**Ajax-Enabled Rich Internet Applications**

**Introduction:**

**Rich Internet Application**

A Rich Internet Application (RIA) is a Web application with many of the same features and appearances as a desktop application.

A RIA requires a browser, browser plug-in or virtual machine to deliver a user application. Data manipulation is handled by the server, and user interface and related object manipulation are handled by the client machine.

RIAs have two key attributes—performance and a rich GUI.

RIA performance comes from Ajax (Asynchronous JavaScript and XML), which uses client-side scripting to make web applications more responsive. Ajax applications separate client-side user interaction and server communication, and run them in parallel, reducing the delays of server-side processing normally experienced by the user.

There are many ways to implement Ajax functionality.

1. “Raw” Ajax uses JavaScript to send asynchronous requests to the server, then updates the page using the DOM.

“Raw” Ajax is best suited for creating small Ajax components that asynchro- nously update a section of the page.

1. Ajax toolkits, such as Dojo ,Prototype, Script.aculo.us and ASP.NET Ajax, which provide powerful ready-to-use controls and functions that enrich web applications, and simplify JavaScript coding by making it cross-browser compatible.

The client-side of Ajax applications is written in XHTML and CSS, and uses JavaScript to add functionality to the user interface. XML is used to structure the data passed between the server and the client.

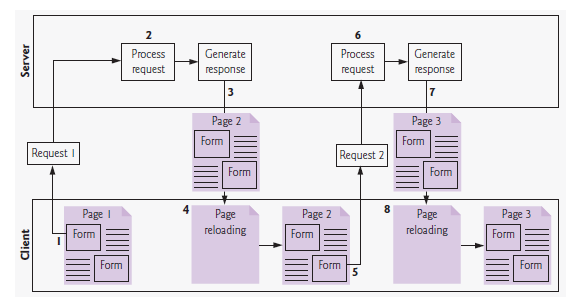
The Ajax component that manages interaction with the server is usually implemented with JavaScript’s XMLHttpRequest object—commonly abbreviated as XHR. The server processing can be implemented using any server-side technology, such as PHP, ASP. NET, Java Server pages , Java Server Faces (JSF) components.

**Traditional Web Applications vs. Ajax Applications**

### Traditional Web Applications:

The typical interactions between the client and the server in a traditional

web application, such as one that uses a user registration form is as follows:

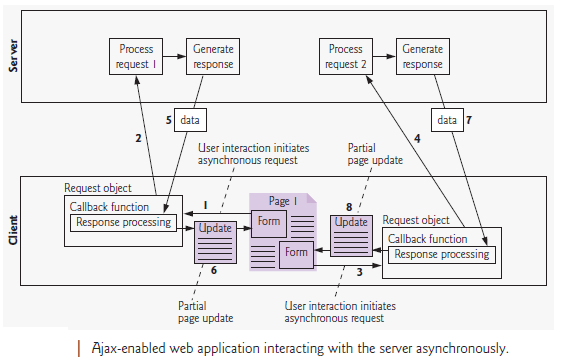


* First, the user fills in the form’s fields, then submits the form.
* The browser generates a request to the server, which receives the request and processes it .
* The server generates and sends a response containing the exact page that the browser will render which causes the browser to load the new page and temporarily makes the browser window blank. The client *waits* for the server to respond and *reloads the entire page* with the data from the response. While such a **synchronous request** is being processed on the server, the user cannot interact with the client web page.

Frequent long periods of waiting, due perhaps to Internet congestion, have led some users to refer to the World Wide Web as the “World Wide Wait.” If the user interacts with and submits another form, the process begins again.

**Ajax Web Applications:**

The typical interactions between the client and the server in a Ajax web application, such as one that uses a user registration form is as follows:



* Ajax applications add a layer between the client and the server to manage communication.
* When the user interacts with the page, the client creates an XMLHttpRequest object to manage a request.
* The XMLHttpRequest object sends the request to the server and awaits the response.
* The requests are **asynchronous**, so the user can continue interacting with the application on the client-side while the server processes the earlier request concurrently. Other user interactions could result in additional requests to the server.
* Once the server responds to the original request the XMLHttpRequest object that issued the request calls a client-side function to process the data returned by the server. This function known as a **callback function**.
* **callback function** uses **partial page updates** to display the data in the existing web page *without reloading the entire page*. At the same time, the server may be responding to the second request
* and the client-side may be starting to do another partial page update .

The callback function updates only a designated part of the page. Such partial page updates

help make web applications more responsive, making them feel more like desktop

applications. The web application does not load a new page while the user interacts with it.

**Rich Internet Applications (RIAs) with Ajax**

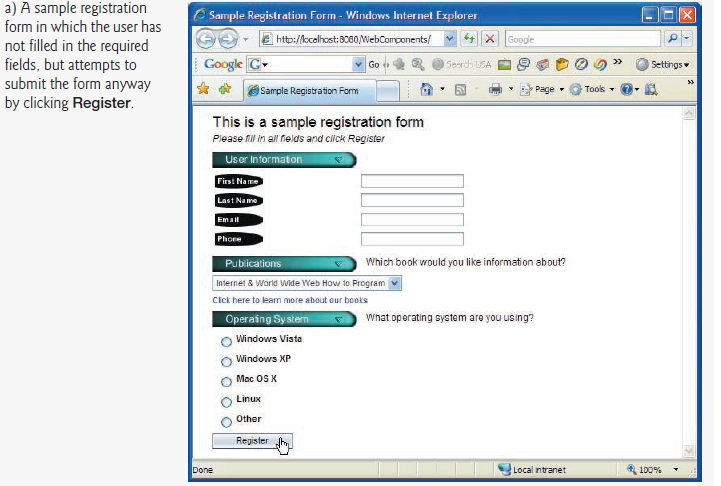
Ajax improves the user experience by making interactive web applications more responsive.

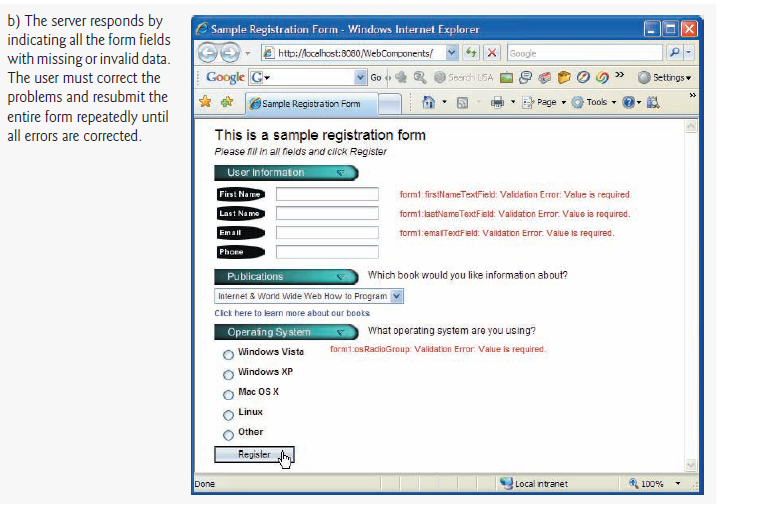
Example:

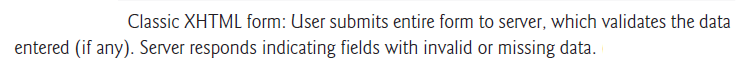
Consider a registration form with a number of fields (e.g., first name, last name email

address, telephone number, etc.) and a **Register** (or **Submit**) button that sends the entered data to the server.

When the user clicks **Register**, a classic XHTML form sends the server all of the data to be validated.







While the server is validating the data, the user cannot interact with the page. The server finds invalid data, generates a new page identifying the errors in the form and sends it back to the client.

Once the user fixes the errors and clicks the **Register** button, the cycle repeats until no errors are found, then the data is stored on the server. The entire page reloads every time the user

submits invalid data.

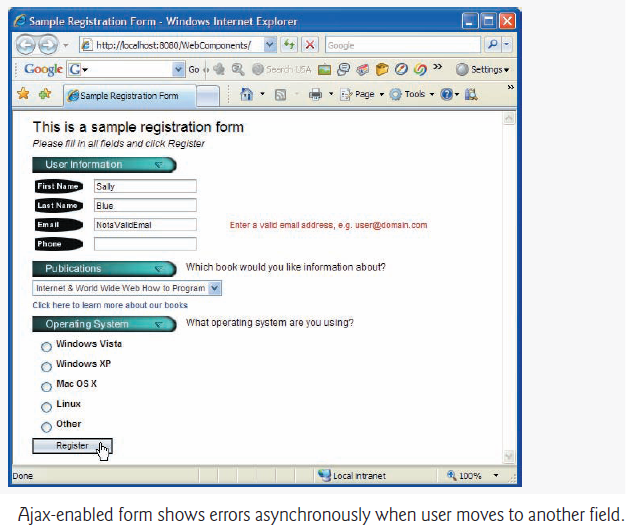
Ajax-enabled forms are more interactive. Rather than sending the entire form to be

validated, entries are validated dynamically as the user enters data into the fields.

For example:

consider a website registration form that requires a unique e-mail address. When the user enters an e-mail address into the appropriate field, then moves to the next form field to continue entering data, an asynchronous request is sent to the server to validate the e-mail address. If the

e- mail address is not unique, the server sends an error message that is displayed on the page informing the user of the problem.



By sending each entry asynchronously, the user can address each invalid entry quickly, versus making edits and resubmitting the entire form repeatedly until all entries are valid. Asynchronous requests could also be used to fill some fields based on previous fields.

**History of Ajax**

The term Ajax was coined by Jesse James Garrett of Adaptive Path in February 2005.

In the 1990s, Netscape’s LiveScript made it possible to include scripts in web pages (e.g., web forms) that could run on the client. LiveScript evolved into JavaScript. In 1998, Microsoft introduced

The XMLHttpRequest object to create and manage asynchronous requests and responses.

Popular applications like Flickr and Google’s Gmail use the XMLHttpRequest object to update pages dynamically.

For example, Flickr uses the technology for its text editing, tagging and organizational features; Gmail continuously checks the server for new e-mail; and Google Maps allows you to drag a map in any direction, downloading the new areas on the map without reloading the entire page.

The name Ajax immediately caught on and brought attention to its component technologies.

Ajax has become one of the hottest web-development technologies, enabling webtop applications to challenge the dominance of established desktop applications.

**Raw Ajax Example using the XMLHttp Request Object**

XMLHttpRequest object is used to create and manage asynchronous requests (Asynchronous - means that if we start some request (call some API), we can move on to another task before that request is finished.). The XMLHttpRequest object (which resides on the client) is the layer between the client and the server that manages asynchronous requests in Ajax applications.

XMLHttpRequest (XHR) is an API that can be used by JavaScript, JScript, VBScript, and other web browser scripting languages to transfer and manipulate XML data to and from a webserver using HTTP, establishing an independent connection channel between a webpage's Client-Side and Server-Side.

The data returned from XMLHttpRequest calls will often be provided by back-end databases. Besides XML, XMLHttpRequest can be used to fetch data in other formats, e.g. JSON or even plain text.

## XMLHttpRequest Methods

* **abort()**

Cancels the current request.

* **getAllResponseHeaders()**

Returns the complete set of HTTP headers as a string.

* **getResponseHeader( headerName )**

Returns the value of the specified HTTP header.

* **open( method, URL )**
* **open( method, URL, async )**
* **open( method, URL, async, userName )**
* **open( method, URL, async, userName, password )**

Specifies the method, URL, and other optional attributes of a request.

The method parameter can have a value of "GET", "POST", or "HEAD". Other HTTP methods such as "PUT" and "DELETE" (primarily used in REST applications) may be possible.

The "async" parameter specifies whether the request should be handled asynchronously or not. "true" means that the script processing carries on after the send() method without waiting for a response, and "false" means that the script waits for a response before continuing script processing.

* **send( content )**

Sends the request.

* **setRequestHeader( label, value )**

Adds a label/value pair to the HTTP header to be sent.

## XMLHttpRequest Properties

* **onreadystatechange**

An event handler for an event that fires at every state change.

* **readyState**

The readyState property defines the current state of the XMLHttpRequest object.

The following table provides a list of the possible values for the readyState property –

|  |  |
| --- | --- |
| **State** | **Description** |
| 0 | The request is not initialized. |
| 1 | The request has been set up. |
| 2 | The request has been sent. |
| 3 | The request is in process. |
| 4 | The request is completed. |

**readyState = 0** After you have created the XMLHttpRequest object, but before you have called the open() method.

**readyState = 1** After you have called the open() method, but before you have called send().

**readyState = 2** After you have called send().

**readyState = 3** After the browser has established a communication with the server, but before the server has completed the response.

**readyState = 4** After the request has been completed, and the response data has been completely received from the server.

* **responseText**

Returns the response as a string.

* **responseXML**

Returns the response as XML. This property returns an XML document object, which can be examined and parsed using the W3C DOM node tree methods and properties.

* **status**

Returns the status as a number (e.g., 404 for "Not Found" and 200 for "OK").

* **statusText**

Returns the status as a string (e.g., "Not Found" or "OK").

# AJAX - The XMLHttpRequest Object

 To use AJAX in JavaScript we need to do four things. Those are:

1. create a XMLHttpRequest object
2. write the callback function
3. open the request
4. send the request

## Create an XMLHttpRequest Object

Syntax :

*variable*= new XMLHttpRequest();

## Define a Callback Function

A callback function is a function passed as a parameter to another function.

 The callback function should contain the code to execute when the response is ready.

xhttp.onload = function() {  
  // What to do when the response is ready  
}

## Send a Request

To send a request to a server, we can use the open() and send() methods of the XMLHttpRequest object.

xhttp.open("GET", "filename");  
xhttp.send();

Example 1:

Html File:

<body>

<div id="demo">

<h1>The XMLHttpRequest Object</h1>

<button type="button" onclick="loadDoc()">Change Content</button>

</div>

<script>

function loadDoc() {

var 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", true);

xhttp.send();

}

</script>

</body>

**ajax\_info.txt**

With the XMLHttpRequest object we can update parts of a web page, without reloading the whole page.

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

Output:



Click the button (Change Content)to execute ajax\_info.txt file.

****

Example 2:

Html File:

<html>

<head>

<title>Simple Ajax Example</title>

<script language="Javascript">

function postRequest(strURL) {

var xmlHttp;

if (window.XMLHttpRequest) { // Mozilla, Safari, ...

var xmlHttp = new XMLHttpRequest();

}else if (window.ActiveXObject) { // IE

var xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");

}

xmlHttp.open('POST', strURL, true);

xmlHttp.setRequestHeader

('Content-Type', 'application/x-www-form-urlencoded');

xmlHttp.onreadystatechange = function() {

if (xmlHttp.readyState == 4) {

updatepage(xmlHttp.responseText);

}

}

xmlHttp.send(strURL);

}

function updatepage(str){

document.getElementById("result").innerHTML =

"<font color='red' size='5'>" + str + "</font>";;

}

function SayHello(){

var usr=window.document.f1.username.value;

var rnd = Math.random();

var url="sayhello.php?id="+rnd +"&usr="+usr;

postRequest(url);

}

</script>

</head>

<body>

<h1 align="center"><font color="#000080">Simple Ajax Example</font></h1>

<p align="center"><font color="#000080">Enter your name and then press

"Say Hello Button"</font></p>

<form name="f1">

<p align="center"><font color="#000080">&nbsp;

Enter your name: <input type="text" name="username" id="username">

<input value="Say Hello" type="button"

onclick='JavaScript:SayHello()' name="showdate"></font></p>

<div id="result" align="center"></div>

</form>

<div id=result></div>

</body>

</html>

**PHP File:**

**<?php**

**$usr=$\_GET["usr"];**

?>

<p>Welcome <?=$usr?>!</p>

<p>Request received on:

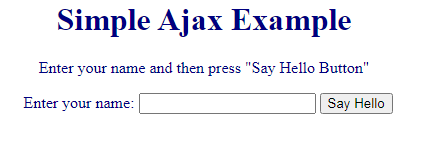
<?php

print date("l M dS, Y");

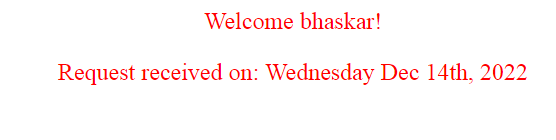
?>

</p>

Output:



Enter Name and Click the button Say Hello



**Using XML and the DOM**

Ajax applications use XML to pass structured data between the server and the client because it is easy to generate, consumes little bandwidth and also to parse.

When the XMLHttpRequest object receives XML data, it parses and stores the data as an XML DOM object in the responseXML property.

• The XMLHttpRequest object’s responseXML property contains the XML returned by the server.

• DOM method createElement creates an XHTML element of the specified type.

• DOM method setAttribute adds or changes an attribute of an XHTML element.

• DOM method appendChild inserts one XHTML element into another.

• The innerHTML property of a DOM element can be used to obtain or change the XHTML that is displayed in a particular element.

Example:

**HTML File:**

<html>

<style>

table,th,td {

border : 1px solid black;

border-collapse: collapse;

}

th,td {

padding: 5px;

}

</style>

<body>

<h1> Students Enrolled For Paper Presentation </h1>

<button type="button" onclick="loadDoc()">Get Details</button>

<br><br>

<table id="demo"></table>

<script>

function loadDoc() {

var xhttp = new XMLHttpRequest();

xhttp.onreadystatechange = function() {

if (this.readyState == 4 && this.status == 200) {

myFunction(this);

}

};

xhttp.open("GET", "student\_details.xml", true);

xhttp.send();

}

function myFunction(xml) {

var i;

var xmlDoc = xml.responseXML;

var table="<tr><th>Student Name</th><th>Branch</th></tr>";

var x = xmlDoc.getElementsByTagName("student");

for (i = 0; i <x.length; i++) {

table += "<tr><td>" +

x[i].getElementsByTagName("Name")[0].childNodes[0].nodeValue +

"</td><td>" +

x[i].getElementsByTagName("Branch")[0].childNodes[0].nodeValue +

"</td></tr>";

}

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

}

</script>

</body>

</html>

**XML File:**

<data>

<student>

<Name>Bhaskar</Name>

<RollNo>501</RollNo>

<Branch>CSE</Branch>

</student>

<student>

<Name>Vijay</Name>

<RollNo>456</RollNo>

<Branch>ECE</Branch>

</student>

<student>

<Name>Ajay</Name>

<RollNo>204</RollNo>

<Branch>EEE</Branch>

</student>

<student>

<Name>Ramu</Name>

<RollNo>428</RollNo>

<Branch>ECE</Branch>

</student>

<student>

<Name>Sunil</Name>

<RollNo>524</RollNo>

<Branch>CSE</Branch>

</student>

<student>

<Name>Lokesh</Name>

<RollNo>236</RollNo>

<Branch>EEE</Branch>

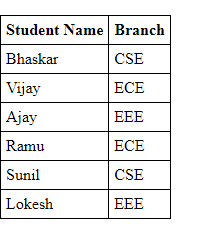
</student>

</data>

**Output:**



Click the Button(Get Details) to get Table.



Creating a full-Scale Ajax-Enabled Application:

# JSON

JSON or JavaScript Object Notation is a lightweight text-based open standard designed for human-readable data interchange.

JSON is built on two structures:

* A collection of name/value pairs. In various languages, this is realized as an *object*, record, struct, dictionary, hash table, keyed list, or associative array.
* An ordered list of values. In most languages, this is realized as an *array*, vector, list, or sequence.

## Functions for Working with JSON

### JSON.stringify()

TheJSON.stringify() function converts an object to a JSON string.

### JSON.parse()

The JSON.parse() method **parses a JSON string, constructing the JavaScript value or object described by the string**.

Example:

data.json

{"name": "dhoni", "country": "India"}

Sample.html

<html> <head>

<script >

function loadJSON() {

var data\_file = "data.json";

var http\_request = new XMLHttpRequest();

http\_request.onreadystatechange = function() {

if (http\_request.readyState == 4 ) {

var jsonObj = JSON.parse(http\_request.responseText);

document.getElementById("Name").innerHTML = jsonObj.name;

document.getElementById("Country").innerHTML = jsonObj.country;

}

}

http\_request.open("GET", data\_file, true);

http\_request.send();

}

</script> <title>Using JSON object</title> </head>

<body>

<h1>Cricketer Details</h1>

<table class = "src">

<tr><th>Name</th><th>Country</th></tr>

<tr><td><div id = "Name">Sachin</div></td>

<td><div id = "Country">India</div></td></tr>

</table>

<div class = "central">

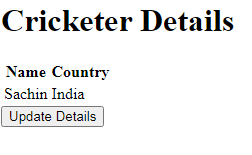
<button type = "button" onclick = "loadJSON()">Update Details </button>

</div>

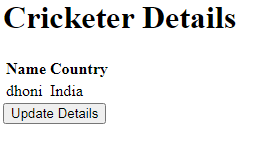
</body>

</html>

Output:



Click the update Details button



Dojo Toolkit

Dojo toolkit is another JavaScript framework that helps us to create web applications in a fast and easy way with a lot of functionality inside ready to use in any type of application

The Dojo Toolkit is a powerful open source JavaScript library that can be used to create rich and varied user interfaces running within a browser. The library requires no browser-side runtime plug-in and runs natively on all major browsers.

The Dojo Toolkit is a powerful and flexible modular Ajax software development kit. It is broken down in to three major layers: Dojo Core, Dijit, and DojoX.

* Dojo Core - All the major functions that are needed to do Ajax development, plus many features that are not found in other toolkits
* Dijit - A high quality set of interaction rich widgets and themes for use when developing Ajax applications.
* DojoX (Dojo eXtensions) - A module to contain widgets and APIs that are useful for developing Ajax applications, but are not needed in all applications.

**Installing Dojo**

Method 1:

To install Dojo, download the latest Dojo version from www.Dojotoolkit.org/downloads

to local hard drive.

Extract the files from the archive file to web development directory or web server. Including the dojo.js script file in your web application will give you access to all the Dojo functions.

To do this, place the following

script in the head element of your XHTML document:

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

where *path* is the relative or complete path to the Dojo toolkit’s files.

Method 2:

Include Dojo toolkit from a CDN(**Content Delivery Network**).

In this method no need to download the files, but provide the links to the files.

In this case the code will not work in offline mode.

The three most important modules to start programming are:

* ***dojo/ready***— To known when the DOM is ready and start with our app
* ***dojo/dom***— To use our DOM elements like with *dom.byId()* function
* ***dojo/dom-construct***— To create new DOM elements on the fly.

***Program***:

<!doctype html>

<html lang="en">

<head> <title>Title</title>

<!-- Required meta tags -->

<meta charset="utf-8">

<meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no">

<!-- Bootstrap CSS -->

<link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.3.1/css/bootstrap.min.css" integrity="sha384-ggOyR0iXCbMQv3Xipma34MD+dH/1fQ784/j6cY/iJTQUOhcWr7x9JvoRxT2MZw1T" crossorigin="anonymous">

<script src="https://ajax.googleapis.com/ajax/libs/dojo/1.7.12/dojo/dojo.js"></script>

<script>typeof(dojo) === "undefined" && document.write(unescape('%3Cscriptsrc="js/libs/dojo/dojo.js"%3E%3C/script%3E'))</script>

<script src="https://code.jquery.com/jquery-3.3.1.slim.min.js" integrity="sha384-q8i/X+965DzO0rT7abK41JStQIAqVgRVzpbzo5smXKp4YfRvH+8abtTE1Pi6jizo" crossorigin="anonymous"></script>

</head>

<body>

<h1>Learn Dojo Toolkit !</h1>

<div>Today is: <span id="day"></span></div>

<script>

require(["dojo/ready", "dojo/dom", "dojo/dom-construct","dojo/\_base/window"],function(ready,dom,construct,win){

// When the dom is ready

ready(function(){ // Uses dojo/ready

var day = dom.byId("day"); //Get <span id="day"> uses dojo/dom

//The week day

var d = new Date();

var weekDay = null;

switch(d.getDay()){

case 0:

weekDay = 'Sunday';

break;

case 1:

weekDay = 'Monday';

break;

case 2:

weekDay = 'Tuesday';

break;

case 3:

weekDay = 'Wednesday';

break;

case 4:

weekDay = 'Thursday';

break;

case 5:

weekDay = 'Friday';

break;

case 6:

weekDay = 'Saturday';

break;

}

day.innerHTML = weekDay;

var node = construct.create("h2",{

innerHTML: "Happy " + weekDay + " !"

},

win.body()

);

});

});

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

Output:

