Jack", "employeeid" : 001, "present" : false}
```

```
[{  
  "PizzaName" : "Country Feast",  
  "Base" : "Cheese burst",  
  "Toppings" : ["Jalepenos", "Black Olives", "Extra cheese", "Sausages", "Cherry tomatoes"],  
  "Spicy" : "yes",  
  "Veg" : "yes"  
},  
  
{  
  "PizzaName" : "Veggie Paradise",  
  "Base" : "Thin crust",  
  "Toppings" : ["Jalepenos", "Black Olives", "Grilled Mushrooms", "Onions", "Cherry tomatoes"],  
  "Spicy" : "yes",  
  "Veg" : "yes"  
}  
]
```

JSON

```
[  
  {  
    "PizzaName": "Country Feast",  
    "Base": "Cheese burst",  
    "Toppings": [  
      "Jalepenos",  
      "Black Olives",  
      "Extra cheese",  
      "Sausages",  
      "Cherry tomatoes"  
    ],  
    "Spicy": "yes",  
    "Veg": "yes"  
  },  
  {  
    "PizzaName": "Veggie Paradise",  
    "Base": "Thin crust",  
    "Toppings": [  
      "Jalepenos",  
      "Black Olives",  
      "Grilled Mushrooms",  
      "Onions",  
      "Cherry tomatoes"  
    ],  
    "Spicy": "yes",
```


JSON AND XML

- JSON and XML are human readable formats and are language independent. They both have support for creation, reading and decoding in real world situations.

- **JSON**

```
{ "car": { "company": "Volkswagen", "name": "Vento", "price": 800000 } }
```

- **XML**

```
<car>
```

```
<company>Volkswagen</company>
```

```
<name>Vento</name>
```

```
<price>800000</price>
```

```
</car>
```

A SYNTAX FOR STORING AND EXCHANGING DATA

- When exchanging data between a browser and a server, the data can only be text.
- JSON is text, and we can convert any JavaScript object into JSON, and send JSON to the server.
- We can also convert any JSON received from the server into JavaScript objects.
- This way we can work with the data as JavaScript objects, with no complicated parsing and translations.
- If you have data stored in a JavaScript object, you can convert the object into JSON, and send it to a server using `JSON.stringify()`.
- If you receive data in JSON format, you can convert it into a JavaScript object using `JSON.parse()`.

USES OF JSON

- It is used while **writing JavaScript based applications** that includes browser extensions and websites.
- JSON format is used for serializing and transmitting structured data over network connection.
- It is primarily used to **transmit data between a server and web applications.**
- **Web services and APIs use JSON** format to provide public data.
- JSON uses JavaScript syntax, but the JSON format is text only. Text can be read and used as a data format by any programming language.

```
{  
  "Title": "The Cuckoo's Calling"  
  "Author": "Robert Galbraith",  
  "Genre": "classic crime novel",  
  "Detail": {  
    "Publisher": "Little Brown"  
    "Publication_Year": 2013,  
    "ISBN-13": 9781408704004,  
    "Language": "English",  
    "Pages": 494  
  }  
  "Price": [  
    {  
      "type": "Hardcover",  
      "price": 16.65,  
    }  
    {  
      "type": "Kindle Edition",  
      "price": 7.03,  
    }  
  ]  
}
```

Object Starts

Object Starts

Value string

Value number

Object ends

Array starts

Object Starts

Object ends

Object Starts

Object ends

Array ends

Object ends

EXAMPLE 1 – TO STORE EMPLOYEE DATA - FIRST.JSON

```
{ "employees": [  
  {   "name": "Sonoo",  
      "email": "sonoojaiswal1987@gmail.com" },  
  {   "name": "Rahul",  
      "email": "rahul32@gmail.com" },  
  {   "name": "John",  
      "email": "john32bob@gmail.com" }  
  ]  
}
```


EXAMPLE 2 – TO STORE BOOK DATA – BOOK.JSON

```
        { "BOOK": [  
            {  
                "id": "01",  
                "language": "Java",  
                "edition": "third",  
                "author": "Herbert Schildt"            },  
  
            {  
                "id": "07",  
                "language": "C++",  
                "edition": "second",  
                "author": "E.Balagurusamy"            }  
        ] }
```

JSON SYNTAX

- Data is represented in name/value pairs. JSON names require double quotes. JavaScript names don't.
- Curly braces hold objects and each name is followed by ':'(colon), the name/value pairs are separated by , (comma).

```
{  
  "employee": { "name": "sonoo", "salary": 56000, "married": true }  
}
```

- Square brackets hold arrays and values are separated by ,(comma).

1. ["Sunday", "Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday"]

→ Values in an Array

2. [{ "name": "Ram", "email": "Ram@gmail.com" },
 { "name": "Bob", "email": "bob32@gmail.com" }
]

→ Objects in an Array

JSON DATATYPES

- Number

- Integer (0-9 ,+ ve,-ve), Fraction, Exponent.
- Octal and hexadecimal formats are not used.
- No NaN or Infinity is used in Number.

```
{  
  "integer": 34,  
  "fraction": .2145, "exponent":  
  6.61789e+0  
}
```

- String → It is a sequence of zero or more double quoted Unicode characters with backslash escaping

Eg: “aaa ”,’aaa’,\ n,\ t

JSON DATATYPES

- Boolean → true, false
- Array → an ordered collection of values
- Value – String, number, true or false, null etc
- Object – unordered collection of key:value pairs
- Whitespace – can be used between any pair of tokens. It can be added to make a code more readable.

```
var obj1 = { "name": "Sachin Tendulkar" }
```

```
var obj2 = { "name": "Saurav Ganguly" }
```

- null → empty type `var i = null;`
- JSON values **cannot** be one of the following data types: a function, a date, *undefined*

ACCESSING OBJECT VALUES

```
myObj = { "name":"John", "age":30, "car":null };
```

➤ `x = myObj.name;` → **Output: John**

➤ `x = myObj["name"];` □ **Output: John**

➤ `for (x in myObj) {
 document.getElementById("demo").innerHTML += x;
}`
Output: name age

➤ `for (x in myObj) {
 document.getElementById("demo").innerHTML += myObj[x];
}`
Output:
John 30
null

EXAMPLE.HTML

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

```
<p>Use bracket notation to access the property values.</p>
```

```
<p id="demo"></p>
```

```
<script>
```

```
var myObj = {"name":"John", "age":30, "car":null};
```

```
for (x in myObj) {
```

```
    document.getElementById("demo").innerHTML += myObj[x] + "<br>";}
```

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



NESTED OBJECTS

<html> <body> <p>How to access nested JSON objects.</p>

<p id="demo"></p><script> var myObj = {
 "name":"John", "age":30,
 "cars": {
 "car1":"Ford",
 "car2":"BMW",
 "car3":"Fiat" } }

How to access nested JSON objects.

BMW
BMW

document.getElementById("demo").innerHTML += myObj.cars.car2 + "
"; //or:

document.getElementById("demo").innerHTML += myObj.cars["car2"];

</script> </body></html>

Modify Values

- myObj.cars.car2 = "Mercedes"; // or
- myObj.cars["car2"] = "Mercedes";

Delete Object Properties

delete myObj.cars.car2;

How to delete properties of a JSON object.

Ford
Fiat

```
<p id="demo"></p>

<script>
var myObj, i, x = "";
myObj = {
  "name": "John",
  "age": 30,
  "cars": {
    "car1": "Ford",
    "car2": "BMW",
    "car3": "Fiat"
  }
}
delete myObj.cars.car2;

for (i in myObj.cars) {
  x += myObj.cars[i] + "<br>";
}

document.getElementById("demo").innerHTML = x;

</script>
```

ARRAYS IN JSON OBJECTS

<body> <p>Looping through an array using a for in loop:</p>

<p id="demo"></p> <script>

var myObj, i, x = "";

myObj = {

"name": "John", "age": 30,

"cars": ["Ford", "BMW", "Fiat"] };

for (i in myObj.cars) {

**x += myObj.cars[i] + "
";}**

document.getElementById("demo").innerHTML = x;

</script></body></html>

Looping through an array using a for in loop:

Ford
BMW
Fiat

```
for (i = 0; i < myObj.cars.length; i++) {  
  x += myObj.cars[i];  
}
```


NESTED ARRAYS IN JSON OBJECTS

- Values in an array can also be another array, or even another JSON object:

- ```
myObj = {
 "name": "John", "age": 30,
 "cars": [
 { "name": "Ford", "models": ["Fiesta", "Focus", "Mustang"] },
 { "name": "BMW", "models": ["320", "X3", "X5"] },
 { "name": "Fiat", "models": ["500", "Panda"] }
]
}
```



- **Accessing:**

```
for (i in myObj.cars) {
 x += "<h1>" + myObj.cars[i].name + "</ h1>";
 for (j in myObj.cars[i].models) {
 x += myObj.cars[i].models[j];
 }
}
```

---

- **Modify using index:**

```
myObj.cars[1].name= "Mercedes";
```

- **Delete using index:**

```
delete myObj.cars[1];
```

A COMMON USE OF JSON IS TO EXCHANGE DATA TO/FROM A WEB SERVER. WHEN SENDING DATA TO A WEB SERVER, THE DATA HAS TO BE A STRING. CONVERT A JAVASCRIPT OBJECT INTO A STRING WITH JSON.STRINGIFY()

```
<html> <body>
<h2>Create JSON string from a JavaScript object.</ h2>
```

```
<p id="demo"></p><script>
```

```
var obj = { "name": "John", "age": 30, "city": "New York" };
```

```
var myJSON = JSON.stringify(obj);
```

```
document.getElementById("demo").innerHTML = myJSON;
```

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

**Create JSON string from a JavaScript object.**

```
{"name": "John", "age": 30, "city": "New York" }
```

A common use of JSON is to exchange data to/from a web server. When receiving data from a web server, the data is always a string. Parse the data with `JSON.parse()`, and the data becomes a JavaScript object.

```
<html> <body><h2>Create Object from JSON String</h2>
```

```
<p id="demo"></p><script>
```

```
var txt = '{"name":"John", "age":30, "city":"New Yrk"}'
```

```
var obj = JSON.parse(txt);
```

```
document.getElementById("demo").innerHTML = obj.name + ", " + obj.age;
```

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

**Create Object from JSON String**

John, 30

```
<!DOCTYPE html>
<html>
<body>
<h2>Convert a string into a date object.</h2>
<p id="demo"></p>
<script>
var text = '{"name":"John", "birth":"1986-12-14", "city":"New York"}';
var obj = JSON.parse(text, function (key, value) {
 if (key == "birth") {
 return new Date(value);
 } else {
 return value;
 }
});
document.getElementById("demo").innerHTML = obj.name + ", " + obj.birth;
</script>
</body>
</html>
```

**Convert a string into a date object.**

John, Sun Dec 14 1986 05:30:00 GMT+0530 (India Standard Time)

# JSON TOOLS

- JSON Editors
- JSON Parser
- JSONViewer
- JSONTree Viewer Tool



```
1 - {
2 "array": [
3 1,
4 2,
5 3
6],
7 "boolean": true,
8 "color": "#82b92c",
9 "null": null,
10 "number": 123,
11 - "object": {
12 "a": "b",
13 "c": "d",
14 "e": "f"
15 },
16 "string": "Hello World"
17 }
```

Ln: 1 Col: 1

```
Select a node...
 ▣ ▼ object {7}
 ... ▣ ▶ array [3]
 ... ▣ boolean : ☒ true
 ... ▣ color :  #82b92c
 ... ▣ null : null
 ... ▣ number : 123
 ... ▣ ▶ object {3}
 ... ▣ string : Hello World
```

# Json viewer

Url:

Get JSON and Parse (IE only)

## RAW json data:

```
{ "book": [
 { "name": "aa", "price": 22.45 },
 { "name": "bb", "price": 12 }
]}
```

Parse JSON data

## Json Object view

+ Array	book[2]	
+ Object	book[0]	
String	name	aa
Number	price	22.45
+ Object	book[1]	
String	name	bb
Number	price	12

**JSON FROM THE SERVER :** YOU CAN REQUEST JSON FROM THE SERVER BY USING AN AJAX REQUEST. AS LONG AS THE RESPONSE FROM THE SERVER IS WRITTEN IN JSON FORMAT, YOU CAN PARSE THE STRING INTO A JAVASCRIPT OBJECT.

```
<html><body>
<h2>Use the XMLHttpRequest to get the content of a file.</h2>
<p>The content is written in JSON format, and can easily be converted into a JavaScript
object.</p>
<p id="demo"></p>
<script>
var xmlhttp = new XMLHttpRequest();
xmlhttp.onreadystatechange = function() {
 if (this.readyState == 4 && this.status == 200) {
 var myObj = JSON.parse(this.responseText);
 document.getElementById("demo").innerHTML = myObj.name;
 }
};
xmlhttp.open("GET", "json_demo.txt", true);
xmlhttp.send();
</script>
<p>Take a look at json_demo.txt</p>
</body></html>
```

## **Use the XMLHttpRequest to get the content of a file.**

The content is written in JSON format, and can easily be converted into a JavaScript object.

John

Take a look at [json\\_demo.txt](#)



S.NO	JSON	Javascript
1	Strings in JSON must be written in double quotes. Eg: “name”：“John”	JavaScript names don't.
2	<i>values</i> must be one of the following data types: a string,a number an object (JSON object),an array,a boolean and null	values can be all of the JSON value types, plus any other valid JavaScript expression, including: a function, a date, undefined



# EXERCISE

- Develop a json program for creating 20 students personal information using the following fields as a inputs and print the outputs in table format,student name,  
date\_of\_birth(date,month,year),parents(fathers\_name,mothers\_name),  
blood\_group,email,phone(landline,mobile),address(door\_no  
,street\_name,place\_name,pincode),degree(ug,pg and  
others),employee(self or organization).