AJAX for Java Web Applications

AJAX for Java Web Applications

APTECH LIMITED
Contact E-mail: ov-support@onlinevarsity.com

Edition 1 - 2015

Dear Learner,

We congratulate you on your decision to pursue an Aptech course.

Aptech Ltd. designs its courses using a sound instructional design model – from conceptualization to execution, incorporating the following key aspects:

- Scanning the user system and needs assessment

Needs assessment is carried out to find the educational and training needs of the learner

Technology trends are regularly scanned and tracked by core teams at Aptech Ltd. TAG* analyzes these on a monthly basis to understand the emerging technology training needs for the Industry.

An annual Industry Recruitment Profile Survey# is conducted during August - October to understand the technologies that Industries would be adapting in the next 2 to 3 years. An analysis of these trends & recruitment needs is then carried out to understand the skill requirements for different roles & career opportunities.

The skill requirements are then mapped with the learner profile (user system) to derive the Learning objectives for the different roles.

- Needs analysis and design of curriculum

The Learning objectives are then analyzed and translated into learning tasks. Each learning task or activity is analyzed in terms of knowledge, skills and attitudes that are required to perform that task. Teachers and domain experts do this jointly. These are then grouped in clusters to form the subjects to be covered by the curriculum.

In addition, the society, the teachers, and the industry expect certain knowledge and skills that are related to abilities such as *learning-to-learn, thinking, adaptability, problem solving, positive attitude etc*. These competencies would cover both cognitive and affective domains.

A precedence diagram for the subjects is drawn where the prerequisites for each subject are graphically illustrated. The number of levels in this diagram is determined by the duration of the course in terms of number of semesters etc. Using the precedence diagram and the time duration for each subject, the curriculum is organized.

- Design & development of instructional materials

The content outlines are developed by including additional topics that are required for the completion of the domain and for the logical development of the competencies identified. Evaluation strategy and scheme is developed for the subject. The topics are arranged/organized in a meaningful sequence.

The detailed instructional material – Training aids, Learner material, reference material, project guidelines, etc.- are then developed. Rigorous quality checks are conducted at every stage.

- Strategies for delivery of instruction

Careful consideration is given for the integral development of abilities like thinking, problem solving, learning-to-learn etc. by selecting appropriate instructional strategies (training methodology), instructional activities and instructional materials.

The area of IT is fast changing and nebulous. Hence considerable flexibility is provided in the instructional process by specially including creative activities with group interaction between the students and the trainer. The positive aspects of web based learning –acquiring information, organizing information and acting on the basis of insufficient information are some of the aspects, which are incorporated, in the instructional process.

- Assessment of learning

The learning is assessed through different modes – tests, assignments & projects. The assessment system is designed to evaluate the level of knowledge & skills as defined by the learning objectives.

- Evaluation of instructional process and instructional materials

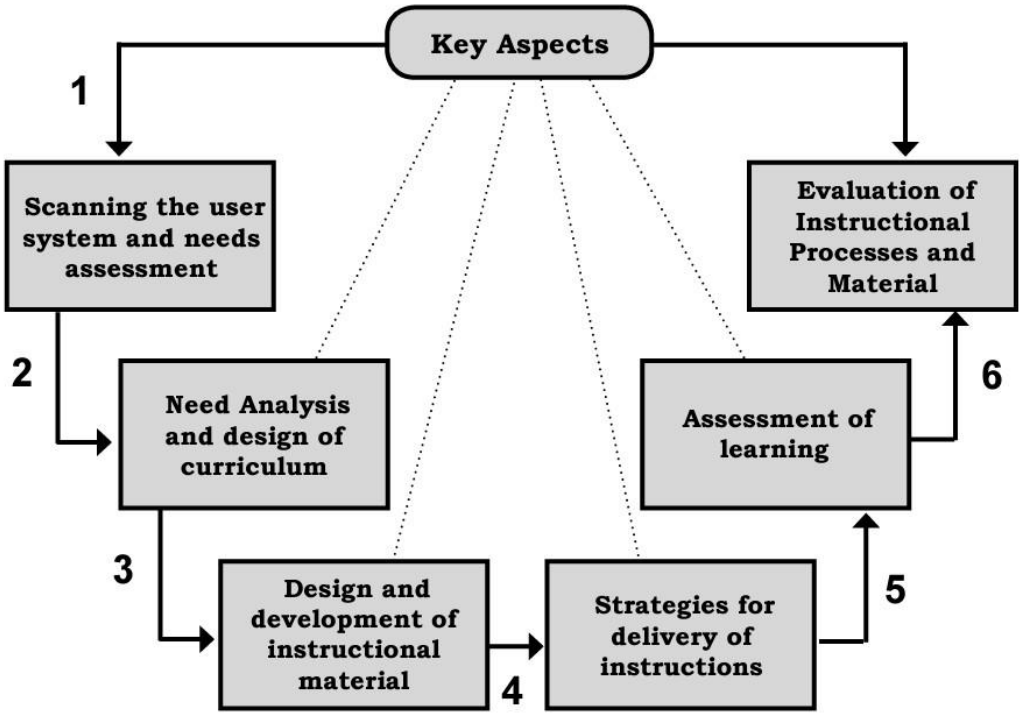
The instructional process is backed by an elaborate monitoring system to evaluate - on-time delivery, understanding of a subject module, ability of the instructor to impart learning. As an integral part of this process, we request you to kindly send us your feedback in the reply pre-paid form appended at the end of each module.

*TAG – Technology & Academics Group comprises of members from Aptech Ltd., professors from reputed Academic Institutions, Senior Managers from Industry, Technical gurus from Software Majors & representatives from regulatory organizations/forums.

Technology heads of Aptech Ltd. meet on a monthly basis to share and evaluate the technology trends. The group interfaces with the representatives of the TAG thrice a year to review and validate the technology and academic directions and endeavors of Aptech Ltd.

Industry Recruitment Profile Survey - The Industry Recruitment Profile Survey was conducted across 1581 companies in August/September 2000, representing the Software, Manufacturing, Process Industry, Insurance, Finance & Service Sectors.

Aptech New Products Design Model



Preface

Today, a new breed of Web applications have emerged having better, faster, and more user-friendly interfaces. These Web applications do not need page reloading unlike traditional Web applications for every request. These Web applications use AJAX extensively.

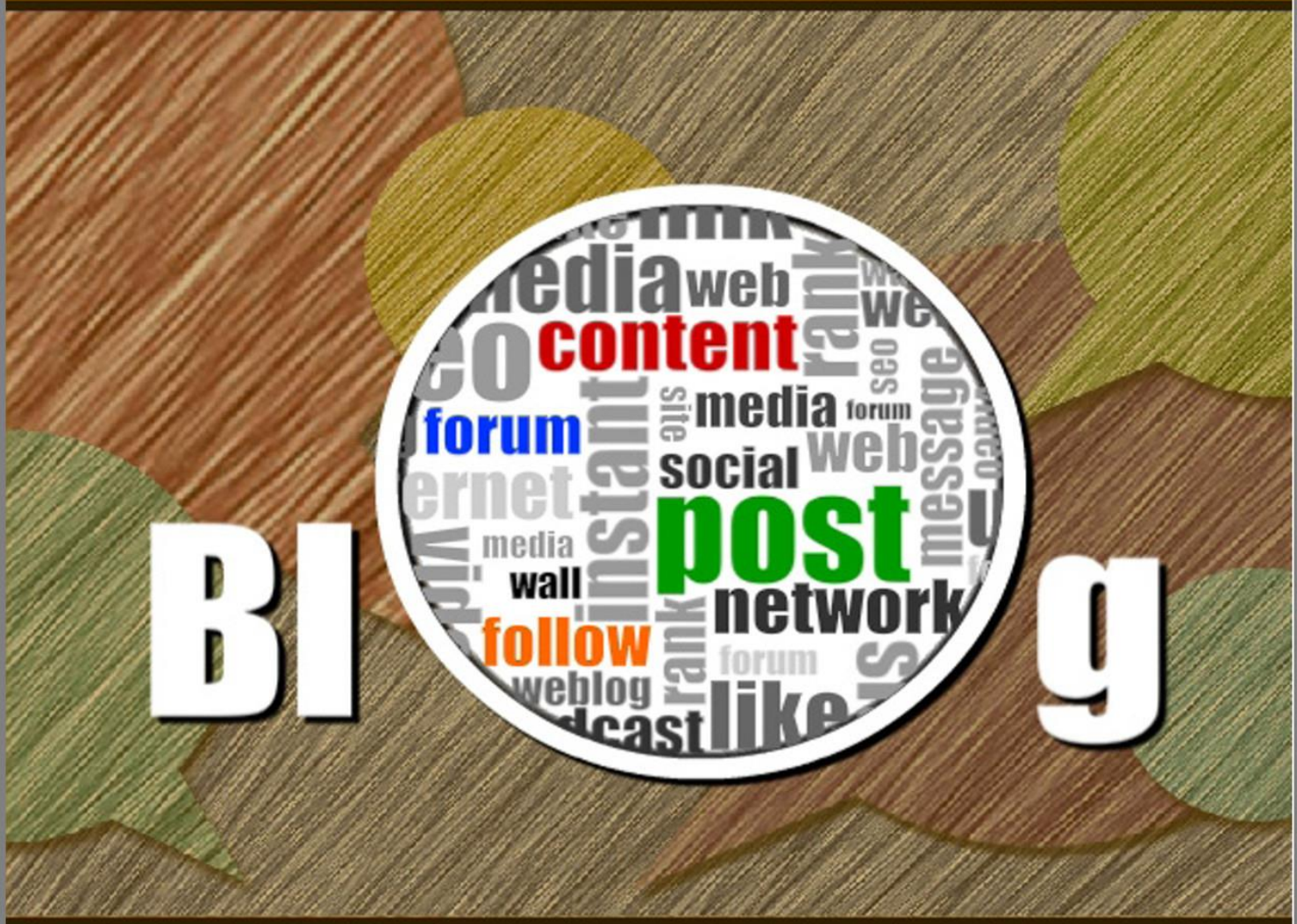
AJAX is not a new programming language, rather it is a new technique of performing the same old tasks. It is a combination of JavaScript and XML. Besides these two technologies, several other technologies such as Java, .NET, and a host of open source technologies are used to implement the AJAX effect in a traditional Web application.

This book begins with an overview of a typical 'AJAXified' Web application. It then familiarizes you with various AJAX features, technologies, toolkits, and frameworks used in these new breed of Web applications.

The knowledge and information in this book is the result of the concentrated effort of the Design Team, which is continuously striving to bring to you the latest, the best and the most relevant subject matter in Information Technology. As a part of Aptech's quality drive, this team does intensive research and curriculum enrichment to keep it in line with industry trends and learner requirements.

We will be glad to receive your suggestions. Please send us your feedback, addressed to the Design Centre at Aptech's corporate office.

Design Team



Balanced Learner-Oriented Guide

for enriched learning available @

Aptech COMPUTER EDUCATION Unleash your potential

Table of Contents

Sessions

01 Introduction to AJAX

Welcome to the Session, **Introduction to AJAX**.

This session explain about Asynchronous JavaScript and XML (AJAX) which refers to a group of Web technologies. AJAX provides Web applications with rich User Interface (UI), similar to desktop applications, and improves their response time and interactivity. Reduced request-response time of Web applications makes the Web applications highly responsive.

In this Session, you will learn to:

- Explain Asynchronous JavaScript and XML
- Describe the Document Object Model (DOM)
- Explain XMLHTTPRequest object methods and its properties

Session 01

Introduction to AJAX

1.1 Introduction

Asynchronous JavaScript and XML (AJAX) is the art of updating parts of a Web page without reloading the whole Web page and exchanging data with a server. AJAX allows Web pages to be updated asynchronously by exchanging small amounts of data behind the scenes with the server (asynchronously).

1.1.1 Conventional Web Applications

Conventional Web applications work in a simple manner. When a user clicks a Submit button, a link, and so on, an HTTP request is triggered by the browser at the client's end. This HTTP request is transmitted to the server.

At the server's end, server-side components process the HTTP request and send the results back to the client browser. On receiving the results from the server, the client browser displays the results after refreshing the Web page.

This mode of working is termed as 'click, wait, and refresh', as the user clicks, waits, and then views the results after the browser refreshes the Web page. The 'click, wait, and refresh' concept makes it easy to develop Web applications. However, this ease in development comes with a very great price, high request-response latency periods which in turn lead to slow responsiveness. Figure 1.1 shows the conventional Web application model.

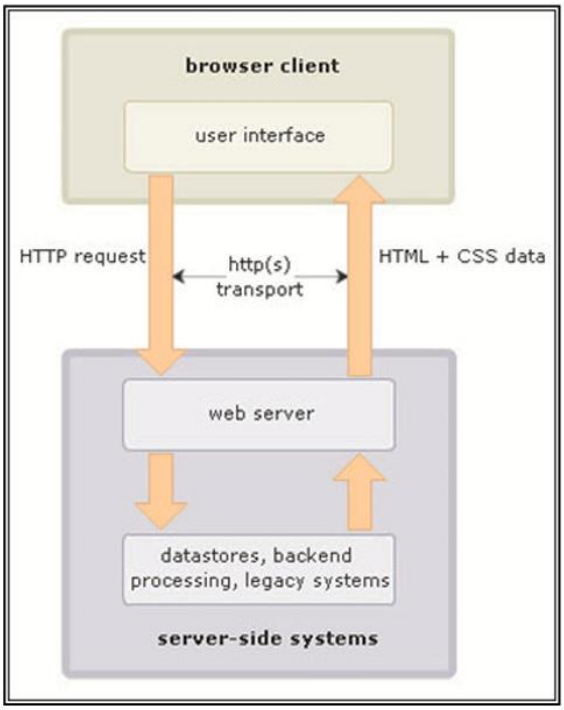


Figure 1.1: Conventional Web Application Model

Session 01

Introduction to AJAX

1.1.2 Drawbacks of Conventional Web Applications

Conventional Web applications communicate with the server synchronously. Once an HTTP request is generated, the user can no longer interact with the application. The user is forced to stay idle while the browser refreshes the Web page to display the results retrieved from the server. The page refresh also means that the existing browser contents to vanish, making the application completely unreadable. This synchronous mode of communication causes the long idle spells.

In addition to the long recurring waiting periods, conventional Web applications also lack rich user interface. This is because Web technologies such as servlets and JSP hardly provide any options to enhance the appearance of standard HTML components such as buttons, links, labels, and so on.

Thus, long waiting periods due to synchronous communication and poor user experience due to the lack of rich user interface are the major drawbacks of conventional Web applications. Figure 1.2 shows the synchronous model for conventional Web application.

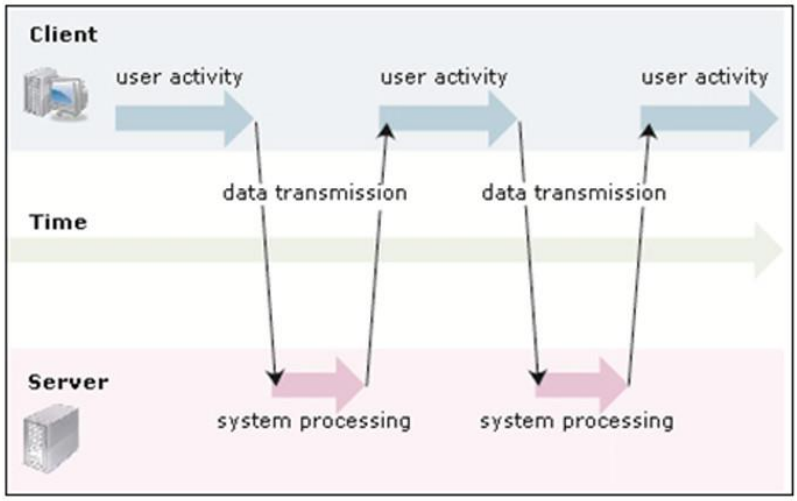


Figure 1.2: Synchronous Model for Conventional Web Application

1.1.3 AJAX vs. Conventional Web Page – A Real-time Example

The difference will be easily understood with a real-time example. For example, consider a situation in which it is required to create a Web page with a header, footer, and the menus that are common for all pages.

Conventional Web page

For the given requirement, if one designs in the conventional way, then for each links that is placed in the page, a request is sent to the server and the server responds with a page. The response contains the contents retrieved from the server based on the query for the page.

Along with that, the menus, header, and footer will also be sent again from the server.

Similarly, for each subsequent request, the menu, header, and footer are loaded again from the server.

Using AJAX

With AJAX, the page can be designed as follows:

- Create the server code which returns only the content for the requests it receives.
- The initial requests loads everything by default, but when the links are clicked, only the requested contents will be loaded through AJAX, whereas the other contents will remain static on the Web page.

Note - In such situations, using conventional way, one can use frames to avoid the problem of reloading the redundant contents. However, frames are not advisable in this case.

1.1.4 Birth of AJAX

The need to overcome the conventional Web application's drawbacks forced Web developers to look for alternative Web development methods. Developers overcame the drawbacks using technologies such as JavaScript, XML, DOM, CSS, and so on.

Web developers all over the world adapted to this trend of using disparate technologies. However, Jesse James Garrett was the first to talk about them. According to his article, 'Ajax: A New Approach to Web Applications'. Garrett listed the various technologies involved, discussed their roles, and explained how they worked together. Garrett named this group of technologies as Asynchronous JavaScript and XML (AJAX). Garrett's article marked the birth of AJAX.

AJAX is a group of technologies. HTML and CSS can be used to markup and style information and the DOM is accessed with JavaScript. For exchanging data asynchronously, JavaScript and the XMLHttpRequest object is used which avoid reloading of full page.

Figure 1.3 shows the technologies used in AJAX Web application.

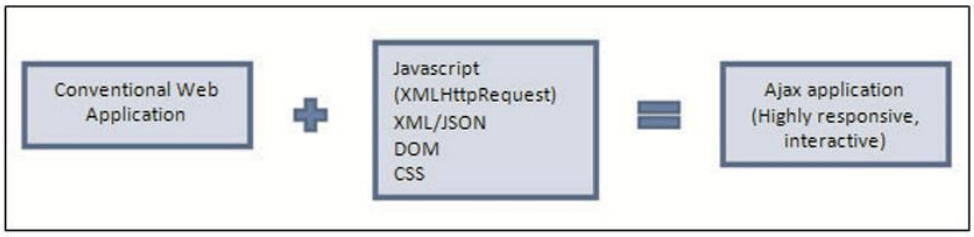


Figure 1.3: Technologies Used in AJAX Web Application

The technologies such as XML/JSON, DOM and CSS can be part of AJAX application. XMLHTTPRequest API is the core of AJAX. There cannot be an AJAX application without XMLHTTPRequest.

Session 01

Introduction to AJAX

1.1.5 AJAX Technologies

AJAX comprises JavaScript, XML, DOM, and CSS. These technologies enable development of highly responsive and rich UI-based Web applications.

- **JavaScript**

  JavaScript facilitates the creation of XMLHttpRequest objects. XMLHttpRequest objects facilitate asynchronous communication between the client and the server. Asynchronous communication permits the user to continue interacting with the Web page on the client-side, while the XMLHttpRequest object retrieves data from the server. Thus, the user does not experience a page refresh.

- **XML/JSON**

  The server-side components process the client request and send a corresponding response back to the client. The response contains the data requested by the client. This data is sent to the client in XML/JSON format. With wide range of support and other advantages, JSON is mostly used in client-server communication, instead of XML.

- **DOM**

  DOM performs dual role in AJAX. Firstly, DOM allows parsing the XML response received from the server and extract the data from it. Secondly, it allows accessing the Web page's DOM tree to add or update existing nodes with new data received from the server. In other words, DOM facilitates the update of a Web page.

- **CSS**

  CSS enables adding desktop application like look-and-feel to Web applications.

Note - Simply combining these technologies does not mean AJAX. AJAX mainly means having asynchronous mechanism offered by XMLHttpRequest and the mentioned technologies have support to be used in AJAX.

1.1.6 Working of AJAX

A group of interrelated Web development methods, used to create interactive Web applications on the client-side is AJAX. Web applications can retrieve data from the server asynchronously with AJAX in the background without interfering with the behavior and display of the existing page.

Consider the Web page of an AJAX-enabled Web application. A user needs to provide a credit card number. The application validates the credit card number asynchronously using the Validate CCN servlet and a Web service on the server. Based on the result of validation, a corresponding message is displayed next to the text box.

Figure 1.4 shows the various steps involved in processing the AJAX request and getting the response from the server.

Session 01

Introduction to AJAX

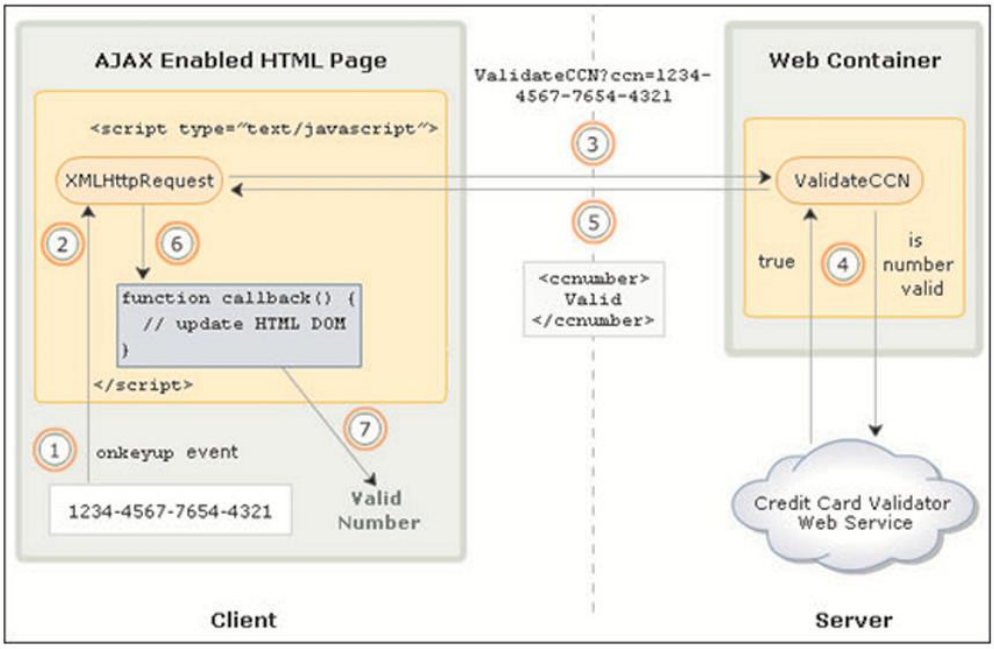


Figure 1.4: AJAX Request and Response Model

Step 1

The input text box is associated with an event handler with the help of the onkeyup event. Thus, every time the user types a digit in the text box, the onkeyup event occurs. Every time the onkeyup event occurs, an event handler is invoked.

Step 2

The event handler is a JavaScript function that creates an XMLHttpRequest object and configures it using the open() method. The open() method specifies the HTTP method (GET or POST), the URL of the server-side component that processes the request, and the mode (synchronous or asynchronous) of communication. Note that the credit card number is included as a request parameter in the HTTP request.

Step 3

The send() method of the XMLHttpRequest object is then invoked. This establishes a connection with the server-side component such as a servlet or a JSP page. In this case, this component is a servlet. Therefore, the servlet whose URI is mapped to ValidateCCN is executed.

Step 4

The processing of AJAX begins. The servlet first retrieves the credit card number from the request object. This number is then sent to a credit card validator Web service. This service verifies the credit card number and accordingly intimates the servlet.

Step 5

The servlet then generates an XML response. This XML response is an XML document containing the element <ccnumber> with the text Valid or Invalid in it. If the credit card number is valid, the text Valid is enclosed in the <ccnumber> element. This XML response is sent to the client.

Step 6

After the client receives the XML response, a callback function is called. The callback function is usually specified in Step 2 while configuring the XMLHttpRequest object. The XML response sent by the server is accessible through the responseXML property of the XMLHttpRequest object.

Step 7

The callback function displays a message on the Web page about the validity of credit card number. This is achieved by reserving a div element specifically for displaying such a message. The callback function retrieves this div element using DOM API and sets its innerHTML property to display the message.

No HTTP(S) request is initiated until the send() method is called by XMLHttpRequest implementations. However, the XMLHttpRequest API is designed in a way as if each method was writing to a network stream. This means that the XMLHttpRequest method must be called which matches the structure of an HTTP request. For example, setRequestHeader() must be called after open() is called and before the call of send() or it will throw an exception.

The request is asynchronous as the user does not need to wait for the response to proceed ahead or perform other activity on the Web page.

1.2 Document Object Model

The Document Object Model is a platform and language-independent standard for representing HTML and XML documents. DOM is a standard object model. DOM allows accessing and manipulating the HTML and XML document content. Thus, DOM facilitates dynamic modification of Web pages.

DOM represents HTML or XML document as a collection of objects referred as nodes. Based on how these objects are placed in the document, DOM connects each of these nodes creating a tree like structure. DOM classifies every node as a specific type of node. For example, tags are classified as element nodes, whereas text within the tags is classified as text nodes.

Consider the Example.html file demonstrated in Code Snippet 1.

Code Snippet 1:

```
<!DOCTYPE html>
<html>
      <head>
           <title>
```

Session 01

Introduction to AJAX

```
                DOM Example
            </title>
      </head>
      <body>
            <p id="id1"> DOM example paragraph</p>
            <a href="http://google.com"> Google.com </a>
      </body>
</html>
```

The html tag is the main tag. It has two sub tags, title and body. The title tag contains text, while body tag consists sub tags, p and a. Again, each of these tags contains text.

Using these observations, a tree structure can be created as shown in figure 1.5.

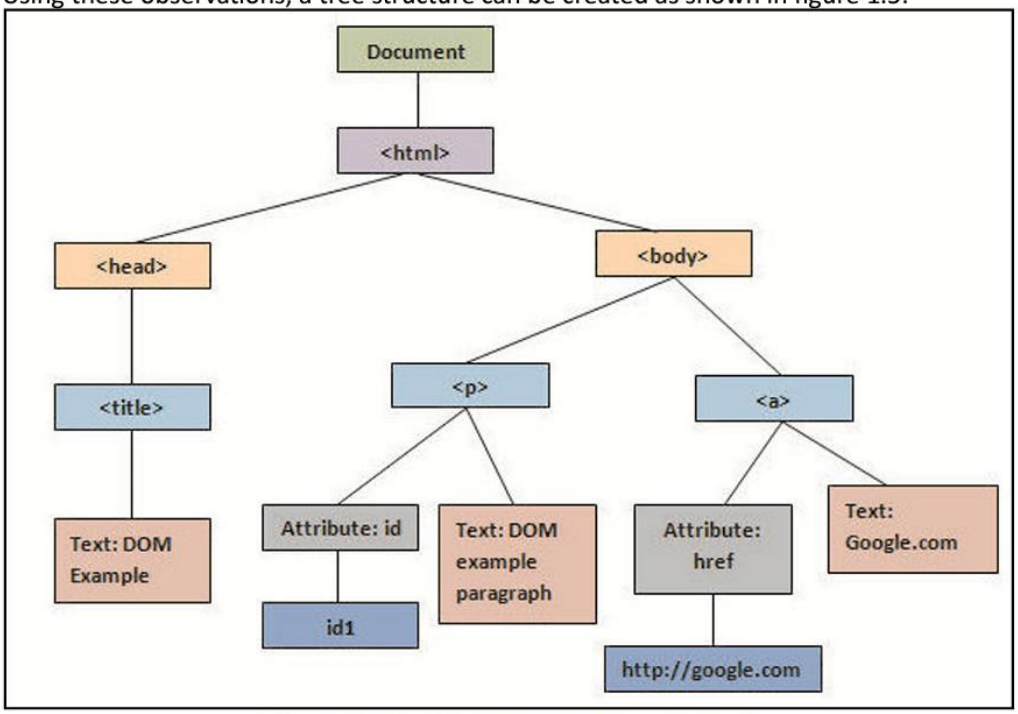


Figure 1.5: Tree Structure of Example.html

1.2.1 HTML DOM Methods

HTML Document Object Model (DOM) allows accessing and manipulating the HTML documents using built-in methods. These HTML DOM methods are as follows:

- **getElementById(id)**

  The getElementById() method searches for and returns the element whose id is specified as the input parameter.

Session 01

Introduction to AJAX

Syntax:

```
getElementById(id)
```

where,

id - value of the id attribute of an element in an HTML document

Code Snippet 2 demonstrates the use of the getElementById() method.

Code Snippet 2:

```
<!DOCTYPE html>
<html>
<head>
<script type="text/javascript">
function updateDiv(theDiv,theText){
      document.getElementById(theDiv).innerHTML=theText;
}
</script>
</head>
<body>

<h1>Accessing by id</h1>
<input type="button" value="Update div" onclick="updateDiv
('demo','Content updated by accessing id!!!');">

<div id="demo">Default content.</div>
</body>
</html>
```

The code snippet consists of a button and a div with default content, as shown in figure 1.6.

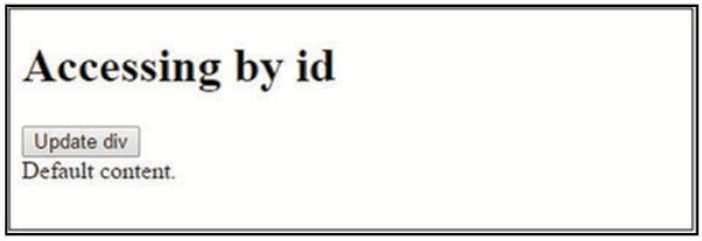


Figure 1.6: HTML Page with Button and Div

When the button is clicked, the default content of the div will be updated. The div will be accessed using the id attribute. Figure 1.7 shows the output when the button is clicked.

Session 01

Introduction to AJAX

Figure 1.7: Updated Content of Div After Button Click

- **getElementsByTagName(name)**

The getElementsByTagName() method searches for and returns all such elements whose name matches to the one specified as the input parameter. It returns the entire list of elements in the form of an array.

Syntax:

```
getElementsByTagName(name)
```

where,

name – name of a tag in an HTML document

Code Snippet 3 demonstrates the use of the getElementsByTagName() method.

Code Snippet 3:

```
<!DOCTYPE html>
<html>
      <head>
            <title>
                  Accessing by tag name!!!
            </title>
            <script type="text/javascript">
            function updateByTagName(theTagName){
            x = document.getElementsByTagName(theTagName);
            for (i=0;i < x.length; i++){
                  x[i].innerHTML+="  -  Modified  content  by
accessing the tag name.";
                  }
            }
            </script>
      </head>
      <body>
     <h1>Accessing by tag name!!!</h1>
      <input type="button" value="Update By Tag Name" onclick
="updateByTagName('p');"/>
```

```
            <p>   Hello World 1!   </p>
            <p>   Hello World 2!   </p>
            <p>   Hello World 3!   </p>
            <p>   Hello World 4!   </p>
      </body>
</html>
```

The code snippet consists of the script that modifies the node value of <p> tag which is accessed using the tag name. The script will be triggered on clicking the button, 'Update By Tag Name'.

Figure 1.8 shows the output of the code snippet.

Figure 1.8: Output of Code Snippet 3

On clicking the button, the code results in the following output, as shown in figure 1.9.

Figure 1.9: Output - Updated by Accessing the Tag Value

- **appendChild(node)**

The appendChild() method appends the node specified as input parameter to the tree structure.

Syntax:

```
appendChild(node)
```

where,

node - the exact node that needs to be appended

Code Snippet 4 demonstrates the use of the appendChild() method.

Code Snippet 4:

```
<!DOCTYPE html>
<html>
      <head>
            <title>
                  Append Child Demo!!!
            </title>
            <script type="text/javascript">
            function appendChildTest(theTagName){
      var x = document.getElementsByTagName(theTagName);
      var aNewNode = document.createElement("p");
      var  aTextNode  =  document.createTextNode("Hello  World
Child!!!");
     aNewNode.appendChild(aTextNode);
                  x[0].appendChild(aNewNode);
            }
            </script>
      </head>
      <body>
      <h1>Append Child Demo!!!</h1>
      <input type="button" value="Append Child" onclick="appe
ndChildTest('p');"/>
            <p> Hello World 1! </p>
            <p> Hello World 2!</p>
            <p> Hello World 3! </p>
            <p> Hello World 4! </p>
      </body>
</html>
```

The code provides the user with a button. The script creates a new node of type specified in the parameter of the appendChildTest function. Next, it appends the new node to the <p> node at index 0, that is, the first <p> node. Figure 1.10 shows the output of the code snippet.

Figure 1.10: Output of Code Snippet 4

When the button is clicked, the code results in the following output, as shown in figure 1.11.

Figure 1.11: Child Added to the Node on Button Click

- **removeChild(node)**

The removeChild() method removes the node specified as input parameter from the tree structure.

Syntax:

```
removeChild(node)
```

where,

node – the exact node that needs to be removed

Code Snippet 5 demonstrates the use of the removeChild() method.

Code Snippet 5:

```
<!DOCTYPE html>
<html>
      <head>
            <title>
                  Remove Child Demo!!!
            </title>
            <script type="text/javascript">
            function removeChildTest(theTagName,thenthChild){
            var x = document.getElementsByTagName(theTagName);
            x[0].parentNode.removeChild(x[thenthChild]);
            }
            </script>
      </head>
      <body>
      <h1>Remove Child Demo!!!</h1>
      <input type="button" value="Remove Child" onclick="remov
eChildTest('p',2);"/>
         <p> Hello World 1! </p>
          <p> Hello World 2!</p>
```

```
            <p> Hello World 3! </p>
            <p> Hello World 4! </p>
      </body>
</html>
```

The code consists of a button, which when clicked, removes the node specified in the second parameter of the removeChildTest method.

Figure 1.12 shows the output of the code snippet.

Figure 1.12: Output of Code Snippet 5

When the button is clicked, the code results in the output as shown in figure 1.13.

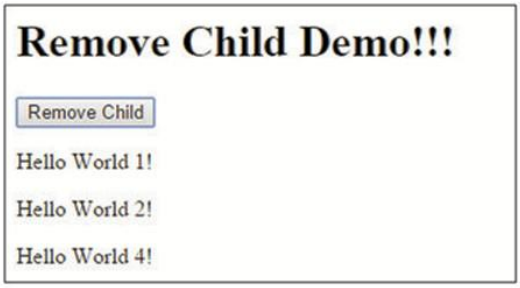


Figure 1.13: Node at Index 2 Removed on Button Click

Note that node at index 2, that is, Hello World 3! has been removed.

1.2.2 HTML DOM Properties

HTML DOM properties provide information about the various nodes in a DOM tree. You can use these properties to access the information stored within the nodes as well. The HTML DOM object properties are as follows:

- **innerHTML**

  The innerHTML property provides access to the content of an element. Therefore, this property can be used to retrieve or update the text in an element. Code Snippet 6 demonstrates the use of the innerHTML property.

Code Snippet 6:

```
<!DOCTYPE html>
<html>
      <head>
            <title>
                  innerHTML Demo
            </title>
            <script type="text/javascript">
            function updateInnerHTML(theTagName,thenthChild){
      var x = document.getElementsByTagName(theTagName);
      x[thenthChild].innerHTML="Updated content!!!";
            }
            </script>
      </head>
      <body>
      <h1>innerHTML Demo!!!</h1>
      <input type="button" value="Update innerHTML" onclick="
updateInnerHTML('p',2);"/>
            <p> Hello World 1! </p>
            <p> Hello World 2!</p>
            <p> Hello World 3! </p>
            <p> Hello World 4! </p>
  </body>
</html>
```

The code provides the user with a button which when clicked, triggers a function that access the innerHTML of the DOM element specified in the second parameter of the updateInnerHTML function and updates that. Figure 1.14 shows the output of the code.

Figure 1.14: Output of Code Snippet 6

When the button is clicked, the output is modified as shown in figure 1.15.

Figure 1.15: Updated Content of the Node at Position 2

- **nodeName**

The nodeName property provides access to the name of the current node. It is used to display the name of the current node. Code Snippet 7 demonstrates the use of the nodeName property.

Code Snippet 7:

```
<!DOCTYPE html>
<html>
      <head>
            <title>
                  nodeName Demo!!!
            </title>
            <script type="text/javascript">
            function displayNodeName(tagID){
                  var x = document.getElementById(tagID);
                  alert("Node name: '"+x.nodeName+"'");
            }
            </script>
      </head>
      <body>
      <h1>nodeName Demo!!!</h1>
      <input type="button" value="Alert Node Name" onclick="
displayNodeName('hello');"/>
            <div id="hello"> Hello World 1! </div>
      </body>
</html>
```

The code accesses the value of a node using the nodeName property. The id of the node is passed as parameter to the displayNodeName function. The value of the node is retrieved using the nodeName property and displayed using an alert. Figure 1.16 shows the output of the code.

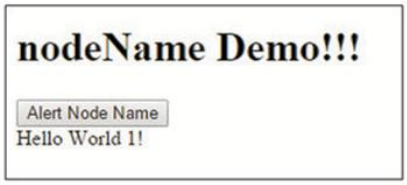


Figure 1.16: Output of Code Snippet 7

When the button is clicked, the function is called and the alert displays the node's name using the nodeName property, as shown in figure 1.17.

Figure 1.17: Name of the Node Displayed in the Alert

- **childNodes**

The childNodes property provides access to all the child nodes of the given node. One can access the child nodes by using prefixes. For example, childNodes[1] will return the second child node of the current node.

- **nodeValue**

The nodeValue property can be used to display a node's content. The nodeValue property when used with text nodes, returns the nodes content. The nodeValue property when used with element nodes returns the value 'null'. To get the actual text content in the element, use '.childNodes[0].nodeValue' for the selected node. Code Snippet 8 demonstrates the use of the childNodes and nodeValue properties.

Code Snippet 8:

```
<!DOCTYPE html>
<html>
      <head>
            <title> nodeValue Demo!!! </title>
            <script type="text/javascript">
            function displayNodeValue(tagId) {
      var x = document.getElementById(tagId);
      alert("Node Value: '"+x.childNodes[0].nodeValue+"'");
            }
```

Session 01

Introduction to AJAX

```
    </script>
 </head>
   <body>
     <h1>nodeValue Demo!!!</h1>
     <input type="button" value="Alert Node value" onclick="
displayNodeValue('div1');"/>
          <div id="div1">
               Hello World 1!
          </div>
    </body>
</html>
```

The text value will be displayed using the childNodes property of the div that is used in the code. To get the value of that node, '.childNodes[0].nodeValue' has been used on the div.

Figure 1.18 shows the output of the code.

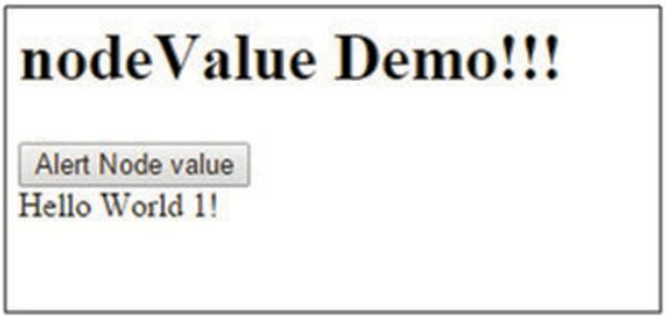


Figure 1.18: Output of Code Snippet 8

On clicking the button, the content of the div will be displayed in the alert, as shown in figure 1.19.

Figure 1.19: Output - Alert Showing the Value in the Div

- **parentNode**

The parentNode property provides access to the parent node of the current node. Code Snippet 9 shows the use of parentNode property.

Code Snippet 9:

```
<!DOCTYPE html>
<html>
      <head>
            <title>
                  parentNode Demo!!!
            </title>
            <script type="text/javascript">
            function displayParentNode(tagId){
                  var x = document.getElementById(tagId);
                  alert("Parent  Node  name:  '"+x.parentNode.
nodeName+"'");
            }
            </script>
      </head>
      <body>
      <h1>parentNode Demo!!!</h1>
            <input type="button" value="Alert Parent Node" on
click="displayParentNode('div1');"/>
            <div id="div1">
                  Hello World 1!
            </div>
      </body>
</html>
```

The code retrieves the tag name using the id and then displays the name of its parent node using parentNode.nodeName. Figure 1.20 shows the output of the code.

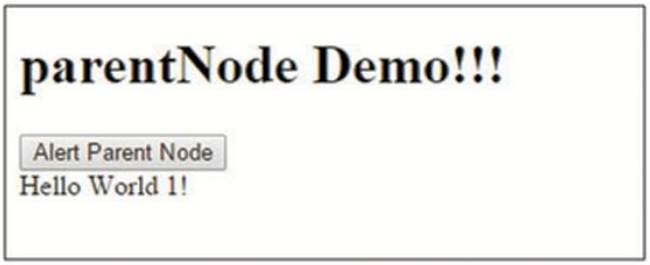


Figure 1.20: Output of Code Snippet 9

On clicking the button, the alert is displayed with the name of the parent node of div tag as shown in figure 1.21.

Figure 1.21: Parent Node Name Displayed in the Alert

- **length**

The length property returns the number of nodes present in a node list. The getElementsByTagName() method returns a list of all the nodes having a specific tag name. The length property returns the total number of nodes present in the list.

Code Snippet 10 shows the use of the length property.

Code Snippet 10:

```
<!DOCTYPE html>
<html>
      <head>
            <title>
                  length Demo!!!
            </title>
            <script type="text/javascript">
            function displayBodyLength(theTagName){
                  var x = document.getElementsByTagName(theTa
gName);
                  alert("Number of div nodes: '"+x.length+"'");
            }
            </script>
      </head>
      <body>
     <h1>length Demo!!!</h1>
            <input type="button" value="Alert div count" oncl
ick="displayBodyLength('div');"/>
       <div id="div1">      Hello World 1!   </div>
      <div id="div2"> Hello World 2!   </div>
      </body>
</html>
```

The code gives the length of the selected nodeList. That is, number of nodes that have been selected using the getElementsByTagName method.

Figure 1.22 shows the output of the code.

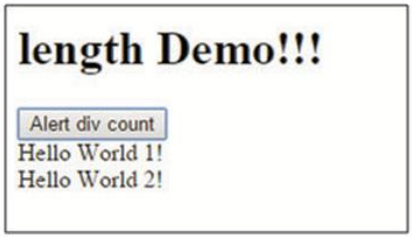


Figure 1.22: Output of Code Snippet 10

On clicking the button, the alert is displayed with the number of div nodes in the selected node list as shown in figure 1.23.

Figure 1.23: Number of Div Nodes Displayed in the Alert

1.2.3 XML DOM Methods

XML Document Object Model (XML DOM) allows accessing and manipulating XML documents using methods similar to that of HTML DOM.

For better understanding, create an XML document named, Student.xml. With this XML, use the Example.html for executing the code snippet of XML DOM methods and properties. Deploy the two files in an application server. The Exampe.html will be accessed through the application server. Code Snippet 11 demonstrates the Student.xml document.

Code Snippet 11:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<students>
     <record id="1">
          <name>Ashley Andrews</name>
          <gender>male</gender>
          <age>12 yrs</age>
          <grade>7th</grade>
          <division>A</division>
     </record>
```

Session 01

Introduction to AJAX

```
      <record id="2">
            <name>Ashley Mathias</name>
            <gender>male</gender>
            <age>5 yrs</age>
            <grade>1st</grade>
            <division>B</division>
      </record>
      <record id="3">
            <name>Ashley Bill</name>
            <gender>male</gender>
            <age>6 yrs</age>
            <grade>1st</grade>
            <division>A</division>
     </record>
</students>
```

Before working with XML DOM methods, the script that loads the XML document as a JavaScript instance must be added to the HTML page as demonstrated in Code Snippet 12.

Code Snippet 12:

```
<script type="text/javascript">
      function loadXMLDoc(theDocName){
            if (window.XMLHttpRequest){
              xhttp=new XMLHttpRequest();
            }
            else{
              xhttp=new ActiveXObject("Microsoft.XMLHTTP");
            }
            xhttp.open("GET",theDocName,false);
            xhttp.send();
            return xhttp.responseXML;
      }
</script>
```

The XML DOM methods are as follows:

- **getElementsByTagName(name)**

  The getElementsByTagName() method searches for and returns elements whose name matches with the input parameter.

  Syntax:

  ```
  getElementsByTagName(name)
  ```

  where,

  name – name of a tag in an XML document

Code Snippet 13 demonstrates the use of the getElementsByTagName() method.

Code Snippet 13:

```
<html>
 <head>
 <title>
     XML DOM methods and properties.
 </title>
     <script type="text/javascript">
          function loadXMLDoc(theDocName) {
               if (window.XMLHttpRequest){
                 xhttp=new XMLHttpRequest();
               }
               else{
          xhttp=new ActiveXObject("Microsoft.XMLHTTP");
               }
               xhttp.open("GET",theDocName,false);
               xhttp.send();
               return xhttp.responseXML;
          }

function printNames(theXML,theNodeName,theIdToUpdate)  {
          xmlDoc = loadXMLDoc(theXML);
          x = xmlDoc.getElementsByTagName(theNodeName);
          var OPStr="";
          for (i = 0;i < x.length;i++){
               OPStr+=x[i].childNodes[0].nodeValue+"<br/>";
          }
     document.getElementById(theIdToUpdate).innerHTML=OPStr;
          }
     </script>
 </head>
 <body>
    <input    type="button"    onclick='printNames("Student.
xml","name","divOP");' value="Parse"/>
     <div id="divOP">
     </div>
 </body>
</html>
```

The function printNames accepts the xml file to be loaded, the name of the node to be read and the id to be updated. Further, the node name is read and the values within the node are displayed.

In accordance with the XML content, the script generates the result when the Parse button is clicked, as shown in figure 1.24.

Session 01

Introduction to AJAX

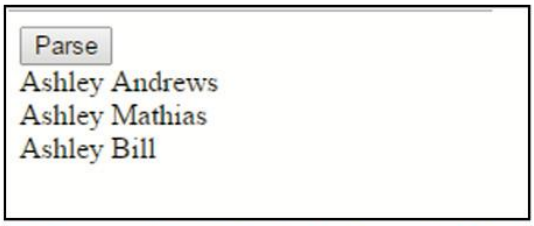


Figure 1.24: Output - The Values in Tag 'name' of the XML File

- **removeChild(node)**

The removeChild() method removes the node specified as input parameter from the tree structure.

Syntax:

```
removeChild(node)
```

where,

node – the exact node that needs to be removed

Code Snippet 14 demonstrates the use of the removeChild() method.

Code Snippet 14:

```
<html>
 <head>
 <title>
     XML DOM methods and properties.
 </title>
     <script type="text/javascript">
          function loadXMLDoc(theDocName){
               if (window.XMLHttpRequest){
                 xhttp=new XMLHttpRequest();
               }
               else{
          xhttp=new ActiveXObject("Microsoft.XMLHTTP");
               }
               xhttp.open("GET",theDocName,false);
               xhttp.send();
               return xhttp.responseXML;
          }

     function removeNode(theXML,theNodeName,theIdToRemove){
          var xmlDoc = loadXMLDoc(theXML);
          var x = xmlDoc.getElementsByTagName(theNodeName);
          var temp = x[theIdToRemove];
          x[0].parentNode.removeChild(temp);
          console.log(xmlDoc);
```

Session 01

Introduction to AJAX

```
    }
    </script>
</head>
<body>
```

In accordance with the XML content, when the button is clicked, the script removes a node from the XML document and logs the XML to the console using console.log.

To view the console in Google chrome, perform the following steps:

1. Run the HTML page in Google Chrome.
2. Click the Settings  icon in the upper right corner of Chrome.
3. Select More tools → JavaScript console.
4. Expand the tags to view the result.

Figure 1.25 shows the console output in the chrome for the code snippet. This clearly indicates the node has been removed.

Figure 1.25: XML Node at Index 1 is Removed

- **appendChild(node)**

The appendChild() method appends the node specified as input parameter to the tree structure.

Session 01

Introduction to AJAX

Syntax:

```
appendChild(node)
```

where,

node – the exact node that needs to be appended

Code Snippet 15 shows the use of the appendChild() method.

Code Snippet 15:

```
<html>
 <head>
 <title>
      XML DOM methods and properties.
 </title>
      <script type="text/javascript">
            function loadXMLDoc(theDocName){
                  if (window.XMLHttpRequest){
                    xhttp=new XMLHttpRequest();
                  }
                  else{
            xhttp=new ActiveXObject("Microsoft.XMLHTTP");
                  }
                  xhttp.open("GET",theDocName,false);
                  xhttp.send();
                  return xhttp.responseXML;
            }

            function removeAndAppendNode(theXML,theNodeName,t
heIdToRemove){
            var xmlDoc = loadXMLDoc(theXML);
            var x = xmlDoc.getElementsByTagName(theNodeName);
            var temp = x[theIdToRemove];
            x[0].parentNode.removeChild(temp);
            x[0].parentNode.appendChild(temp);
            console.log(xmlDoc);
          }
      </script>
 </head>
 <body>
      <input type="button" onclick='removeAndAppendNode("Stud
ent.xml","record",1);' value="Re-append a node"/>
 </body>
</html>
```

The function, removeAndAppendNode, accepts the name of the xml file, the name of the node, and the id of the node to be replaced.

Further, the id is stored in a temporary variable and the node is removed from the parent node and appended to the end of the parent node.

On clicking the button, the script removes the node at index 1 (id - 2) from its current location in the XML document and appends it at the end, as shown in figure 1.26.

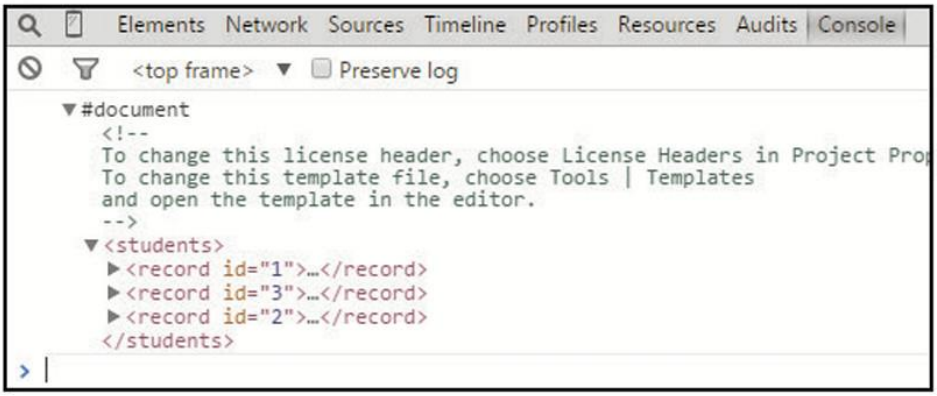


Figure 1.26: Record with 'id' 2 is Appended at the End

Note - Unlike HTML DOM, XML DOM does not support the getElementById() method.

1.2.4 XML DOM Properties

XML DOM nodes possess a string of properties. These properties enable extraction of data from the XML document. The XML DOM object properties are as follows:

- **nodeName**

  The nodeName property provides access to the name of the current node. It is widely used to display the name of the current node.

- **nodeValue**

  The nodeValue property can be used to display a node's value. Not all nodes have values. For example, an element node could either have a text node or another element node as its child node. In either case, the element node will always have a value of null.

- **parentNode**

  The parentNode property provides access to the parent node of the current node. One can use the parentNode property to access or gain control of the current node's parent.

- **childNodes**

  The childNodes property provides access to all the child nodes of the given element. It returns a list of all the child nodes present. One can access the various child nodes using prefixes. For example, the second child node is accessed using expression `childNodes[1]`.

Session 01

Introduction to AJAX

- **length**

The length property returns the number of nodes present in a node list. The getElementsByTagName() method returns a list of all the nodes having a specific tag name. The length property returns the total number of nodes present in the list.

- **attributes**

The attributes property returns a list containing the specified node's attributes. If the specified node is not an element, this property returns NULL.

Code Snippet 16 demonstrates the use of the different DOM properties.

Code Snippet 16:

```
<html>
<head>
<title>
     XML DOM methods and properties.
</title>
     <script type="text/javascript">
          function loadXMLDoc(theDocName){
               if (window.XMLHttpRequest){
                 xhttp=new XMLHttpRequest();
               }
               else{
               }
               xhttp.open("GET",theDocName,false);
               xhttp.send();
               return xhttp.responseXML;
          }

          function updateTable(theXML,theNodeName){
               var xmlDoc = loadXMLDoc(theXML);
               var x = xmlDoc.getElementsByTagName(theNode
Name);

// Update the node name. This is same as the name that is used
to select.
// That is, the value in the variable theNodeName
document.getElementById("nodeNameId").innerHTML=x[0].
nodeName;

// childNodes[1] selects the first node of x[0]. (x[0].
childNodes[0] is //reserved to contain the text in the x[0])
// From the childNodes the node name is selected.
document.getElementById("childNodesid").innerHTML=x[0].
childNodes[1].nodeName;
```

Session 01

Introduction to AJAX

```
// x[0] selects the first node with the selected node name
// childNodes[1] selects the first node of x[0]. (x[0].
childNodes[0] is
//reserved to contain the text in the x[0])
// Then, from the first XML node of x[0] the nodeValue is selected
document.getElementById("nodeValueId").innerHTML=x[0].
childNodes[1].childNodes[0].nodeValue;
// Getting the parentNode through which Parent's name is
retrieved

document.getElementById("parentNodeId").innerHTML=x[0].
parentNode.nodeName;

// Getting the length of the selected nodes
document.getElementById("lengthid").innerHTML=x.length;

// attributes property returns the attributes and one can fetch
the value //for the attribute 'id' from that.

document.getElementById("attributesid").innerHTML=x[0].
attributes.getNamedItem("id").value;
          }
</script>
 </head>
 <body>
        <input  type="button"  onclick='updateTable("Student.
xml","record");' value="Update table from XML"/>
     <table border="1">
     <col width="80%">
     <col width="20">
          <tbody>
<tr>
  <td> DOM Property </td>
<td> Output </tx>
<tr>
                <tr>
                     <td>
xmlDocObject.getElementsByTagName(theNodeName)[0].nodeName
                     </td>
                     <td>
                          <span id="nodeNameId">
                          </span>
                     </td>
```

Session 01

Introduction to AJAX

```
                </tr>
                <tr>
                      <td>
xmlDocObject.getElementsByTagName(theNodeName)[0].nodeValue
                      </td>
                      <td>
                            <span id=”nodeValueId”>
                            </span>
                      </td>
                </tr>
                <tr>
                      <td>
xmlDocObject.getElementsByTagName(theNodeName)[0].
parentNode.nodeName
                      </td>
                      <td>
          <span id=”parentNodeId”></span>
                      </td>
                </tr>
                <tr>
                      <td>
     xmlDocObject.getElementsByTagName(theNodeName)[0].
childNodes[1].nodeName
                      </td>
                      <td>
                            <span id=”childNodesid”>
                            </span>
                      </td>
                </tr>
                <tr>
                      <td>
                            xmlDocObject.getElementsByTagName
(theNodeName).length
                      </td>
                      <td>
                            <span id=”lengthid”>
                            </span>
                      </td>
                </tr>
                <tr>
                      <td>
                      xmlDocObject.getElementsByTagName(theN
odeName).attributes.getNamedItem(“id”).value
                      </td>
```

```
                        <td>
                              <span id="attributesid">
                              </span>
                        </td>
                  </tr>
            </tbody>
      </table>
 </body>
</html>
```

The script displays the data of the first record tag.

In accordance with the XML content, on clicking the button, the script generates the result as shown in figure 1.27.

Update table from XML

DOM Property	Output
xmlDocObject.getElementsByTagName(theNodeName)[0].nodeName	record
xmlDocObject.getElementsByTagName(theNodeName)[0].nodeValue	Ashley Andrews
xmlDocObject.getElementsByTagName(theNodeName)[0].parentNode.nodeName	students
xmlDocObject.getElementsByTagName(theNodeName)[0].childNodes[1].nodeName	name
xmlDocObject.getElementsByTagName(theNodeName).length	3
xmlDocObject.getElementsByTagName(theNodeName).attributes.getNamedItem("id").value	1

Figure 1.27: Output of Code Snippet 16

1.3 XMLHttpRequest Object

The XMLHttpRequest object, which is key to AJAX was discovered with IE 5.5 release in July 2000, but it came to light when AJAX and Web 2.0 came into existence.

XMLHttpRequest is an API used by scripting languages for transferring and manipulating XML data to and from a Web server. This is done by establishing an independent connection channel between client-side and server-side of a Web page using HTTP.

XMLHttpRequest object fetches data in XML, JSON, and plain data format.

1.3.1 XMLHttpRequest Methods

- **abort()**

  Cancels the current request.

  Syntax:

  ```
  xmlreqObj.abort()
  ```

  where,

  xmlreqObj is an XMLHttpRequest object

Session 01

Introduction to AJAX

- **getAllResponseHeaders()**

  Returns the complete set of HTTP headers as a string. It contains information about the server and the contents which are retrieved such as content-type.

  Syntax:

  ```
  xmlreqObj.getAllResponseHeaders()
  ```

  where,

  xmlreqObj is an XMLHttpRequest object

- **getResponseHeader(headerName)**

  Returns the value of the specified HTTP header.

  Syntax:

  ```
  xmlreqObj.getResponseHeader(name)
  ```

  where,

  xmlreqObj is an XMLHttpRequest object

- **open(method, URL, [, async [, user [, password]]])**

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

  The method parameter can have a value of 'GET', 'HEAD', or 'POST'.

  The request is to be handled asynchronously or not is specified by 'async' parameter. 'true' means without waiting for a response, the script processing carries on after the send() method, while 'false' means before continuing script processing, it waits for a response. The `XMLHttpRequest` object is initialize by this method.

  Syntax:

  ```
  xmlreqObj.open(method,URL)
  ```

  where,

  xmlreqObj is an XMLHttpRequest object

- **send(content)**

  Sends the request.

  Syntax:

  ```
  xmlreqObj.send(content)
  ```

  where,

  xmlreqObj is an `XMLHttpRequest` object

  content is the optional parameter and is the message

- **setRequestHeader(label, value)**

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

Syntax:

```
xmlreqObj.setRequestHeader(label,value)
```

where,

xmlreqObj is an `XMLHttpRequest` object

label is the name of the Header

value is the value to be assigned to the Header

1.3.2 XMLHttpRequest Properties

Following are the properties of XMLHttpRequest object:

- **onreadystatechange**

  It is an event handler which fires at every state change.

- **readyState**

  The current state of the XMLHttpRequest object is defined by readyState property.

  Table 1.1 lists the values for the readyState.

State	Description
0	Request is not initialized
1	Request has been set up
2	Request has been sent
3	Request is in process
4	Request is completed

Table 1.1: Values for readyState property of XMLHttpRequest Object

- **responseText**

  The response returned is a string or Text format.

- **responseXML**

  Returns the response as XML. This XML can be parsed.

- **status**

  Returns the status as a number. For example, 200 for “OK” and 404 for “Not Found”.

- **statusText**

  Returns the status as a string. For example, “OK” or “Not Found”.

  Figure 1.28 shows the contents of the Content.txt file.

Session 01

Introduction to AJAX

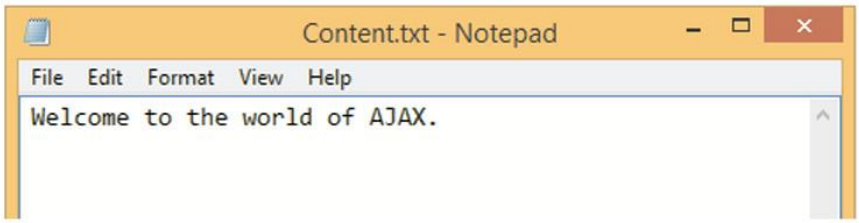


Figure 1.28: Contents of Content.txt File

Code Snippet 17 demonstrates the use of XMLHttpRequest object and its properties and methods. The AJAX.html page loads the content of the Content.txt file. The request is sent via XMLHttpRequest object asynchronously.

Code Snippet 17:

```
<html>
      <head>
      <script type="text/javascript">
         var ajaxRequest;   // The variable that makes AJAX
possible!

       function SendRequest(){
      try {
      // Common standard to create XMLHttpRequest instance
that most common //browsers except IE supports.
           ajaxRequest = new XMLHttpRequest();
       } catch (e) {
   // Older Internet Explorer Browsers doesn't support new
XMLHttpRequest()
  // For those the following code can be used to create the
instance //of XMLHttpRequest
  try {
      ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
   } catch (e){
       try {
       // Legacy way. Older than new ActiveXObject("Msxml2.
XMLHTTP")
       ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
        } catch (e){
             // Something went wrong
             alert("Your browser broke!");
             return false;
        }
 }
      }
```

Session 01

Introduction to AJAX

```
// Call back function. Which will be called when after the
completion of
 // AJAX request
 ajaxRequest.onreadystatechange = function (){
      if (ajaxRequest.readyState == 4) {
           varajaxDisplay=document.getElementById('ajaxDiv');
            ajaxDisplay.innerHTML = ajaxRequest.responseText;
      }
  }
   // specifying the name of the file to be opened
       ajaxRequest.open("GET", "Content.txt", true);
         ajaxRequest.send();
     }

</script>
 <title>
     AJAX example
</title>
</head>
<body>
           <div style="font-weight:bold;"> Data from a file will
be pasted here using AJAX</div>
           <br />
           <input type="button" value="Load file through AJAX"
onclick="SendRequest();"></input>
       <div id="ajaxDiv">
           </div>
     </body>
</html>
```

In the code, the XMLHttpRequest object is used with its methods and properties.

On event onload, the request is sent asynchronously and the data is requested from a file named, Content.txt.

The XMLHttpRequest object xmlDoc is created in the SendRequest() function. Next, the open() method is called where the type of method used in the connection, the URL, and whether the request is synchronous or asynchronous is specified. In the given code, the file name from where the contents are to be taken is specified as URL, 'GET' method is used, and 'true' indicates that the request is asynchronous.

Then, the send() method is called to send the request.

The response is accessed using the responseText property of the XMLHttpRequest object and the response is displayed in the html page. Figure 1.29 shows the output after clicking the button.

Session 01

Introduction to AJAX

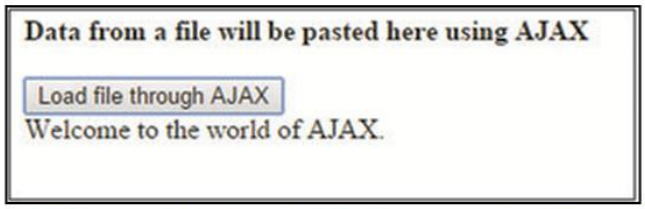


Figure 1.29: Output Showing the File Content Loaded Through AJAX

The file contents are loaded asynchronously and the div, ajaxDiv are updated to display the text from the file.

1.4 AJAX in Java with Servlet and JSP

Servlets provide a platform-independent, component-based method for building Web-based applications. Servlets can use Java APIs.

JavaServer Pages (JSP) are HTML pages with embedded Java code saved with the extension, '.jsp'. JSPs can use Java APIs.

To understand the use of JSP and Servlet for the asynchronous, consider the following example:

This example demonstrates two files, index.jsp and TimeServlet.java. The index.jsp page consists of a JavaScript code wherein the request is passed to the Servlet asynchronously and the current time of the server is received as the response.

Code Snippet 18 depicts the index.jsp page.

Code Snippet 18:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
<head>
<metahttp-equiv="Content-Type"content="text/html;charset=UTF-8">
  <title>Server time through AJAX</title>
  <style>
  body {font-family: verdana;margin:20px; font-size: 14px;}
  div.container {border: 1px solid black;
      padding:10px;width:460px;
  }
  p.result {color:red;font-weight:bold;}
  h3 {color:blue;}
  </style>

<script>
```

Session 01

Introduction to AJAX

```
function ajaxAsyncRequest(){
      try{
// Common standard to create XMLHttpRequest instance that most //
common browsers except IE supports.
            ajaxRequest = new XMLHttpRequest();
 }catch (e){
// Older Internet Explorer Browsers doesn't support new //
XMLHttpRequest()

// For those the following can be used to create the instance of
//XMLHttpRequest
      try{
            ajaxRequest = new ActiveXObject("Msxml2.XMLHTTP");
      }catch (e){
        try{
      // Legacy way. Older than new ActiveXObject("Msxml2.XMLHTTP")
            ajaxRequest = new ActiveXObject("Microsoft.XMLHTTP");
         }catch (e){
            // Something went wrong
            alert("Your browser broke!");
            return false;
         }
      }
  }
// Call back function. Which will be called when after the //
completion of AJAX request
   ajaxRequest.onreadystatechange = function (){
     if (ajaxRequest.readyState == 4) {
      var ajaxDisplay = document.getElementById('message');
      ajaxDisplay.innerHTML = ajaxRequest.responseText;
      }
    };
   ajaxRequest.open("GET","TimeServlet", true);
   ajaxRequest.send(null);
}
</script>
</head>
<body>
<div class="container">
The server time will be displayed on clicking the button
<input      type="button"      value="Show      Server      Time"
onclick='ajaxAsyncRequest();'/>
</div>
```

```
    <br>
    <div id="message">
    </div>
</body>
</html>
```

The index.html page contains a button, on click of which, the function ajaxAsyncRequest() is called to send an Asynchronous AJAX request to TimeServlet servlet. The XMLHttpRequest object, ajaxRequest, is responsible for making the server call and getting the response from the server.

Code Snippet 19 depicts the servlet class, TimeServlet.

Code Snippet 19:

```
import java.io.IOException;
import java.util.Date;
import javax.servlet.ServletException;
import javax.servlet.ServletOutputStream;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

public class TimeServlet extends HttpServlet {
    private static final long serialVersionUID = 1L;

    @Override
  public void doGet (HttpServletRequest request,HttpServletResponse
response) throws ServletException, IOException
    {
        ServletOutputStream out = response.getOutputStream();
            // AJAX Request
        Date currentTime= new Date();
         String message = String.format("Currently time is %tr on
%tD.",currentTime, currentTime);
        out.print(message);
        out.flush();
    }
}
```

In the servlet, the Java code is executed asynchronously and the response is sent back to the JSP page. The doGet() method returns the date and time of the server.

Code Snippet 20 shows the web.xml file.

Code Snippet 20:

```
<?xml version="1.0" encoding="UTF-8"?>

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
           xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
       version="3.1">
    <welcome-file-list>
        <welcome-file>index.html</welcome-file>
    </welcome-file-list>
    <servlet>
    <servlet-name>TimeServlet</servlet-name>
    <servlet-class>TimeServlet</servlet-class>
    </servlet>
    <servlet-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
</web-app>
```

In web.xml include the servlet, TimeServlet that has been implemented.

Figure 1.30 shows the output for the index.jsp page after clicking the button.

Figure 1.30: Output - Server Time Displayed Through AJAX

On clicking the Show Server Time button, an asynchronous request is sent to the server. The doGet() method of the servlet is executed which returns current time of the server.

Session 01

Introduction to AJAX

Check Your Progress

1. What does AJAX stands for?

(A)	Asynchronous Java and XML	(C)	Asynchronous JavaScript and XML
(B)	Asynchronous JSP and XSLT	(D)	Asynchronous JSP and XSD

2. Which of the following statements about the conventional Web applications are true?

(a)	The click, wait, and refresh model led to high request-response latency periods which in turn lead to slow responsiveness.
(b)	The user is forced to stay idle while the browser refreshes the Web page to display the results retrieved from the server.
(c)	Conventional Web applications also lack rich user interface.
(d)	The page refresh did not make existing browser contents to vanish.

(A)	a, b, c	(C)	a, b, d
(B)	a, c, d	(D)	b, c, d

3. Which of the following is not a property of the XMLHttpRequest Object?

(A)	state	(C)	status
(B)	readyState	(D)	responseText

4. Which of the following statements about the working of AJAX are true?

(a)	The callback function is specified in the responseXML property with the help of the send() method of XMLHttpRequest object.
(b)	A client event on the Web page invokes an event handler that updates the Web page instantaneously.
(c)	The callback function is called only after successful receipt of XML response from the server.
(d)	The XMLHttpRequest object's send() method invokes the server-side component that processes the AJAX request.
(e)	The callback function uses DOM to display data on a Web page.

Session 01

Introduction to AJAX

Check Your Progress

(A)	a, b, c	(C)	b, c, d
(B)	c, d, e	(D)	b, d, e

5. Which of the following statements about DOM are true?

(a)	DOM is a language-dependent standard object model that represents HTML or XML documents.
(b)	DOM creates the tree structure based on how the document objects are connected to each other.
(c)	DOM classifies every node as a specific type of node.
(d)	DOM facilitates dynamic modification of Web pages.
(e)	DOM provides iterators to iterate through the content of Web pages.

(A)	a, b, c	(C)	b, c, d
(B)	c, d, e	(D)	b, d, e

6. Which of the following is not a parameter of open() method?

(A)	URL	(C)	username
(B)	password	(D)	requestName

Session 01

Introduction to AJAX

Answers

1.	C
2.	A
3.	A
4.	B
5.	C
6.	D

Session 01

Introduction to AJAX

Summary

- Conventional Web applications communicate with the server synchronously. Once an HTTP request is generated, the user can no longer interact with the application.
- Synchronous communication between the client and server introduces long delays in the request-response cycle.
- AJAX comprises JavaScript, XML, DOM, and CSS. These technologies enable development of highly responsive and rich UI-based Web applications
- The Document Object Model is a platform and language-independent standard for representing HTML and XML documents.
- Using properties and methods that are supported by HTML/XML DOM API, the content of a HTML/XML can be manipulated dynamically at client-side.
- XMLHttpRequest is an API supported by JavaScripting for transferring and manipulating data to and from a Web server by establishing an asynchronous connection channel between client-side and server-side using HTTP.
- XMLHttpRequest object fetches data in XML, JSON, and plain data format.

ASK to LEARN

Questions *in your* mind?

are here to HELP

Post your questions in the ASK to LEARN section for solutions.

02 JSON and DWR

Welcome to the Session, **JSON and DWR**.

This session explains about JSON and its data types and objects. The session also describes Direct Web Remoting which allows a client browser to remotely invoke server functionalities and Reverse Ajax which is new feature of DWR.

In this Session, you will learn to:

- Explain JSON
- Describe the JSON data types and arrays
- Explain how to use JSON with AJAX
- Explain DWR and steps to use DWR
- Explain Reverse Ajax

Session 02

JSON and DWR

2.1 Introduction

AJAX is a group of interrelated technologies. Thus, JSON, DWR, and many other frameworks can be used with AJAX to simplify the work of a developer as well as to improve the performance and efficiency of an application.

2.2 JSON

JavaScript Object Notation (JSON), a subset of JavaScript language, is a text-based data format that supports data exchange between a Web browser and a server. It is available as libraries in many languages, such as C#, Java, Python, and so on.

JSON 3 is a modern JSON implementation compatible with a variety of JavaScript platforms, such as IE 6, Safari 2, Opera 7, and Netscape 6. The current version is 3.3.2. It is used for transforming Java arrays, beans, collections, maps, and XML to JSON. The native data types such as Date, Error, Regular Expression, and Function were not supported in the previous versions of JSON lib. However, in this version, converting from Date to String is possible.

XML is also a text based format that supports data interchange. However, JSON is preferred over XML due to the following reasons:

- JSON is lighter and faster as compared to XML.
- JSON objects are typed whereas XML objects are untyped. JSON supports data types such as string, number, array and boolean, whereas XML supports only the string data type.
- JSON code is native to JavaScript code. It is readily accessible to JavaScript code. However, XML code needs to be parsed and assigned to variables using tedious DOM operations.

Why use JSON?

JSON is faster and easier than XML for AJAX applications due to the following reasons:

Using XML

- Fetch an XML document
- Use the XML DOM to loop through the document
- Extract values and store in variables

Using JSON

- Fetch a JSON string
- JSON.Parse the JSON string

2.2.1 JSON Data Types

JSON's text-based data format is based on JavaScript's object notation.

Session 02

JSON and DWR

The elements of the object notation supported by JSON are as follows:

- **Object**

  An object is a collection of unordered name/value pairs. A JSON object is represented by {}.

- **Object Member**

  A JSON object member consists of a name/value pair, which is a combination of string and value. Members are separated by using commas. Object name and its value in a name/value pair are separated by a colon.

- **Array**

  A JSON array consists of elements or values that are separated by commas.

  Array indexes are zero(0)-based. All the elements in an array are enclosed in []. Each individual element of the array is enclosed in {} if there are multiple name/value pairs.

- **Value**

  A JSON value can be a string, a number, a boolean, an object, null, or an array.

- **String**

  A JSON string consists of Unicode characters except double quotes (‘), backslash (\), or control characters. A JSON string is enclosed within double quotes. For example, `{“myString”: “abcdef”}`

Code Snippet 1 presents the personal details of Jack Daniels in JSON notation, using JSON elements.

Code Snippet 1:

```
{
“fullname”: “Jack Daniels”,
“company”: “JSON Consulting”,
“age” : 20,
“email”’: [
{“type”: “work”, “value”:”jack.daniels@jsonconsult.com”},
 {“type”: “home”, “value”: “jack@hotmail.com”}
],
“contactno”: [
 {“type”: “work”, “value”: “123456”},
 {“type”: “fax”, “value”: “34567”},
 {“type”: “mobile”, “value”: “9987651111”}
],
 “addresses”: [
{“type”: “work”, “format”: “us”,
 “value”: “1234 Manhutton”},
{“type”: “home”, “format”: “us”,
“value”: “5678 SpringField”}
]
}
```

In Code Snippet 1, "fullname": "Jack Daniels" is an example of name/value pair or an object member. The email, contactno, and addresses are example of arrays that consists of name/value pairs separated by commas. Each array holds its elements as JSON objects in []. Each array element holds its name/value pairs within {}. In the email array, email[0] represents the first element: work, jackdaniels@jsonconsult.com. Code Snippet 1 uses two data types: string, number. For example, 'Jack Daniels' is a string and 20 is a number.

The other data types supported by JSON are listed in table 2.1.

Type	Description
Number	Double- precision floating-point format in JavaScript. For example, `{"myNum": 123.456}`
Boolean	True or False. For example, `{"myBool": true}`
Whitespace	Can be used between any pair of tokens
null	Variable with null (empty) value. For example, `{"myNull": null}`

Table 2.1: Other JSON supported data types

2.2.2 JSON Objects

JSON objects can be used in JavaScript without any additional effort because JSON is a subset of JavaScript. A JavaScript variable can be declared and assigned a JSON-formatted data structure to the variable. A particular element of a JSON-formatted object can be accessed and modified by using the 'dot' notation from JavaScript code.

The different types of JSON objects that can be accessed using 'dot' notations are as follows:

- **JSON object elements**

  JSON object elements can be accessed and modified using 'dot' notation. For example, Student.name can be used to access the name attribute of a Student object.

- **Array elements**

  An array element can be accessed and modified by using an index. Attribute of an array element can be accessed as array_name[index].attribute. If multiple name/value pairs are present instead of single attribute, these will be enclosed in curly braces {}.

- **Nested JSON objects**

  A nested object can be accessed and modified by using 'dot' notation as first_object.second_object.attribute.

Code Snippet 2 demonstrates use of 'dot' notation to access different types of JSON objects.

Code Snippet 2:

```
email[0].value;
email[1].value;
```

Each array holds its elements as JSON objects in square braces []. Each array element holds its name/value pairs within curly braces {}. The notation email[0] represents the first element and the value for it will be jackdaniels@jsonconsult.com. While email[1] represents the second element which is jack@hotmail.com.

2.2.3 Receiving JSON Data from the Server

In an AJAX application, both client and server can receive and process JSON text.

When a client requests a server for some data, the server can send the requested data in JSON format, XML format, or as plain text. If the server returns the data in JSON format, the client receives the data in string format. The client uses the following steps to process the data:

1. The client converts the string data into a JavaScript object by using the statement, eval('('+request. responseText+')').
2. The client can access and modify properties of the converted JavaScript object.

Figure 2.1 shows the client requesting for JSON data.

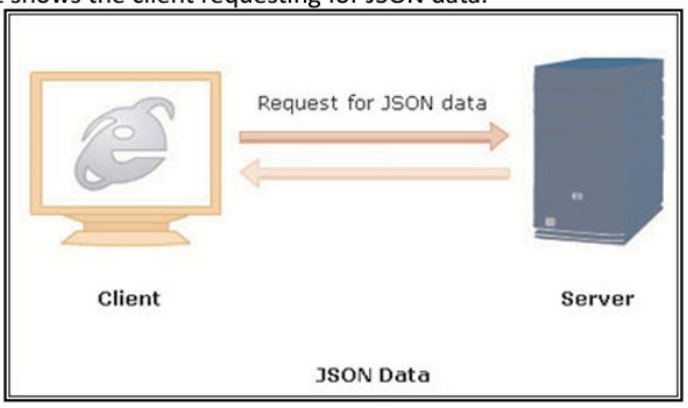


Figure 2.1: Client requesting for JSON data

2.2.4 Receiving JSON Data from the Client

When a client sends the processed data to the server, the client converts the JSON object to JSON text. The server uses the following steps to process the JSON text received from the client:

1. The server finds an appropriate parser depending on the programming language of the server-side program. For example, if the server application is written using JSP/ Servlets, the server uses a parser from the org.json package.
2. The parser interprets JSON text into a language that the server understands.

Figure 2.2 shows receiving of JSON data from server.

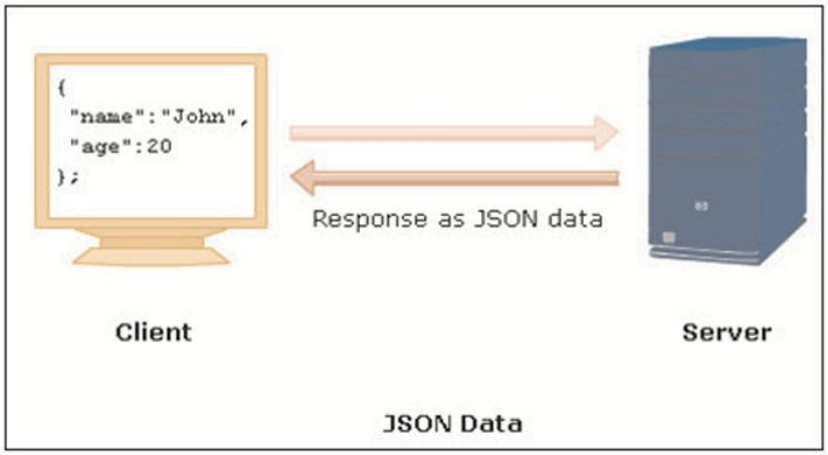


Figure 2.2: Receiving JSON data from server

2.2.5 'Assignment' Technique

The three techniques of receiving JSON formatted text from server are: Assignment, Callback, and Parse.

The steps followed by the 'Assignment' technique to receive JSON data are as follows:

1. A JavaScript variable is assigned the JSON formatted text from server.
2. JavaScript's eval() method is invoked to convert the JSON text into a JSON object.
3. The 'dot' notation is used to access properties of the JSON object.

Code Snippet 3 demonstrates the 'Assignment' technique to receive JSON data.

Code Snippet 3:

```
var JSONres= "week = { 'weekday': 'Monday'}";
eval(JSONres);
document.writeln(week.weekday);
```

2.2.6 'Callback' Technique

In the 'Callback technique, a pre-defined function is called and the server response is passed in JSON format as the first argument to the function. The Callback technique is used to receive JSON data from Web sites of external domains.

Code Snippet 4 demonstrates the 'Callback' technique to receive JSON data.

Code Snippet 4:

```
function processData(inputJSON) {
      document.writlen(inputJSON.weekday); //output Monday
}
var week= "processData( { 'weekday' : 'Monday' } );
eval(week);
```

The code defines a callback function with JSON object as argument that Outputs 'Monday'. Next, it passes JSON object as argument to processData() function and assigns it to JavaScript variable. Finally, it executes the JavaScript code.

2.2.7 'Parse' Technique

The 'Parse' technique parses and executes only JSON text that comes from the server as part of response text. Therefore, this technique of receiving JSON data is the safest. The JSON method parseJSON() is used to parse and execute JSON text that is received as a response from the server.

Code Snippet 5 demonstrates the 'Parse' technique to receive JSON data.

Code Snippet 5:

```
JSONresponse = '{"weekday" : "Monday" }';
object=JSONresponse.parseJSON();
document.writeln(object.weekday);
```

The code assigns JSON text to JSONresponse. Next, it parses JSON text and displays the result.

Note - The method parseJSON() is part of JSON specification, but it is not yet part of JavaScript specification. It will be incorporated in the next version of JavaScript.

2.2.8 Sending JSON Data

To send a JSON object as part of a client request, convert the JSON object into a string data by using toJSONString() and then, use GET or POST method with XMLHttpRequest object.

GET request method allows transferring of JSON data from client to server, but it compromises security of data. Besides, it does not allow sending large amount of data. If large amount of confidential data is to be sent, use the POST request method.

Note - In contrast to JSON 2, JSON 3 does not,

- add toJSON() methods to the Boolean, Number, and String prototypes. These are not part of any standard, and are made redundant by the design of the stringify() implementation.
- add toJSON() or toISOString() methods to Date.prototype.

Session 02

JSON and DWR

The server can generate JSON data and send it to client as part of response text.

The steps for sending JSON data are as follows:

1. Create JSONObject Java object
2. Add name/value pairs to the JSONObject using put()
3. Convert it to String type using toString()
4. Send it to the client with content-type as 'text/plain' or 'text/json'

Code Snippet 6 demonstrates the steps for sending JSON data.

Code Snippet 6:

```
var JSONres= "week = { 'weekday': 'Monday'}";
eval(JSONres);
document.writeln(week.weekday);
```

The code creates a JSON object. Next, it adds a name/value pair to the JSON object and converts the JSON object to String.

2.2.9 JSON with AJAX

JSON can be used to pass data asynchronously using AJAX between the client and server. For example, consider a Website which updates live sports scores. The scores should be stored on the server so that they can be retrieved by the Web page when required. Thus, for this, JSON formatted data can be used.

JSON files, when necessary, can be accessed and parsed by JavaScript using AJAX and then, the two tasks to be performed are as follows:

- The parsed values are stored in variables for further processing before displaying them on the Web page.
- The value gets displayed on the Web page as the data is directly assigned to the DOM elements of JavaScript.

Consider a scenario where the Web page is updated asynchronously using JSON and JavaScript.

Following are the steps to be followed for using JSON:

1. Download the JSON library from the http://mvnrepository.com/artifact/net.sf.json-lib/json-lib/2.4 or any other freely available site.
2. Open Netbeans IDE and create a Web application.
3. The JSON jar file is to be added in the Libraries folder of the project. Right-click the Libraries folder and select Add JAR/Folder and select the JSON jar file from the file dialog box.

4. Right-click the Web Pages folder and then, select New → JSP. Name it as index.jsp.
5. Right-click the Web Pages folder and then, select New → JSON file. Name it as Objects.json.

The directory structure of the Web application is as shown in figure 2.3.

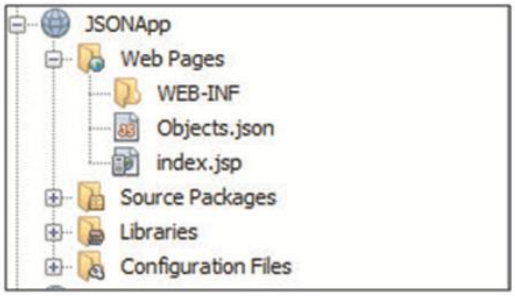


Figure 2.3: Directory Structure

Code Snippet 7 demonstrates the JSON file named Objects.json which consists of nested objects.

Code Snippet 7:

```
{
"Objects": [
{
"Students": [
{
"StudentID":"1",
"Name": "Andrews"
},
{
"StudentID":"2",
"Name":"Ashley"
},
{
"StudentID":"3",
"Name":"Jeff"
}
]
},
{
"Subjects":[
{
"ID":"1",
"Name": "Networking"
},
```

Session 02

JSON and DWR

```
{
"ID":"2",
"Name":"C-Programming"
},
{
"ID":"3",
"Name":"Java"
}
]
}
]
}
```

There are two types of JSON objects inside the Objects.json file. The nested objects are Students and Subjects.

The index.jsp file accesses the data from the Objects.json file and displays it accordingly.

Code Snippet 8 demonstrates the index.jsp file.

Code Snippet 8:

```
<%@page contentType="text/html; charset=UTF-8"%>
<html>
<head>
<script type="text/javascript">
function loadJSON(object){
  var data_file
   data_file = "Objects.json";
  var ajax_request = new XMLHttpRequest();
  try{
      // Opera 8.0+, Firefox, Chrome, Safari
      ajax_request = new XMLHttpRequest();
   }catch (e){
      // Internet Explorer Browsers
      try{
         ajax_request = new ActiveXObject("Msxml2.XMLHTTP");
      }catch (e) {
         try{
           ajax_request = new ActiveXObject("Microsoft.XMLHTTP");
         }catch (e){
            // Something went wrong
            alert("Your browser broke!");
            return false;
         }
      }
   }
```

```
   ajax_request.onreadystatechange  = function(){
      if (ajax_request.readyState == 4  ){
        // Javascript function JSON.parse to parse JSON data
        var jsonObject = JSON.parse(ajax_request.responseText);
        var str="<table border=1 width='40%'>";

        // jsonObject variable now contains the data structure
        if(object=="Student"){
     for(var i=0; i< jsonObject.Objects[0].Students.length; i++){
str  =  str  +  "  <tr><td>"  +  jsonObject.Objects[0].Students[i].
StudentID + "</td>";
           str = str + "<td>" + jsonObject.Objects[0].Students[i].
Name + "</td> </tr>";
             }
        }
        else {
     for(var i=0; i< jsonObject.Objects[1].Subjects.length; i++){
   str = str + " <tr><td>" + jsonObject.Objects[1].Subjects[i].ID
+ "</td>";
    str = str + "<td>" + jsonObject.Objects[1].Subjects[i].Name +
"</td> </tr>";
             }
        }
        str=str +"</table>";
       document.getElementById("container").innerHTML=str;
      }
     };
   ajax_request.open("GET", data_file, true);
   ajax_request.send();
}
</script>
<title>JSON Demo</title>
</head>
<body>
<h1>Details </h1>
<button type="button" id="btnStudent"
onclick="loadJSON('Student')">Student Details </button>
<button type="button" id="btnSubject"
onclick="loadJSON('Subject')">Subject Details </button>
<br/>
<div id="container">
</div>
</body>
</html>
```

In Code Snippet 8, the details of students and subjects are displayed on click of Student Details and Subject Details buttons, respectively. The loadJSON() method is called to update the Web page with AJAX request. The data_file variable is used to store the name of the Objects.json file. The jsonObject variable is used to store the JSON file value in the form of DOM object. The details of Students and Subjects are loaded in the div tag with id, container.

Figure 2.4 shows the output of index.jsp Web page.

Figure 2.4: Output of index.html Web page

On clicking the Student Details button, the loadJSON() method is called in which details about students in the Objects.json file will be displayed in a tabular format as shown in figure 2.5.

Figure 2.5: Output on Click of the Student Details button

The div container is updated asynchronously with the Student details retrieved from the Objects.json file.

Figure 2.6 shows the subject details displayed on click of the Subject Details button.

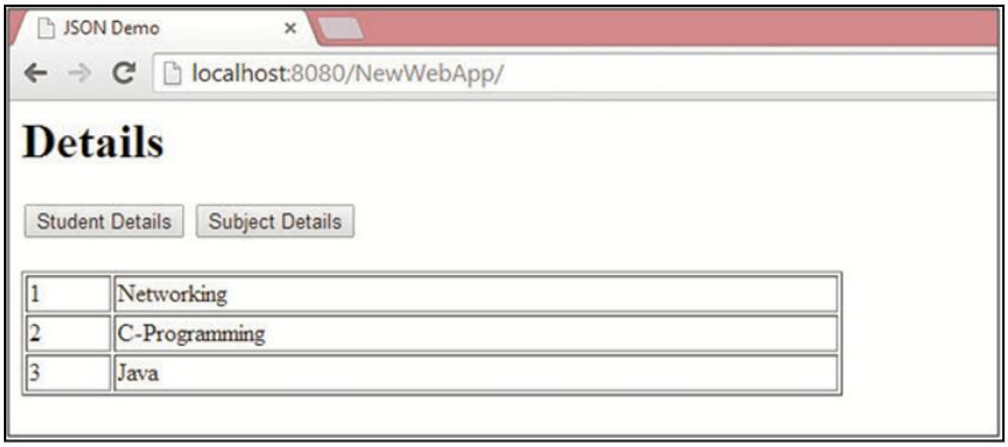


Figure 2.6: Output on click of the Subject Details button

Thus, in the output, the whole Web page is not updated but only the div container is getting updated on click of a button.

2.3 Direct Web Remoting (DWR)

Direct Web Remoting (DWR) is an AJAX framework that allows clients to remotely access server functionalities. It is based on Java technology. It can generate JavaScript code from Java classes.

Client applications cannot access the server-side Java classes directly. To allow the client applications to access the server-side Java classes, DWR converts the Java classes into JavaScript code. The developer can decide which Java classes would be accessible to the client browser by configuring DWR.

Session 02

JSON and DWR

Figure 2.7 shows the DWR framework.

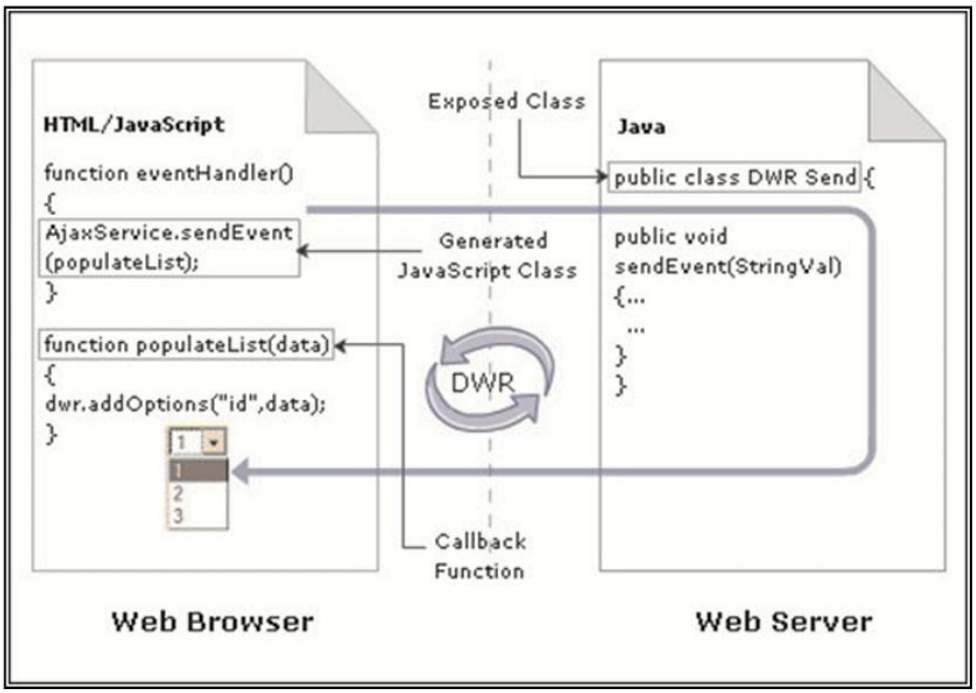


Figure 2.7: DWR Framework

2.3.1 Features of Direct Web Remoting

DWR hides the complexities of AJAX technology and relieves the developer from writing additional code for providing AJAX functionalities. For example, DWR automatically manages the XMLHttpRequest object, object serialization, and remote invocation of Java classes in the Web server. DWR library is freely available.

Features that make DWR more useful compared to similar products are as follows:

- **Support for marshalling/unmarshalling of objects**: DWR library can marshal/unmarshal any JavaScript object that is exchanged between a client and a server application. During the marshal/unmarshal process, DWR converts datatypes of Java to JavaScript and JavaScript to Java.
- **Integration with popular frameworks**: DWR library supports Spring, Hibernate, Struts, and JSF frameworks.
- **Documentation**: DWR provides comprehensive documentation for its libraries.
- **Reverse AJAX**: DWR allows the server to connect with clients and send updated data to clients asynchronously.

2.3.2 Components of DWR

DWR exposes methods of server-side Java classes to client-side JavaScript code. DWR consists of the following two components:

- **Java Servlet**: Processes client requests and sends responses back to the client. DWR contains a runtime library that helps the servlet to process requests and responses.
- **JavaScript Code**: Runs in the client application. It can dynamically update the Web page with the help of a JavaScript library that is part of the DWR architecture.

Figure 2.8 shows the components of DWR.

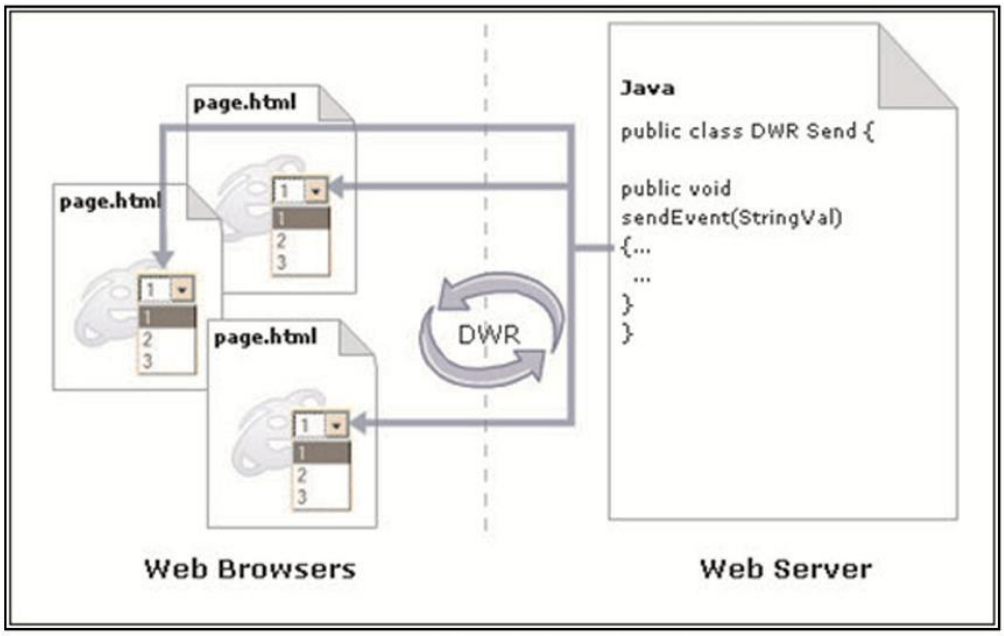


Figure 2.8: Components of DWR

2.3.3 Working of DWR

In DWR-based AJAX applications, the DWR servlet dynamically generates JavaScript classes for each exposed server-side Java class.

The workflow of DWR can be summarized as follows:

1. DWR dynamically generates the client-side JavaScript code, called stub. The stub handles remote communication between browser and server.
2. The JavaScript code at the client-side routes calls to the server-side methods through the stub. The DWR servlet receives the client requests and calls the appropriate server-side methods.

3. DWR converts the data types of method parameters received from the client application to Java data types.

4. DWR converts the data types of return values from the server-side methods to JavaScript data types. Since data types of Java and JavaScript are different, conversion of the data types is required.

5. DWR asynchronously invokes the client-side callback methods by using the XMLHttpRequest object and sends the server response from the servlet to the callback function.

Figure 2.9 shows the working of DWR.

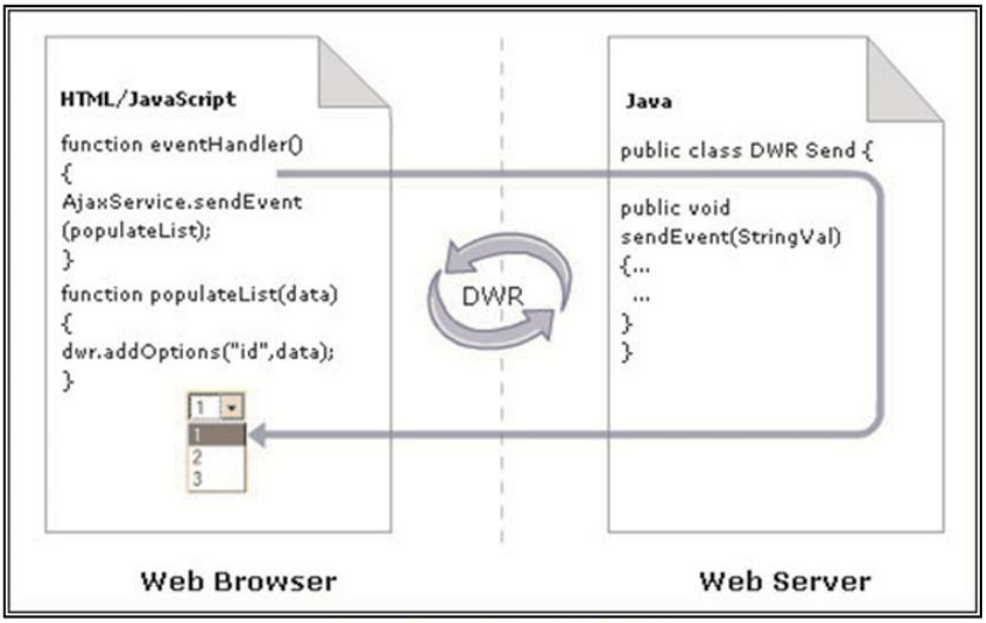


Figure 2.9: Working of DWR

2.3.4 Handling Asynchronous AJAX Calls

In an AJAX application, a client sends a request asynchronously by using JavaScript code. The server that runs Java servlets processes the request synchronously, and sends back the response to the client. The processing at server is synchronous because Java technology is synchronous in nature.

To enable asynchronous communication between a client and a server, perform the following steps:

1. Declare a callback function in the client-side code. The callback function handles responses from the server.

2. Register the callback function with the server. To register the callback function, it is necessary to pass an additional parameter when calling the remote methods. After making an asynchronous request, the client can carry out other tasks. When the server completes synchronous processing of the client request, the server invokes the callback function to pass the response data.

Code Snippet 9 demonstrates how to declare a callback function in JavaScript-based client and register it with server.

Code Snippet 9:

```
//Declare callback function on client-side
function handleGetName(str){
alert(str);
 }

//Server-side Java Class that will be remotely accessed by browser
public class Student{
public String getName(String name) {
...
} }
// Invoke remotely getName() from client-side and
// register callback function getName()
StudentJavaScript.getName(42, handleGetName);
```

In Code Snippet 9, handleGetName() is declared as callback function while '42' is the parameter passed to the Java function getName(). The server-side Java class, Student, is exposed to the client application. DWR generates a JavaScript class, StudentJavaScript, from this class.

The client calls the getName() method of the remote Student class by using the generated JavaScript class. It also registers the callback method, handleGetName(), with the server by passing it as argument to getName().

2.3.5 Steps of Using DWR in AJAX

Consider a DWR Web application in which user is required to register. The Web application displays the account registration page. The registration page contains textboxes to accept username, password, e-mail address, and a Submit button. When a user enters his/her details and clicks the Submit button, the page waits for a response from the Web server. If the username already exists, the user will be notified about the same on leaving the text box focus.

Following steps demonstrate how an AJAX based application can be developed using DWR:

1. Create a new Web application in the NetBeans IDE named DWRRegistration.
2. Create a new JSP page named RegistrationForm.jsp in Web Pages folder.

3. Download the latest DWR jar (version 3.0) file from DWR official site: http://directwebremoting.org/dwr/downloads
4. Add the dwr-jar file to the Libraries folder of the Web application.
5. Download the commons-logging-1.3.jar from http://www.java2s.com/Code/Jar/c/Downloadcommonslogging13jar.htm
6. Add the commons-logging-1.3.jar to the Libraries folder of the Web application.

The directory structure of the Web application is as shown in figure 2.10.

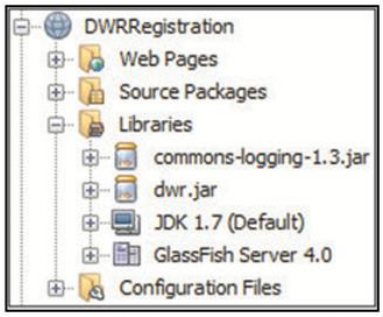


Figure 2.10: Directory structure

The files to be created are as follows:

- **Server-Side Java Class**

  Code Snippet 10 demonstrates a server-side Java class that will be exposed for generation of a JavaScript class. The getMessage() method will be used in the RegistrationForm.jsp page.

Code Snippet 10:

```
package com;

public class Registration {
    private String uname;

    public String getUname() {
        return uname;
    }
   public void setUname(String uname) {
       this.uname = uname;
   }
   public String getMessage(String username){
         String str = null;
         String[] unames= new String[4];
         unames[0]=”jsmith”;
         unames[1]=”ashandrews”;
```

```
        unames[2]="johnsmith";
        unames[3]="maryjoe";
        for (String name : unames) {
           if(name.equalsIgnoreCase(username))
           {
             str= "Username already exists";
             break;
           }
           else
              str= "";
            }
       return str;
    }
}
```

The getMessage() method takes the username as parameter and checks in the array if it already exists. The message is returned by the method accordingly.

- **Modify web.xml to add the DwrServlet mapping**

The DwrServlet in an AJAX application processes requests and responses. Mapping for DwrServlet must be added in the web.xml file, located in the WEB-INF folder of the Web application.

The tag <servlet-name> indicates a DWR servlet named dwr-invoker. The tag <servlet-class> indicates actual class name of the DwrServlet. The DwrServlet is located in the org.directWebremoting.servlet package. The initial parameters in the tag <init-param> indicate that DWR servlet can be in debug mode. Finally, the <servlet-mapping> tag maps the servlet to the url pattern /dwr/*.

Code Snippet 11 shows the configuration values for web.xml.

Code Snippet 11:

```
<servlet>
<servlet-name>dwr-invoker</servlet-name>
<servlet-class>org.directWebremoting.servlet.DWRServlet</
servlet-class>
<init-param>
<param-name>debug</param-name>
<param-value>true</param-value>
</init-param>
 </servlet>
<servlet-mapping>
<servlet-name>dwr-invoker</servlet-name>
<url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

Session 02

JSON and DWR

- **Configure DWR**

Add one more XML file named dwr.xml in the WEB-INF folder. This file is the DWR configuration file.

Code Snippet 12 demonstrates the dwr.xml file which determines which Java classes DWR can convert to JavaScript classes.

Code Snippet 12:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE dwr PUBLIC "-//GetAhead Limited//DTD Direct Web
Remoting 3.0//EN" "http://getahead.org/dwr/dwr30.dtd">

<dwr>
<allow>
<convert converter="bean" match="com.Registration"/>
<create creator="new" javascript="RegistrationJS"
class="com.Registration">
<!-- here RegistrationJS is dynamic javascript class generated
by dwr-->
<param name="class" value="com.Registration"/>
<include method="getMessage"/>
</create>
</allow>
</dwr>
```

The tag, <create> generates a JavaScript class RegistrationJS from the com.Registration class defined in the <param> tag. The <include> and <exclude> tags can be used to specify the methods to be included and excluded in the JavaScript class from the Java class. Converters are required for custom JavaBean and parameters. For example, one can specify a Bean converter as demonstrated in the <convert> tag in the code.

- **Client-Side JavaScript**

The JavaScript files engine.js and util.js should be included in the client-side application. These files are part of the DWR JavaScript library. The engine.js file provides core DWR functionalities. The util.js file provides DWR utilities. The file RegistrationJS.js is the generated JavaScript file.

Code Snippet 13 demonstrates the inclusion of engine.js and util.js files in a JSP page.

Code Snippet 13:

```
<script type='text/javascript' src='/DWRRegistration/dwr/
engine.js'></script>
<script type='text/javascript' src='/DWRRegistration/dwr/
util.js'></script>
<script type='text/javascript' src='/DWRRegistration/dwr/
interface/RegistrationJS.js'></script>
```

Session 02

JSON and DWR

Code Snippet 14 demonstrates the RegistrationForm.jsp page where the data from the server is retrieved through DWR.

Code Snippet 14:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>User Registration Form</title>
<script   type='text/javascript'   src='/DWRRegistration/dwr/
engine.js'></script>
<script type='text/javascript' src='/DWRRegistration/dwr/util.
js'></script>
<script   type='text/javascript'   src='/DWRRegistration/dwr/
interface/RegistrationJS.js'></script>
<script type='text/javascript'>
function update(){
   var uname;
    //debugger;
    uname=document.getElementById("txtUname").value;
    Registration.getMessage(uname,function(data) {
    dwr.util.setValue("message", data);
    });
}
function register(){
    alert("Registration Completed");
 }

</script>
</head>
<body>
<form method='post'>
       Username:
          <input type='text' id='txtUname' onblur="update()">
<span id="message" style="color:red"></span><br/>
        Password: 
       <input type='password' id='txtPwd'><br/>
        Email Id:  
       <input type='text' id="email"><br/>
<input type="submit" value="Submit" onclick="register();">
</form>
</body>
</html>
```

The update method is called on blur event of the text box. The update function calls the JavaScript class RegistrationJS.js which is generated automatically by DWR.

Figure 2.11 shows the output of RegistrationForm.jsp.

Figure 2.11: Output on Execution of the Application

When an existing username is entered for registration, the appropriate message is displayed as shown in figure 2.12.

Figure 2.12: Output when a Username Already Exists

The message is retrieved from the server after checking the existing names in the array. If the name does not exist, no such message is displayed as shown in figure 2.13.

Figure 2.13: Output when Username Does Not Exist

Session 02

JSON and DWR

DWR enables Java on the server as well as the JavaScript within a browser. This allows an equal interaction of calls with one another. DWR is a simple AJAX for Java.

DWR generates JavaScript for permitting Web browsers to safely call into Java code similar to something that runs on a local machine. It assembles data collections such as POJOs, XML, and binary data similar to images and PDF files. The important factors involved are the security policies that are included with the item.

With Reverse Ajax, DWR permits Java code to run on a server to use client side APIs. This is used for publishing updates within arbitrary groups of browsers. It permits a two way interaction, that is, a browser calling a server and a server calling a browser. DWR supports Comet, Polling, and Piggyback (that transmits data along with normal requests) as way of publishing it within browsers.

DWR integrates with Spring, Struts, Guice, Hibernate, and other such technologies. It is an Open Source technology available under the Apache Software License v2.

2.3.6 Reverse Ajax

DWR continues on the same theme with release 2.0 version onwards. It removes the problem where developer has to pass the information. To describe the asynchronous transfer of messages from browser to server and vice-versa, DWR introduced a new term, 'reverse Ajax'. The Reverse Ajax feature neatly wraps all the existing techniques and selects the best method for the user.

DWR supports three techniques which are as follows:

- **Polling**

  At frequent and regular intervals, request is sent by browser to the server to check if the page has been updated.

- **Piggyback**

  In this technique, server waits for the browser request if it has any update to send and in addition to the requested information, also sends the update.

- **Comet**

  The server keeps the reply open after a request has been received from the browser, so that it can pass the information when the request arrives. This technique is also called as a long-lived HTTP connection.

By comparing the three techniques, polling and comet have the disadvantage of producing extra network traffic, while piggyback does not. While using the polling or comet techniques, updates would occur quicker than when using piggyback from the server to the browser.

In the Web page, for allowing active Reverse Ajax (ARA) to occur, the following line should be included:

```
<body onload="dwr.engine.setActiveReverseAjax(true);">
```

Code Snippet 15 should be included in the web.xml file.

Code Snippet 15:

```
<init-param>
      <param-name>pollAndCometEnabled</param-name>
      <param-value>true</param-value>
</init-param>
```

For Comet/Polling DWR can be configured when extra load is acceptable and for having faster response time. This mode is known as active Reverse Ajax. By default Reverse Ajax is turned off in the DWR and thus, allows only piggyback mechanism by default. It is easy to use Reverse Ajax and DWR from a thread outside of DWR.

2.3.7 Alternate Technologies

DWR is a remoting technology that is based on Java. It uses a custom protocol for remoting. Alternative remoting technologies available are: XML-RPC, JSON-RPC, and SOAP.

- XML-RPC is a lightweight protocol that can exchange structured information in XML format between nodes of a distributed environment. It is independent of any programming language.
- Simple Object Access Protocol (SOAP) is a protocol that exchanges XML-based messages in distributed environment using HTTP/HTTPS. SOAP provides a basic messaging framework for Web services.
- JSON-RPC is a protocol that allows bidirectional communication between server and client, unlike XML- RPC or SOAP. JSON-RPC allows peer-to-peer communication between server and client. It contains only a few commands and data types.

Session 02

JSON and DWR

Check Your Progress

1. Which of the following statements with respect to DWR architecture are true?

(a)	DWR is an MVC framework.
(b)	DWR is based on Microsoft technology.
(c)	DWR manipulates XMLHttpRequest object.
(d)	DWR can dynamically generate JavaScript code from Java code.
(e)	DWR converts Java data types to JavaScript data types.

(A)	a, b	(C)	b, d
(B)	d, e	(D)	c, e

2. Which of the following Code Snippet correctly handles asynchronous AJAX calls?

(A)	function handleGetName(str){
	alert(str);
	}
	public class Student{
	public String getName(String name) {...}
	}
	StudentJavaScript.getName(42, handleGetName);

(B)	function handleGetName(str){
	alert(str);
	}
	JavaScript.getName(handleGetName);

Session 02

JSON and DWR

Check Your Progress

(C)	function handleGetName(str){
	alert(str);
	}
	public class Student{
	public String getName(String name) {...}
	}
	Student.(42, handleGetName);

(D)	function handleGetName(str){
	alert(str);
	}
	public class Student{
	public String getName(String name) {...}
	}

3. Match the following descriptions with the corresponding JSON element.

Description		JSON Element	
(a)	Each element is separated by comma	(1)	Object
(b)	Elements are enclosed in []	(2)	Object Member
(c)	Collection of unordered name/value pairs	(3)	Array
(d)	Can be string, number, or Boolean	(4)	String
(e)	Consists of UNICODE characters	(5)	Value

(A)	a-2, b-3, c-1, d-5, e-4	(C)	a-3, b-2, c-5, d-1, e-4
(B)	a-4, b-3, c-1, d-5, e-2	(D)	a-5, b-4, c-3, d-2, e-1

Session 02

JSON and DWR

Check Your Progress

4. Which of the following Code Snippet will send JSON data to client?

(A)	JSONObject jsonObject = new JSONObject(); jsonObject.toString();

(B)	JSON jsonObject = new JSONObject();
	jsonObject.put("JSON", "Hello from JSON!");
	jsonObject.toString();

(C)	JSONObject jsonObject = new JSONObject();
	jsonObject.put("JSON", "Hello from JSON!");
	jsonObject.toString();

(D)	JSONObject jsonObject = new JSONObject(); jsonObject.put("JSON", "Hello from JSON!");

Session 02

JSON and DWR

Answers

1.	B
2.	A
3.	A
4.	C

Session 02

JSON and DWR

Summary

- JSON is a subset of JavaScript language that is used for interchanging of data between a browser and the server. It is easier to manipulate as compared to XML.
- JSON supports data types such as string, number, array, and boolean.
- JSON can be used to pass data asynchronously using AJAX between the client and server.
- DWR is an AJAX framework that can generate JavaScript code from Java classes. It is based on Java technology.
- A client browser can access remote Java classes that run in the Web server by invoking the generated JavaScript functions.
- The Java classes that would be accessible to the client browser can be decided by configuring DWR.
- The Reverse Ajax feature introduced in DWR 2.0 neatly wraps all the existing techniques and selects best method for the user.

03 AJAX Client Frameworks-I

Welcome to the Session, **AJAX Client Frameworks-I**.

This session explains about AJAX client frameworks such as Prototype, script.aculo.us, and Dojo. It describes the usage and implementations of these frameworks with AJAX.

In this Session, you will learn to:

- Explain AJAX Client frameworks
- Explain how to use Prototype with AJAX
- Explain how to use script.aculo.us with AJAX
- Explain how to use Dojo with AJAX

Session 03

AJAX Client Frameworks-I

3.1 Introduction

AJAX has brought a lot of frameworks to the bill board. Cross-browser APIs of these frameworks are used for communications, event handling, DOM manipulation, and also to alter the graphic loaded elements.

The overall list of AJAX frameworks can be classified into two categories:

- **Server-Side Frameworks** - Server-side frameworks, as the name suggests, are used on the server-side. One can use the same language APIs provided by the framework to perform the client-side activities. On deployment, these client-side activities coded on the server-side get translated to normal AJAX functionalities.

  Google Web Toolkit (GWT) is an example of a server-side framework. It is used as a normal Java Swing type of programming on the server-side in Web pages using the GWT API. Other server-side frameworks include Yahoo Toolkit, DWR, and so on.
- **Client-Side Frameworks** - These provide packages and APIs which help a developer to implement functionalities such as cross-browser compatibility, multiple request handling, graceful degradation (in case the browser does not support the XMLHTTPRequest object), exception and error handling, and a lot of extra features which can be found in free JavaScript code snippets over the Internet.

  The examples of a client side framework are Prototype, DOJO Toolkit, Rico, Script.aculo.us, and so on.

3.2 AJAX Client Frameworks

AJAX is generally implemented using the XMLHTTPRequest object in a JavaScript file and then, some callback function is called to handle the asynchronous requests. This process is sufficient for a simple AJAX page which has one or two requests. However, when it comes to a large application, much more complex functionality is required. A complex application requires a lot of JavaScript code which can be avoided by use of client-side frameworks.

These frameworks provide APIs and packages to implement the complex functionalities and also, provide extra features. The extra features may include Widgets or even an IDE. The small interfaces or components which can be used easily through client-side framework are known as Widgets.

Following are some examples of Widgets:

- Auto completing combo box
- Effects such as fade, zoom in/out, and so on
- Menus
- Tabbed pane windows

Session 03

AJAX Client Frameworks-I

3.2.1 Advantages of Client-Side Frameworks

Besides the features mentioned earlier, the complete flexibility is in the developers' hands.

A combination of features of widgets can be used and at the same time provide extra flexibility to use JavaScript.

Some frameworks even provide a GUI to build Web pages with AJAX components with drag and drop style.

3.3 Prototype

Prototype is a JavaScript Framework which helps to easily develop dynamic Web applications. The framework was developed by Sam Stephenson.

It is a JavaScript library which enables a developer to manipulate DOM in a very easy manner as per the requirement and it also provides cross-browser safety.

Prototype features:

- Provides built-in types with useful methods and extends DOM elements.
- Class-style OOP including inheritance support is provided.
- Advanced event management support is provided.
- Powerful AJAX features are included.

3.3.1 AJAX Response Callbacks

Prototype handles JavaScript code returned from a server efficiently and also provides helper classes for polling.

The global 'Ajax' object contains the AJAX functionality. It provides the necessary methods to handle AJAX requests and responses.

Following are the classes used in Prototype based AJAX:

- **Ajax.Request**

  The request is initiated and processed by the Request object. It is a class for making HTTP requests which handles the boilerplate, life-cycle request, and even the callback functions can be embedded as per requirement.

  Syntax:

  ```
  new Ajax.Request(url[,options]);
  ```

  The first parameter is the URL and second parameter which is optional is options hash. For example: `new Ajax.Request('/some_url', { method:'get' });`

  The 'method' option refers to the HTTP method to be used; default method is POST.

Note that AJAX requests are asynchronous by default. Hence, callbacks are required to will handle the data from a response. The callback methods are passed in the options hash while making a request. Callback functions such as onSucess/onFailure can be used.

The request is initiated as soon as the object is created, then it processes throughout its life-cycle. The life-cycle has the following states:

- Created
- Initialized
- Request sent
- Response received (can occur several times based on arrival of packets)
- Response received, request complete

Table 3.1 lists the callback functions for Prototype's 'Ajax'.

Function	Description
onCreate	It is a callback reserved to AJAX global responders
onUninitialized	It is mapped on created
onLoading	It is mapped on initialized
onLoaded	It is mapped on request sent
onInteractive	It is mapped on response is received
onXYZ	Returns a numerical status code
onComplete	It is called only after first callback

Table 3.1: Callback Functions in Prototype's 'Ajax'

Callbacks are called at various points in the life-cycle of a request and always feature the same list of arguments. The callbacks are passed to requesters along with their other options. Table 3.2 lists some common options of the callbacks.

Option	Description
asynchronous	Determines whether XMLHttpRequest is used asynchronously or not. Default value is true.
contentType	The Content-Type header for a request. Default value is 'application/x-www-form-urlencoded'.
encoding	The encoding for a request. Default value is UTF-8.
method	The HTTP method to use for the request. Default value is post.
parameters	The parameters for the request. These can be provided either as a URL-encoded string or as any Hash-compatible object with properties representing parameters. Default value is ' '.
postBody	Specific contents for the request body on a 'post' method. Default value is None.

Option	Description
requestHeaders	Request headers can be passed under two forms: 1. As an object, with properties representing headers. 2. As an array, with even-index (0, 2...) elements being header names, and odd-index (1, 3...) elements being values. Prototype automatically provides a set of default headers, that this option can override and augment. These are as follows: ☐ X-Requested-With is set to 'XMLHttpRequest'. ☐ X-Prototype-Version provides Prototype's current version, for example, 1.5.0. ☐ Accept defaults to 'text/javascript, text/html, application/xml, text/xml, */*'. ☐ Content-type is set based on the contentType and encoding options.
evalJS	Automatically evals the content of Ajax.Response#responseText if the content-type returned by the server is one of the following: ☐ application/ecmascript, application/javascript, application/x-ecmascript, application/x-javascript, text/ecmascript, text/javascript, text/x-ecmascript, or text/x-javascript and the request obeys SOP (Simple Origin Policy). Default value is true. If evaluation need to be forced, pass 'force'. To prevent it, pass false.
evalJSON	Automatically evals the content of Ajax.Response#responseText and populates Ajax.Response#responseJSON with it if the content-type returned by the server is set to application/json. If evaluation needs to be forced, pass 'force'. To prevent it altogether, pass false. Default value is true.
sanitizeJSON	Sanitizes the content of Ajax.Response#responseText before evaluating it. The setting is false for local requests, true otherwise.

Table 3.2: Callback Options in Prototype's 'Ajax'

Following are the steps to use the Prototype framework:

1. Download the prototype JavaScript from https://ajax.googleapis.com/ajax/libs/prototype/1.7.2.0/prototype.js. The script may need to be copied and manually saved into a new text file named prototype.js.
2. Create a new Web application and place the prototype.js file in the Web Pages folder.

3. Create a new JSP page named index.jsp. The Web application structure is as shown in figure 3.1.

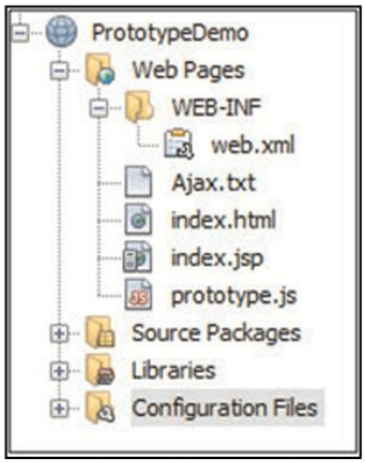


Figure 3.1: PrototypeDemo Folder Structure

4. For using the prototype.js include the following code in the JSP page:

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

Consider a Web page where on click of a button, the data from a file placed on the server is retrieved using Ajax.Request object.

Code Snippet 1 demonstrates the code of index.jsp file using the Ajax.Request object.

Code Snippet 1:

```
<%@page language="java" contentType="text/html;
charset=UTF-8"%>
<html>
<head>
<title>Prototype Demo</title>
   <script type="text/javascript" src="prototype.js">
   </script>
<script type="text/javascript">
function SubmitRequest() {
    new Ajax.Request('Ajax.txt', {
       method: 'get',
      onSuccess: successFunc,
        onFailure:  failureFunc
    });
}
function successFunc(response) {
       debugger;
```

```
         if (200 == response.status) {
        //alert("Call successful");

        var container = $('notice');
        var content = response.responseText;
        container.update(content);
    }
}
function failureFunc(response) {
      alert("Call failed");
}
</script>
</head>
<body>
<p>Click the Update button to view the result.</p>
<br />
<div id="notice">Text from file will be displayed here</div>
<br />
<br />
<input      type="button"      value="Update"      onclick="
SubmitRequest();"/>
</body>
</html>
```

The Ajax.Request Object accesses the file and the contents of the Web page are updated.

Following are the options used in the code:

- method: Specifies the use of HTTP GET or POST method.
- onSuccess: Specifies the callback function on receiving response without error.
- onFailure: Specifies the callback function on error.

Figure 3.2 shows the contents of the Ajax.txt file.

Figure 3.2: Ajax.txt File

The output for the index.jsp page on executing the project is as shown in figure 3.3.

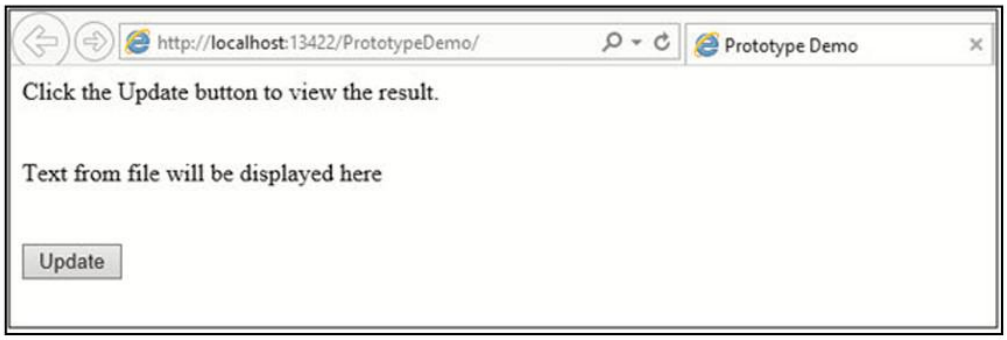


Figure 3.3: Output on Program Execution

On click of the Update button, the text in the tag changes and the file contents are brought asynchronously to be displayed as shown in figure 3.4.

Figure 3.4: Output on Click of Update Button

The Ajax.Request object is created using which, the data from the file, Ajax.txt is brought asynchronously and the content of tag is updated.

- **Ajax.Updater**

This class performs an AJAX request and updates the container's contents based on the contents of the response.

Ajax.Request is the base class of Ajax.Updater and is built for a common use-case.

Syntax:

```
new Ajax.Updater(container, URL [, options]);
```

The Ajax.Updater object can use all the options which are common and has its own specific options as follows:

- **evalScript**: Determines if the response text in the <script> tag is evaluated or not. Default value for evalScript is false.
- **insertion(string)**: By default, the whole content of a container is updated. However, the response can be inserted without deleting the existing content in the container. Default value is None.

Code Snippet 2 demonstrates the use of Ajax.Updater.

Code Snippet 2:

```
new Ajax.Updater('notice','Ajax.cgi', {
        method: 'get',
        insertion: Insertion.Bottom
        });
```

- **Ajax.PeriodicalUpdater**

This class performs an AJAX request periodically and updates the container's contents based on the contents of the response.

Containers are specified similar to the Ajax.Updater class by giving IDs of the HTML elements.

Syntax:

```
new Ajax.PeriodicalUpdater(container, url[, options]);
```

There are two options more to this class with the common options for callback and they are as follows:

- **frequency**: The interval at which AJAX requests are made. The default value is 2.
- **decay**: The value that controls the rate of AJAX request intervals which grows when response does not change. The default value is 1.

- **Ajax.Response**

It is a wrapper class around XMLHttpRequest which deals with HTTP response of AJAX requests.

In the callbacks of AJAX requests, an instance of Ajax.Response is passed. There is no need to create instances of Ajax.Response. It fixes cross-browser issues while including support for JSON through the responseJSON and headerJSON properties.

The methods of Ajax.Response object are listed in table 3.3.

Method	Return Type	Description
getHeaderName()	Null or string	If present, the value of requested header is returned else, is null.
getAllHeaders()	Null or string	It returns string of all headers separated by a line break.
getReponseHeader(name)	String	It returns the value of requested header and throws an error if the header not present.
getAllReponseHeader()	String	It returns string of all headers separated by a line break or throws an error if the header is not present.

Table 3.3: Methods of Ajax.Response Object

The properties of Ajax.Response object are listed in the table 3.4.

Properties	Type	Description
status	Number	It is the HTTP status code returned from the server.
statusText	String	It is the HTTP status text returned from the server.
readyState	Number	It is the current state of the request. 0 – Uninitialized 1 – Loading 2 – Loaded 3 – Interactive 4 – Complete
reponseText	String	It is the text in the response.
reponseXML	Document object or null	It is an XML body of the response if the content-type is application/xml or null.
reponseJSON	Array, Object or null	It is a JSON body of the response if the content-type is application/json or null.
headerJSON	Array, Object or null	It is content of X-JSON header which is auto-evaluated if present or null.
Request	Object	It is the request object itself.
Transport	Object	It is the XMLHttpRequest object itself.

Table 3.4: Properties of Ajax.Response Object

3.3.2 Event Handling

For achieving cross-browser scripting, the main challenge is event management. The key strokes are handled differently by different browsers.

All the cross-browser compatibility issues are handled by the Prototype framework which manages to deal with problems of event management.

Event namespace in the Prototype framework produces information which is requested across all major browsers.

The standardized list of key codes which can be used for keyboard related events are listed in table 3.5.

Key Constant	Description
KEY_BACKSPACE	Represents back space key
KEY_TAB	Represents tab key

Key Constant	Description
KEY_RETURN	Represents return key
KEY_ESC	Represents esc key
KEY_LEFT	Represents left key
KEY_UP	Represents up key
KEY_RIGHT	Represents right key
KEY_DOWN	Represents down key
KEY_DELETE	Represents delete key
KEY_HOME	Represents home key
KEY_END	Represents end key
KEY_PAGEUP	Represents page up key
KEY_PAGEDOWN	Represents page down key

Table 3.5: Key Codes

3.4 script.aculo.us

script.aculo.us is a JavaScript library built on Prototype JavaScript Library to provide the enhanced user interface of Web sites, dynamic visual effects, and utilities.

3.4.1 Ajax Objects

The AJAX controls used in the script.aculo.us library are as follows:

- **Ajax.InPlaceEditor()**

  Non-editable content, such as a <p>, <h1>, and so on can be edited by simply clicking the content.

  The static elements are converted to editable ones and also a cancel and submit button appears for the change to be rolled back or committed respectively.

  Syntax:

  ```
  new Ajax.InPlaceEditor(element, url [ , options ] )
  ```

  The element parameter specifies the associated tag or id of the target element. The next parameter is URL of the server-side script which is to be executed when the text is edited. The third parameter is the usual options list.

  Following is the setup needed to use the script.aculo.us framework:

  1. Download the latest version of scriptaculous.jar from http://script.aculo.us/downloads.
  2. Place the scriptaculous.jar file in the Web Pages folder.

3. Include the script.aculo.us library in the Web page using the following code:

```
<script type="text/javascript"
   src="scriptaculous-js-1.9.0/lib/prototype.js"/>
<script  type="text/javascript"  src="scriptaculous-js-1.9.0/
src/scriptaculous.js?load=effects,controls"/>
```

The effects and controls scripts will be loaded on the page.

Consider a Web page which gets updated with the contents based on updates in the transform.php file.

Code Snippet 3 demonstrates the use of Ajax.InPlaceEditor(). The file name is index.jsp.

Code Snippet 3:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<html>
<head>
<title>Simple AJAX Auto-completer Example</title>
<script type="text/javascript"
   src="scriptaculous-js-1.9.0/lib/prototype.js"></script>
<script  type="text/javascript"  src="scriptaculous-js-1.9.0/
src/scriptaculous.js?load=effects,controls"></script>
<script type="text/javascript">
    window.onload = function () {
        new Ajax.InPlaceEditor('myElement',
      'transform.php',
      {
          okText: 'OK',
          cancelText: 'Cancel'
      }
   );
    }
</script>
</head>
<body>
<p>Click over the text, change the text, and click OK.</p>
<div id="myElement">
      Click me to change.
 </div>
</body>
</html>
```

On clicking the div tag, the tag gets modified to an editor wherein text can be changed as shown in figure 3.5.

Figure 3.5: InPlaceEditor

Specify the text and click OK. The php script changes the text to upper case and displays the modified text.

The transform.php script is as follows:

```
<?php
  if( isset($_REQUEST["value"]) ){
     $str = $_REQUEST["value"];
     $str = strtoupper($str);
     echo "$str";
  }
?>
```

- **Ajax.AutoCompleter()**

Ajax.Autocompleter allows autocompleting text fields which is server-powered.

Syntax:

```
new Ajax.Autocompleter(id_of_text_field, id_of_control_to_
populate, url, options);
```

The first parameter is the reference to a text field or element name that is to be populated with selected data.

The second parameter is the reference to a element or element name to be used as a menu of choices by the control.

The URL parameter specifies the server-side resource that will provide the choices.

The last parameter is the usual set of options.

- **Ajax.InPlaceCollectionEditor()**

The contents of the select box are changed on the fly.

Syntax:

```
new Ajax.InPlaceCollectionEditor( element, url, { collection:
[array], [moreOptions] } );
```

The first parameter is the element which supports editing. The second parameter is the URL from where the changed values would be retrieved for the element. The third parameter is options where the array of collection which is to be placed inside the select box should be called.

3.4.2 Effects

The script.aculo.us effects are divided into following two categories:

- Core Effects
- Combination Effects

Following are the core effects in script.aculo.us library:

Effect.Opacity: An element's opacity is changed to a given level by this effect. This element can be used to show or hide elements. The opacity effect start from the current opacity of the element and continues till the 'to' option if defined, by default 'to' option is 1.0.

Effect.Scale: An element is scaled up or down by this effect, possibly on only one axis (vertical or horizontal). This effect can be used to adjust the size of the target element.

Effect.Move: An element can be moved using this effect. Older version is Effect.MoveBy. The element to be moved must be a positioned element for this effect to work across all the browsers. That is, the CSS position rule must be applied to it and the value of the position can be set to either absolute or relative.

Effect.Highlight: To draw attention to the target element, the Highlight effect is used by changing its background color. The background color of the element with no option specified will change to yellow and then, after the effect duration ends, changes back to the original background color.

Effect.Morph: In this effect, the CSS properties of an element are changed. It takes a set of CSS properties and slowly converts the elements style values to the targets. The option taken by this effect is named as style.

Effect.Parallel: A special effect where more than one core effect can be combined to produce a new effect is known as parallel effect. It takes an array of sub-effects as parameter.

Consider a Web page where on clicking the page the background color and the other properties get changed.

Code Snippet 4 shows the morphing effect with the Morphing.jsp file.

Code Snippet 4:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
<head>
<title>Morphing examples</title>
<script type="text/javascript" src="scriptaculous-js-1.9.0/lib/
prototype.js"></script>
<script type="text/javascript" src="scriptaculous-js-1.9.0/src/
scriptaculous.js?load=effects,controls"></script>
<script type="text/javascript">
function Morphing(element){
    new Effect.Morph(element,
    {style:'background:#f00; color:#fff;'+
            'border: 10px solid #f88; font-size:1em'
        });
    }
</script>

</head>
<body>
<div onclick="Morphing(this)">
    Click here to see the result of Morphing effect.
</div>
</body>
</html>
```

On click of the div tag the Morphing function is called. In the function the CSS properties such as background, color, border, and so on are changed of the same div as this is passed as the parameter.

The output of the Morphing.jsp file is as shown in figure 3.6.

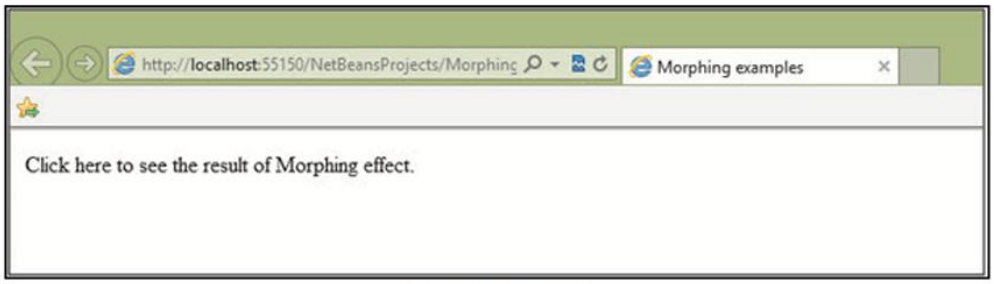


Figure 3.6: Output

On clicking the text, the text CSS properties changes and the output is as shown in figure 3.7.

Session 03

AJAX Client Frameworks-I

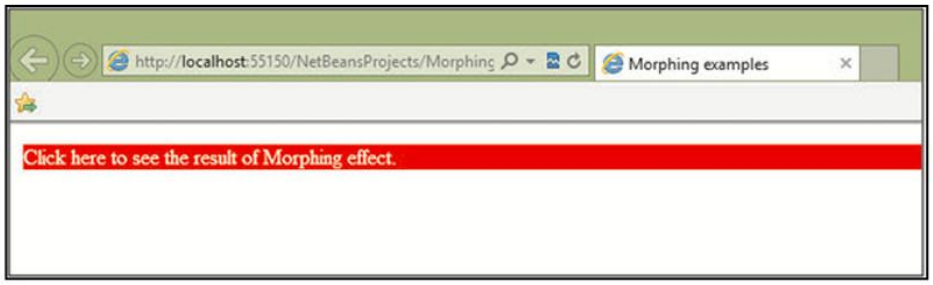


Figure 3.7: Output

All the combination effects are based on the five Core Effects, and are considered as examples for allowing developers to write thier own effects.

Generally, these effects depend on the parallel, synchronized execution of other effects. Since such an execution is readily available, creating customized combined effects is very easy. Here is a list of Combination Effects:

- Effect.Appear
- Effect.Fade
- Effect.Puff
- Effect.DropOut
- Effect.Shake
- Effect.SwitchOff
- Effect.BlindDown
- Effect.BlindUp
- Effect.SlideDown
- Effect.SlideUp
- Effect.Pulsate
- Effect.Squish
- Effect.Fold
- Effect.Grow
- Effect.Shrink

In addition to these, the Effect.toggle utility method is available for elements that need to be shown temporarily with an Appear/Fade, Slide or Blind animation.

3.5 Dojo

Dojo is an open source JavaScript toolkit. Alex Russell, Dylan Schiemann, David Schontzler, and others started working on Dojo in the year 2004. Later, the open source community joined to improve it.

Dojo can be used to add rich look and feel to Web pages. Additionally, asynchronous communication can be enabled using the built-in libraries of Dojo.

3.5.1 Architecture of Dojo

Dojo provides a set of standard libraries that are arranged in layers, one above the other.

The Language Utilities is the first layer that improves and simplifies the coding technique of JavaScript developers while developing Web sites.

The Dojo Event System is the second layer that contains language libraries to process Web application events. These events allow widgets to interact with each other.

The Packaging System is the third and the last layer in the hierarchy. This layer helps developer to customize distribution of Dojo and develop functionalities using modules, resources, and widget namespaces. Dojo code is divided into logical units called modules. Dojo modules are defined in JavaScript files. The JavaScript files are called resources.

A module is usually defined in a single JavaScript file, but sometimes a module definition is split into multiple JavaScript files. Widgets are combined into groups called namespaces. New namespaces can be created and put custom widgets in these namespaces.

Figure 3.8 shows the architecture of Dojo.

Figure 3.8: Architecture of Dojo

3.5.2 Dojo Programming Model

Dojo comprises two programming models, namely Declarative model and Programmatic model.

Code Snippet 5 shows the creation of a Button widget declaratively and programmatically.

Code Snippet 5:

```
<!-- Creating a Button widget declaratively -->
<button data-dojo-type="dijit.form.Button" id="btnExit">
Exit
</button>
<!-- Creating a Button widget programmatically -->
var btnExit = new dijit.form.Button('Button',((label : 'Exit')),
dijit.byId("btnExit"));
```

In declarative model, the Button widget is instantiated declaratively using the tags such as button. The data-dojo-type attribute instructs Dojo to render a button on the Web page. The id attribute assigns a unique identifier to the widget.

In Programmatic model, the widget class, style, and id are passed as parameters to the constructor. The btnExit variable will refer to the instantiated widget. The first parameter refers to the Button class that is used by Dijit to initialize the Button widget's properties. The second parameter shows the label of the button. The third parameter uses the dijit.byId() function to refer the Exit button by its id name.

3.5.3 Setting Up Dojo

To use Dojo toolkit in Web applications, the first step is to download the latest copy of the Dojo toolkit from http://dojotoolkit.org/download/ and copy the libraries to the Web application's root folder. The Dojo toolkit comprises the modules dojo, digit, dojox, and util.

Figure 3.9 shows the Dojo toolkit added to the Web Pages folder.

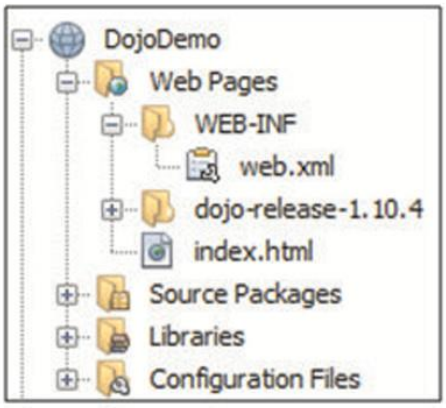


Figure 3.9: Modules in Dojo toolkit

Code Snippet 6 shows the steps to set up and load Dojo in a Web page.

Code Snippet 6:

```
…
<!-- Step 1 -->
<style type="text/css">
      @import "./dojo/resources/dojo.css";
      @import "./dijit/themes/tundra/tundra.css";
</style>
OR
<link  rel="stylesheet"  href="./dojo-release-1.10.4/dijit/themes/
tundra/tundra.css"/>

<!-- Step 2 -->
<script>dojoConfig = {parseOnLoad: true};</script>
      <script src="./dojo-release-1.10.4/dojo/dojo.js" type="text/
javascript"></script>
        <script>
            require(["dijit/form/Button"]);
        </script>
…
```

The first step imports the cascading style sheet file named tundra.css by using the @ import directive or link tag. The tundra.css applies the Tundra theme to the widgets. The other themes can also be applied by importing the respective .css file.

The second step loads the dojo.js file. This file contains the base Dojo script for searching or manipulating DOM, binding events, sending AJAX requests, and applying basic effects. This file is located in the dojo subdirectory.

3.5.4 Creating and Connecting a Button Widget to an Event

The Dojo Widget library, also known as Dijit, contains several widgets for designing Web pages. The most commonly used widget is a button. Code Snippet 7 shows the code to create a Button widget with the caption Hello Dojo!.

Code Snippet 7:

```
<html>
    <head>
        <title>TODO supply a title</title>
<html>
    <head>
        <title>TODO supply a title</title>
      <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <link rel="stylesheet" href="./dojo-release-1.10.4/dijit/
themes/tundra/tundra.css"/>
```

```
      <script>dojoConfig = {parseOnLoad: true};</script>
     <script src="./dojo-release-1.10.4/dojo/dojo.js" type="text/
javascript"></script>
        <script>
            require(["dijit/form/Button"]);
        </script>
    </head>
    <body class="tundra">
    <button data-dojo-type="dijit/form/Button" type="button">Hello
Dojo!
        <script type="dojo/on" data-dojo-event="click" data-dojo-
args="evt">
         alert('Welcome to the world of Dojo!!!');
        </script>
        </button>
    </body>
</html>
```

To create a Button widget, first load the Button widget module by using the require() function. In the body section of the HTML document, create a Button widget by using the button tag. The data-dojo-type attribute instructs Dojo about the type of the widget to be displayed.

Events in JavaScript or Dojo-based applications enable to add interactivity to Web pages.

In Dojo, an event handler can be connected to a widget through a script tag. Note that the script tag is written within the button tag. Additionally, the type attribute of the script tag uses dojo/on to indicate an event handler. This is bound to the click event using the data-dojo-event attribute. Therefore, when the button is clicked, an alert message is displayed on the page.

The output of the file on execution is as shown in figure 3.10.

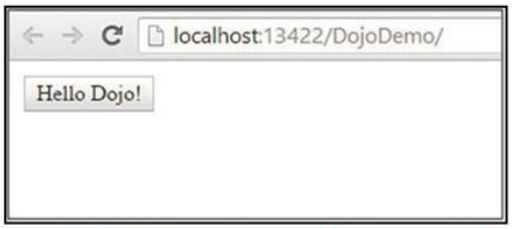


Figure 3.10: Code and output for button Widget

Figure 3.11 shows the output after event is handled for the Button widget.

Session 03

AJAX Client Frameworks-I

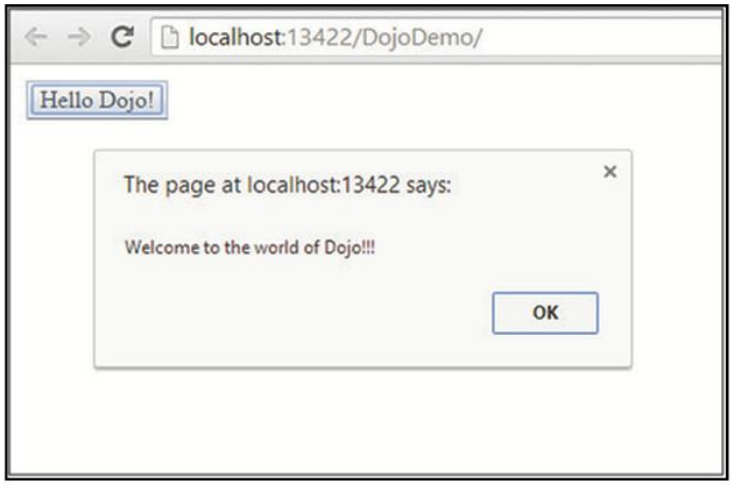


Figure 3.11: Event Handling for Button Widget

3.6 Introduction to Dijit

Dijit is an acronym for Dojo widget. Dijit is a widget system that is placed on top of Dojo. Use Digit to design rich Graphic User Interfaces (GUIs) by using minimal code. Dijit can either be used declaratively by using special attributes within regular HTML tags or programmatically by using JavaScript.

Based on the context, one can use the term Dijit for a single Dojo widget or the term Dijits for all the widgets in the toolkit. Some of the widget libraries available include CheckBox, RadioButton, ComboBox, TextBox, TextArea, ValidationTextBox, and so on.

3.6.1 Dijit Layout

Dijit has multiple layout widgets that are combined together in a hierarchy. Figure 3.12 shows that the screen is broadly split into two parts. The top acts as a toolbar. The bottom is again split into a left section and right section. The left section has three panes, and the right section is split into two parts.

Conceptually, a Dijit is a set of container that contains three types of elements. The first type of elements are those containers that display all their children side by side. The second type of elements are those containers that display one child at a time, and the third type of elements are the leaf nodes that contain only the content.

Figure 3.12 shows the hierarchy of Dijit.

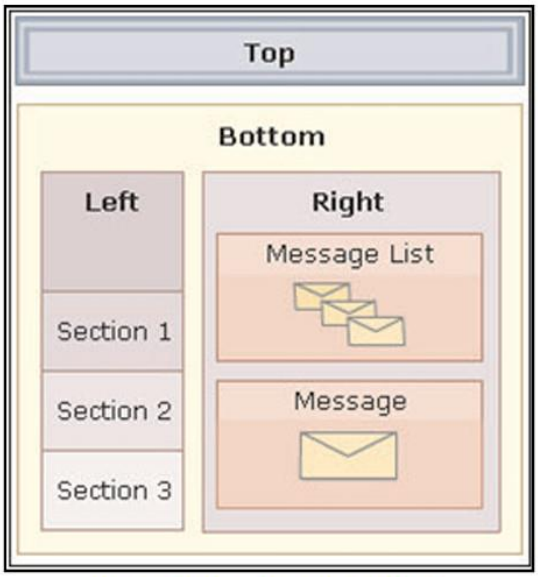


Figure 3.12: Hierarchy of Dijit

3.6.2 Dijit Layout Widgets

Dijit's layout widgets are stored in the dijit.layout subpackage.

Figure 3.13 shows the layout widgets available in the dijit.layout subpackage.

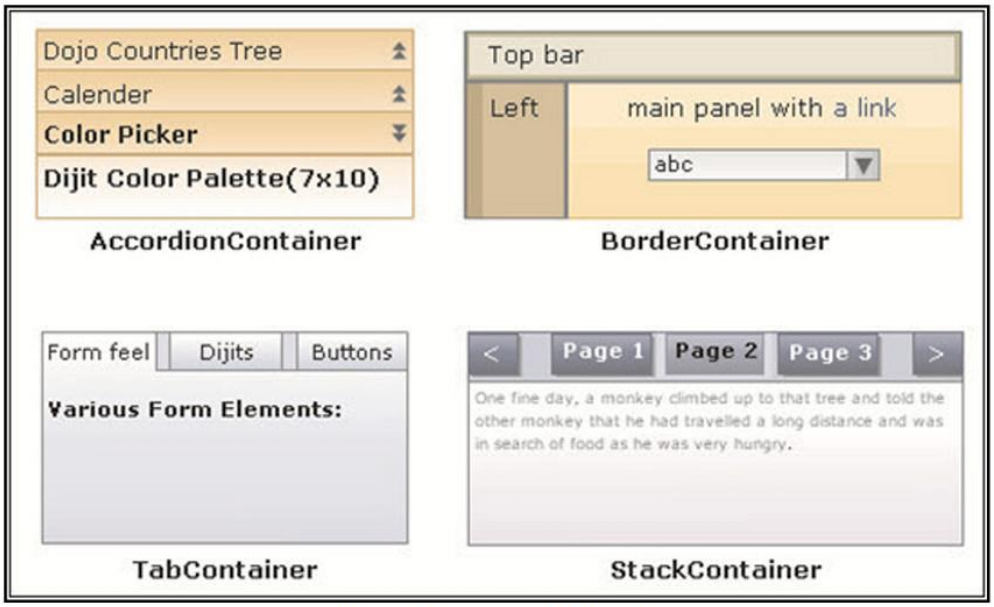


Figure 3.13: Layout Widgets

The different layouts provided by Dijit are as follows:

- **AccordionContainer**

  The AccordionContainer layout displays multiple panes. A pane's title can be clicked to pull up or pull down the panes. Only one complete pane is visible at a time.

- **BorderContainer**

  The BorderContainer layout divides the container into top, left, bottom, right, and center sections. The layout also provides optional splitter controls to allow users to adjust the dimensions. For example, this layout can be used to reserve the top 100 pixels of the screen for title and navigation and the rest for the displaying some other content.

- **ContentPane**

  The ContentPane layout resembles an internal frame, but contains additional design features. A ContentPane is the most basic layout container that is placed inside the layout container.

- **LayoutContainer**

  The LayoutContainer arranges the child nodes in top, left, bottom, right, and client sections. The container has a specific size. It places the child nodes along the edges of the different sections. It then places the child marked as client in the remaining space at the centre of the page. For example, this layout easily formats table of contents.

- **SplitContainer**

  The SplitContainer layout splits the children into many sections. The size of each section can be adjusted.

- **StackContainer**

  The StackContainer layout has multiple children, but shows only one child at a time. This container can be used for slide shows, allowing the user to display one pane at a time.

- **TabContainer**

  The TabContainer layout resembles a tabbed folder. To display the content of a particular tab, click the corresponding tab title.

3.6.3 Check Box and Radio Button Dijits

Check boxes are used when you want to allow a user to select zero or more options from a set of options.

Dojo check boxes are similar to HTML check boxes, but the former provides more styling options. To render a selected check box, you can set the value of the checked attribute as checked. The value attribute returns the value of the selected check box.

Radio buttons are used when there is a list of two or more options and you want to allow the users to select only one option from the list of options. Note that you can import the CheckBox and RadioButton classes by using dijit.form.* or by using the require function.

Code Snippet 8 demonstrates creation of check box and radio button using dijit.

Code Snippet 8:

```
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <link rel="stylesheet" href="./dojo-release-1.10.4/dijit/
themes/tundra/tundra.css"/>
        <script>dojoConfig = {parseOnLoad: true};</script>
     <script src="./dojo-release-1.10.4/dojo/dojo.js" type="text/
javascript"></script>
        <script>
      require(["dijit/form/CheckBox", "dijit/form/RadioButton"]);
        </script>
    </head>
    <body class="tundra">
        <h2>Check Box</h2>
            <input data-dojo-type="dijit/form/CheckBox" id="cb1"
type="checkbox" name="Designer" checked="checked"/>
        <label for="cb1">Are you a Web Designer? </label>
        <input data-dojo-type="dijit/form/CheckBox" id="cb2"
type="checkbox" name="Programmer" checked="checked"/>
        <label for="cb2">Are you a Programmer? </label>
          <br/>
        <h2>Radio Button</h2>
         <input data-dojo-type="dijit/form/RadioButton" id="rb1"
type="radio" name="group1"/>
        <label for="rb1">Are you a Web Designer? </label>
         <input data-dojo-type="dijit/form/RadioButton" id="rb2"
type="radio" name="group1" checked="checked"/>
       <label for="rb2">Are you a Programmer? </label>
    </body>
</html>
```

The name attribute defines a common name, group1, for all the radio buttons. The label tag displays a label for the radio button.

Figure 3.14 shows the creation of check box and radio button using dijit.

Figure 3.14: Check Box and Radio Button Widgets

3.6.4 AutoCompleter Combo Box Dijit

Dojo combo box is a combination of a drop-down list and a single-line text box. A user can display the list by clicking the drop-down arrow. As the user moves the pointer over the list, each option under the pointer is highlighted. If the user selects an option from the list, the current selection is replaced with the selected option.

Create a combo box widget by using the input tag in the code. The data-dojo-type attribute uses the value digit/form/ComboBox.

The autocomplete attribute is set to false, which forces the user to write entire text in the combo box to confirm its availability in the options provided.

Code Snippet 9 shows the creation of AutoCompleter combo box using dijit in the file Autocompleter.jsp.

Code Snippet 9:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Auto Completer</title>
         <link rel="stylesheet" href="./dojo-release-1.10.4/dijit/
themes/tundra/tundra.css"/>
     <script>dojoConfig = {parseOnLoad: true};</script>
      <script src="./dojo-release-1.10.4/dojo/dojo.js" type="text/
javascript"></script>
    <script type="text/javascript" src="/dojo-release-1.10.4/dojo/
dojo.js">
```

```
            require(["dijit/form/ComboBox"]);
        </script>
    </head>
    <body class="tundra">
        <!-- <h2>Auto Completer</h2>
            <div  data-dojo-type="dojo/store/Memory"  data-dojo-
id="stateStore"
            data-dojo-props="data: [{id: '1', name: 'Alex'}, {id:
'2', name: 'Tony'}, {id: '3', name: 'Alice'}]"></div>
        <input data-dojo-type="dijit/form/ComboBox" value="Alex"
data-dojo-props="store:stateStore, searchAttr:'name'"
            name="state" id="stateInput" autocomplete="false" />
    </body>
</html>
```

The output for the Autocompleter.jsp page is as shown in figure 3.15.

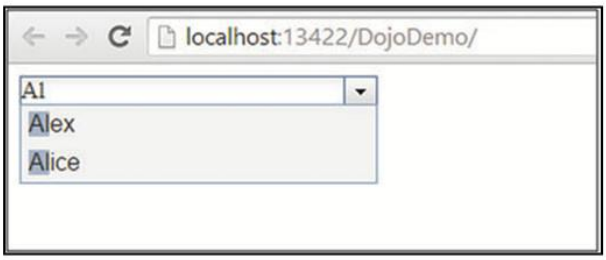


Figure 3.15: AutoCompleter Combo Box Using dijit

3.6.5 Using dojo.xhrGet() Function

Dojo provides a function named dojo.xhrGet() to send and receive data asynchronously. Code Snippet 10 demonstrates the code to send an AJAX request using Dojo's dojo.xhrGet() method.

Code Snippet 10:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
        <title>TODO supply a title</title>
        <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <link rel="stylesheet" href="./dojo-release-1.10.4/dijit/
themes/tundra/tundra.css"/>
        <script>dojoConfig = {parseOnLoad: true};</script>
```

Session 03

AJAX Client Frameworks-I

```
      <script src="./dojo-release-1.10.4/dojo/dojo.js" type="text/
javascript"></script>
     <script>
            require(["dijit/form/Button"]);
       </script>
     <script type="text/javascript">
            function FileReading() {
                dojo.xhrGet({
                    url: "Demo_test.txt",
                    handleAs: "text",
                    load: function(response, ioArgs) {
                       dojo.byId("replace").innerHTML = response;
                        return response;
                    },
                    error: function(response, ioArgs) {
           console.error("HTTP status code: ", ioArgs.xhr.status);
            dojo.byId("replace").innerHTML = 'Loading the resource
from the server failed';
                        return response;
                   }
                });
            }
        </script>
    </head>
    <body class="tundra">
        <h2>Using Dojo with AJAX</h2>
      <button data-dojo-type="dijit/form/Button" type="button">Read
File
<script type="dojo/on" data-dojo-event="click" data-dojo-args="evt">
                FileReading();
            </script>
        </button>
        <br/><br/>
        <div id="replace"> Hello!!! </div>
    </body>
</html>
```

Note that the script tag is enclosed within the button tag and bound to the click event. This ensures that the script is executed on the click of the Dojo button.

Following are the options used with the method:

- **url**

  The url attribute specifies the name of the server-side component such as a JSP, servlet, or a file that will process the AJAX request. Here, the file Demo_test.txt on the server is used for the data to be displayed.

- **handleAs**

  The handleAs attribute specifies the MIME type such as text, json, javascript, and xml. Here, text is used.

- **load**

  The load attribute specifies the callback function that will be executed after response is received successfully.

- **error**

  The error attribute specifies the callback function that will be executed in case an error is encountered while processing the request.

- **response**

  The response is the data from the server. The div is accessed using the by Id() method and the response is displayed into it.

The content of the Demo_test.txt file is as shown in figure 3.16.

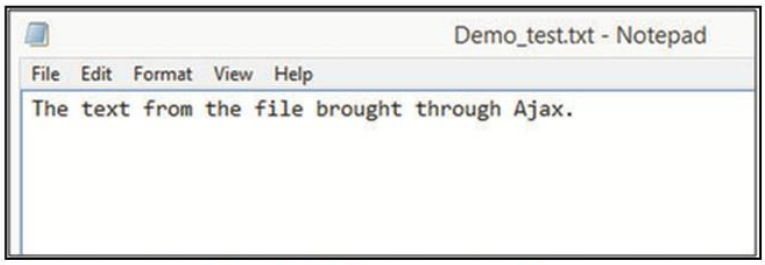


Figure 3.16: Demo_test.txt file

The output for Code Snippet 10 is as shown in figure 3.17.

Figure 3.17: Output on Page Load

The div displays 'Hello!!!' as per the initial text given in the code. To load the file contents asynchronously, click Read File. The contents are loaded from the file asynchronously and displayed in the div as shown in figure 3.18.

Figure 3.18: Output on Clicking the Button

- **Defining the load and error Functions**

Code Snippet 11 demonstrates the use of the functions loginCallback() and loginError().

Code Snippet 11:

```
...
function loginCallback(data, ioArgs){
    alert(data.getElementByTagName("name")[0].childNodes[0].
nodeValue);
}
function loginError(data, ioArgs){
  alert('Error when retrieving data from the server!');
}
```

Note that both the functions accept two parameters, namely data and ioArgs. The data parameter holds the data returned by the server-side component such as DataServlet. The ioArgs parameter holds the request parameters sent using xhrGet() function.

The loginCallback() function is used to process the response received from the server. The

DataServlet returns an XML response containing the request parameter's value enclosed in a name element. Therefore, the code retrieves the element named name using the getElementsByTagName() function. Next, the value of the element is retrieved using the nodeValue property. The value is then displayed in an alert dialog box.

The loginError() function is used to display an error message if AJAX request could not be processed.

- **Working with Server-Side Component**

After the request is sent to the server, the request is processed using a server-side component such as JSP page or a servlet.

Server-Side Code

Code Snippet 12 demonstrates the doGet() method of a servlet named DataServlet. This method will process the AJAX request and send an XML response back to the client.

Code Snippet 12:

```
protected void processRequest(HttpServletRequest request,
HttpServletResponse response) throws ServletException,
IOException {
  response.setContentType("text/html;charset=UTF-8");
try (PrintWriter out = response.getWriter()) {
  String param = request.getParameter("nameParam");
  if(param!=null){
    out.write("<name>" + param + "</name>");
  }else{
    out.write("Error!!!") ;
  }
  out.close();
 }catch(Exception e){
   out.println(e.getMessage());
 }
}
```

First, the content type of the response is set to text/xml indicating that the response will contain XML data. Next, an instance of PrintWriter is created to write data to the response. Then, the value of request parameter named nameParam is retrieved and stored in a String variable, param. Finally, the XML data to enclose the value of variable, param in the name element is written.

Figure 3.19 shows the implementation of the doGet() method at the server-side.

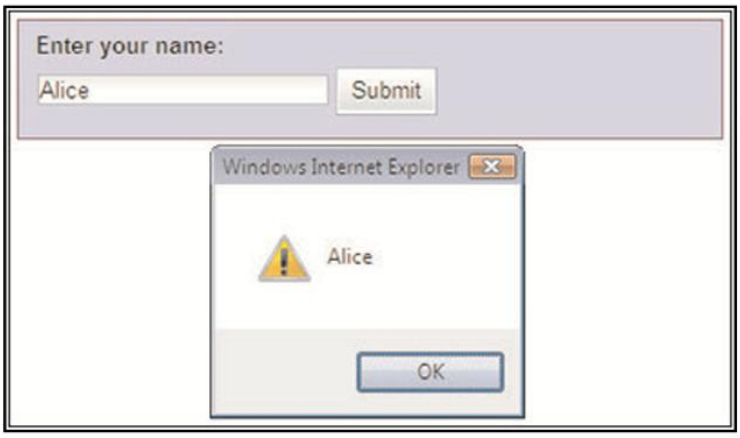


Figure 3.19: Output on Clicking the Submit Button

Session 03

AJAX Client Frameworks-I

Check Your Progress

1. Which of the following statements about Dojo are true?

(a)	Dojo is an open source JavaScript toolkit designed to develop AJAX-based applications.
(b)	Dojo widgets are a combination of only HTML and CSS style declarations.
(c)	Dojo allows storage of data on the client and the server using server-side data store implementations.
(d)	The Web page author need not learn additional programming language to use the Dojo widgets.
(e)	The Dojo code created for one application can be used in another application.

(A)	a, b, c	(C)	b, c, e
(B)	c, d, e	(D)	a, c, d

2. Which of the following statements about Dojo architecture and its working are false?

(a)	Dojo provides a set of three layered libraries namely, the packaging system, the event system and the language utilities layer.
(b)	Dojo fails to overcome compatibility issues across major browsers.
(c)	Dojo's programming model allows you to create widgets using tags.
(d)	The Dojo programming model can be used only declaratively.
(e)	The Dojo code is divided into logical units called modules.

(A)	a, b, c	(C)	b, c, e
(B)	b, d	(D)	a, c

3. Which option is used to append the text in the container using Prototype framework and Ajax.Updater object?

(A)	Insertion	(C)	Append
(B)	Insert	(D)	Join

Session 03

AJAX Client Frameworks-I

Check Your Progress

4. Which is the special effect in script.aculo.us where two or more effects are combined?

(A)	Opacity	(C)	Parallel
(B)	Scale	(D)	Morph

5. Which of the following Code Snippet will create a Dojo button widget with the caption 'Close'?

(A)	… <script> require(["dijit/form/Button"]); </script> … <button data-dojo-type="dijit/form/Button" type="button">Close </button>
(B)	<script type="text/javascript" src="./dojo/dojo.js"> dojo. provide("dijit/form/Button"); </script> ... <body class="tundra" > <button data-dojo-type="dijit.form.Button"> Close</button> </body>
(C)	<script type="text/javascript" src="./dojo/dojo.js"> import("dijit/form/Button"); </script> ... <body class="tundra"> <button data-dojo-type="dojo/form/Button">Close</button> </body>
(D)	<script type="text/javascript" src="./dojo/dojo.js"> </ script> ... <body class="tundra"> <button dojoType="button">Close</button> </body>

Session 03

AJAX Client Frameworks-I

Answers

1.	D
2.	B
3.	A
4.	C
5.	A

Session
03

AJAX Client Frameworks-I

Summary

- Prototype is a JavaScript Framework which helps to easily develop dynamic Web applications.
- The global 'Ajax' object contains the AJAX functionality.
- script.aculo.us is a JavaScript library built on Prototype JavaScript Library to provide the enhanced user interface of Web sites, dynamic visual effects, and utilities.
- The core effects in script.aculo.us are opacity, highlight, morph, move, parallel, and scale.
- Dojo is an open source JavaScript toolkit for developing AJAX-based applications. The benefits of using Dojo toolkit include code simplification and reusability of code.
- Dojo Widget Library (Dijit) is a widget system that enables quick and easy development of Web pages.
- Some of the widgets available in Dijit are CheckBox, RadioButton, ComboBox, TextBox, DialogBox, and so on.

04 AJAX Client Frameworks–II

Welcome to the Session, **AJAX Client Frameworks–II**.

This session explains about different types of AJAX client frameworks. It describes the features of jMaki, jQuery, and Yahoo User Interface (YUI) frameworks with respect to AJAX.

In this Session, you will learn to:

- Explain jMaki and its widgets
- Explain jQuery and its features
- Explain YUI and its features

Session 04

AJAX Client Frameworks–II

4.1 jMaki

jMaki is an open source, light weight client-server framework. jMaki originated in Kumamoto, Japan. The letter 'j' in jMaki represents JavaScript technology and Maki, which is a Japanese word, means 'to wrap'. In other words, jMaki means JavaScript wrappers.

jMaki is used for creating AJAX applications by integrating JavaScript technology into the applications. jMaki allows including styles and templates, widget model, and client services, such as event handling, in a client application. For server applications, jMaki provides server runtime component and a generic proxy, named XMLHttpProxy. XMLHttpProxy enables the server applications to interact with external Web services outside the application domain. JMaki provides access to widgets from various toolkits as a JSP taglib or as a JSF component.

Figure 4.1 shows the origin of jMaki.

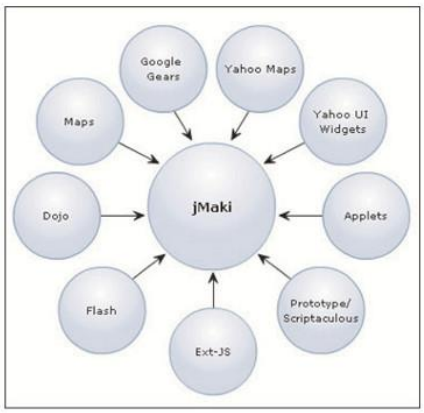


Figure 4.1: Origin of jMaki

Note - A lightweight client-server framework implements most of its functionality as independent modules. The advantages of this framework are that its sections are independently configurable, and it is easy to learn.

Some of the features of jMaki are as follows:

- **Wrapping of AJAX components in Tags**

  jMaki uses JSP tags or JSF components to wrap the AJAX components thus, enabling easy development of applications. For example, to include the clock widget present in Dojo library, in an application, the code to be written is as follows:

```
<a:widget name="dojo.clock" />
```

- **Standardization of JavaScript Toolkit API**

  jMaki provides a common framework for representing, documenting, and working with widgets. This is possible because jMaki wraps all the different APIs in a single widget style. The widgets can be reused from different libraries according to requirements.

Session 04

AJAX Client Frameworks–II

- **Support of Multiple Server Technologies**

  jMaki supports multiple server technologies during runtime such as JSP, JSF, PHP, and JavaScript. jMaki application does not limit itself to any specific server data model and these integrate easily with the existing technology in the server environment.

- **Preference for Convention over Configuration**

  jMaki provides a considerable amount of default data and samples. Since, widgets and APIs are not mutually exclusive, jMaki follows 80/20 rule. Developers spend 20% of their time assembling, configuring the widgets visually, and 80% of their time writing codes to implement the various APIs in AJAX application.

- **Provision of Standardized Event/Data Model**

  jMaki describes tree, table, and menu structures using JavaScript Object Notation (JSON) format. The consistent programming model helps in standardization of data and event model by jMaki. This enables the widgets from various toolkits to work with the same set of data.

Figure 4.2 shows the widgets provided by jMaki.

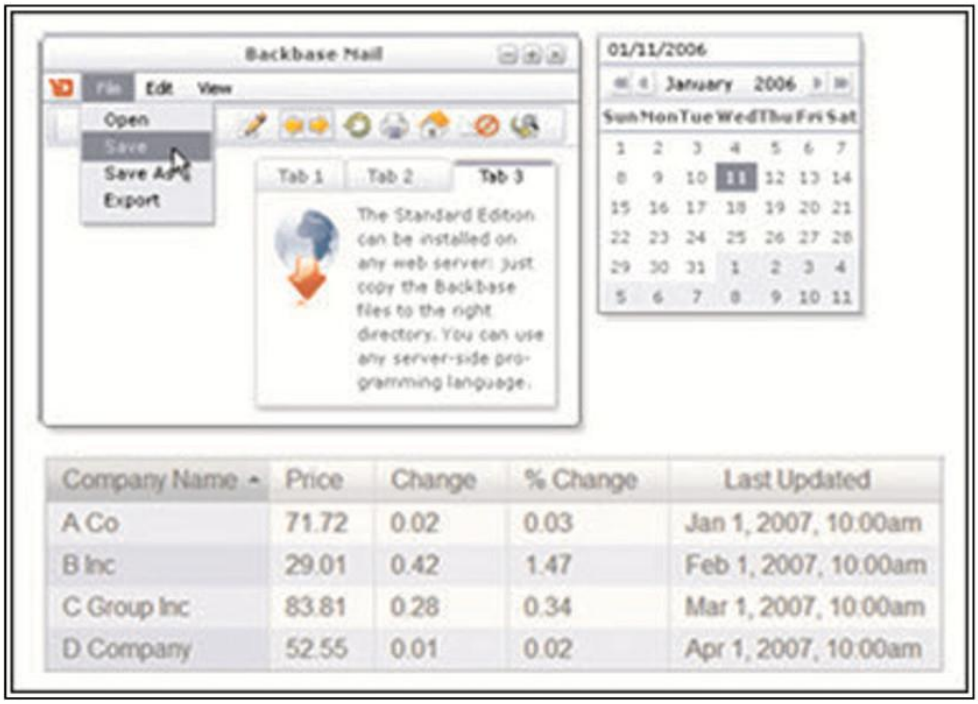


Figure 4.2: Widgets Provided by jMaki

4.1.1 Client–Side Components

Architecture is a framework within which a system is built. It defines the components that constitute a system, and the information exchanged between the components.

Session 04

AJAX Client Frameworks–II

jMaki framework comprises the client components and server components. The client components that make up jMaki Architecture are as follows:

- **jMaki Layouts**

  jMaki provides different layouts to help reduce efforts and time required to create/ design the layout of a Web page. jMaki uses HTML and CSS to create these layouts. These layouts can be easily customized.

- **jMaki Client Runtime**

  jMaki client runtime uses JavaScript. It is responsible for bootstrapping the widgets and passing unique parameters provided by the server-side runtime to the widgets. JavaScript runtime ensures that the correct parameters are passed from the server-side runtime to initialize the widget. However, the runtime assigns default parameters to widgets if no specific parameters are provided.

- **jMaki Client Services**

  jMaki client services provide APIs to use XMLHttpRequest object that allows data transfer between the client and the server. These services also provide publish/ subscribe event handling mechanism to enable communication between the widgets. jMaki Glue is built on top of publish/subscribe mechanism that helps to define the application behavior. Widgets are tied together using the JavaScript actions. jMaki Timers invoke JavaScript action handlers or publishes events at regular intervals.

- **jMaki Widget Model**

  jMaki widget model provides a component model for reusable JavaScript components. The structure of Widgets is based on HTML, CSS, and JavaScript. jMaki stores widget descriptions in widget.json format.

  Figure 4.3 shows the client-side components of jMaki framework.

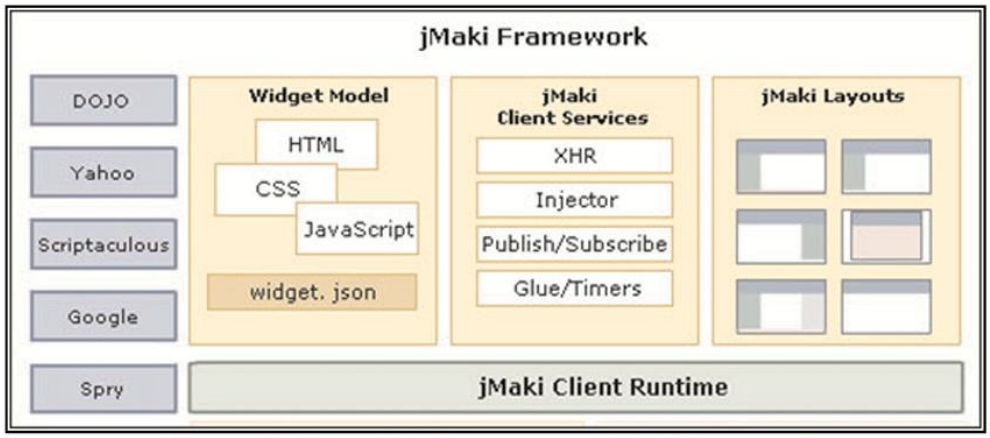


Figure 4.3: Client-Side Components of jMaki Framework

4.1.2 Server–Side Components

The two server components that make up jMaki framework are as follows:

- **jMaki Server Runtime**

  jMaki server runtime binds the server-side runtime with the jMaki client runtime. It is also responsible for tracking and delivering the correct JavaScript, CSS, and HTML references based on the library being used so that these are not duplicated. For ensuring the availability of correct data to a widget instance, serialization of data in JavaScript is performed by server runtime.

- **XMLHttpProxy**

  XMLHttpProxy allows widgets to access JSON or other external services, such as Flickr image searches. Direct Communication takes place between the widgets and the services.

Figure 4.4 shows the server-side components of jMaki framework.

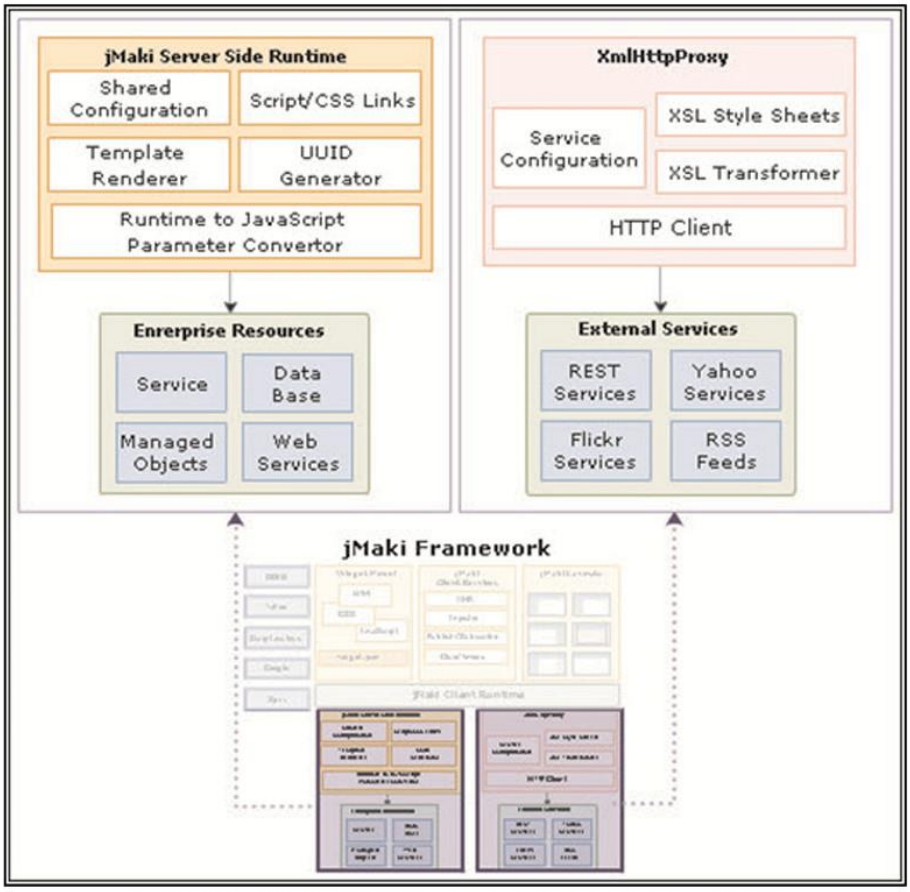


Figure 4.4: Server-Side Components of jMaki Framework

4.1.3 Application Structure

jMaki applications can range from simple applications containing few widgets to complex applications containing multiple jMaki widgets.

Figure 4.5 displays the directory structure of an application.

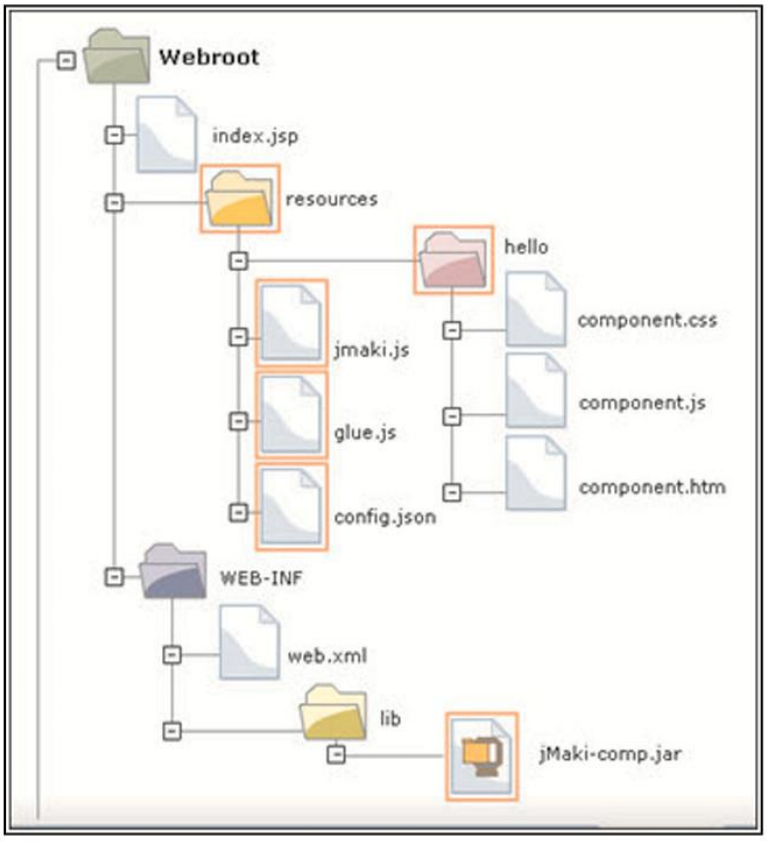


Figure 4.5: Directory Structure of an Application

- **resources**

  The resources directory contains all the resources used by a jMaki application.

- **jmaki.js**

  The jmaki.js is a JavaScript file that contains code for loading jMaki widgets and functions for widget communication. It is present in the resources directory.

- **config.json**

  The config.json file contains theme information, extension mapping information, and glue mapping information for wiring widgets.

- **jmaki-comp.jar**

  The jmaki-comp.jar file that contains the server runtime code is present in the /WEB-INF/lib directory.

- **hello**

  The hello is a widget and its resources are present in the /resources/hello directory. A widget is a directory comprising component.css, component.js, and component. htm file.

- **glue.js**

  The glue.js file glues widgets together. It is used for registering and defining widget event listener, publishing events to a topic and subscribing to a topic.

4.1.4 Widget Model and Properties

A jMaki widget is a reusable parameterized component. jMaki ensures that proper parameters are passed to a widget code to initialize the widget in a page.

The name of a widget maps to a directory. In other words, a jMaki widget is a directory or a package where the widget resides. The directories are separated using 'dot' notation. The directory which makes a widget comprises three core resource files. They are as follows:

- **component.css**

  This file defines the CSS styles for a widget when it is displayed. It contains the code controlling the appearance of the widget. It is optional.

  Code Snippet 1 demonstrates the content of a component.css file.

Code Snippet 1:

```
.header {
height:150px;
border: 1px solid #000000;
}
.main {
position: relative;
width: 100%;
height:auto;
}
.content {
margin: 0 0 0 250px;
height: auto;
border: 1px solid #000000;
}
...
```

The CSS file contains the code defining the appearance of the widget.

- **component.js**

This file defines the behavior of the widget. It contains code for wrapping of widgets, handling of widget events initiated by the user, and interaction with AJAX. It is mandatory to have this file. Code Snippet 2 displays the content of a component.js file.

Code Snippet 2:

```
jmaki.namespace("jmaki.widgets.hello");
jmaki.widgets.hello.Widget = function(wargs) {
//widget code
}
```

In Code Snippet 2, the widget is placed in a jmaki.widgets.hello namespace and is called a widget by appending the term Widget to it. The term Widget represents the constructor to which the widget argument is passed. The jMaki server-side component will look in the same widgets directory for a directory named 'hello' containing the subdirectory, foo. If the directory is found then it will look for component.js and component.htm file under jmaki/widgets/hello. If the directory is not found then the server-side component will look under the resources directory for the widget and its resources.

- **component.htm**

This file defines the default HTML template that will be used by the rendering mechanism to display the widget in the page. In other words, it specifies the page layout for the widget. jMaki ensures that the HTML template is displayed with unique and instance specific parameters. It is mandatory to have this file.

Code Snippet 3 demonstrates the content of component.htm file.

Code Snippet 3:

```
<div id="${uuid}" >
</div>
```

The markup that is included in the page is an instance of the widget that is used on the page. The code displays a template of a simple element with a unique id. The ${uuid} is replaced when jMaki processes the template before the page is displayed.

The design pattern of jMaki helps to create widgets easily. Web applications can configure these widgets. Each widget contains instance parameters. These instance parameters are passed by server-side runtime to JavaScript runtime using a function call. The JavaScript runtime passes the instance parameters, as object literal, to the widget as it is being created. The property values can be specified of a widget by using tag attributes that matches the property name. JavaScript properties that are used with args and value are object literals. These should be enclosed in single quotes or escaped double quotes.

4.1.5 Life-Cycle

Both client and server interactions are needed for displaying jMaki widgets. The sequence for the interactions is as follows:

1. The jMaki widget defined in a JSP page along with the taglibs is interpreted.
2. The jMaki server-side components provide the correct HTML content along with their links to component.css file that is rendered to the page.
3. The jMaki server-side runtime component provides the content from the template file (component. htm) containing unique identifier of the widget to the page.
4. The jMaki bootstrapper script present in the jmaki.js file is rendered first to create a global object named jMaki. The object contains the properties and functions for registering, loading, and supporting jMaki widgets.
5. Once the widget's template has been rendered, addWidget() function of jMaki object creates and registers the widget with the jMaki bootstrapper.
6. When the onload event of the page is fired, the registered widgets are initialized by the jmaki bootstrapper.
7. The rendered page is available for event processing.

Figure 4.6 shows the life-cycle of a jMaki widget.

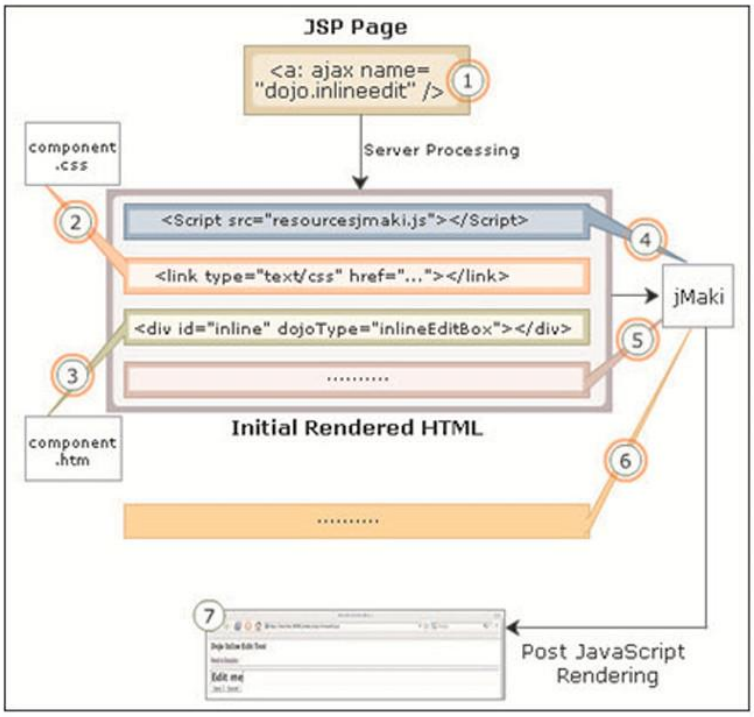


Figure 4.6: Life-Cycle of a jMaki Widget

Session 04

AJAX Client Frameworks–II

4.1.6 Adding Widgets

A jMaki widget is added to a page only after the page with the required template has been created in the application. On adding a widget to a module, the three events that take place are as follows:

- Widget resources such as component.js and component.htm files are added to the application under the resources directory.
- Definition of the jMaki tag library is added to the page.
- Custom jMaki widget tag is added to the page that refers the widgets and sets the widget attributes to default value. The tag represents a JSP handler. It also adds the tag library declaration.

For example, on adding a Dojo table widget, the component.js and component.htm files are added in the resources/dojo directory. Next, the Dojo widget code is added to the resources/lib directory of the application. Finally, the tag library is declared and ajax tag is used to add the widget. Once a widget has been dropped onto a page, the IDE uses the name and value attribute to initialize the widget.

Code Snippet 4 demonstrates adding of the tag library declaration and ajax tag to the page.

Code Snippet 4:

```
<%@ taglib prefix="a" uri="http://jmaki/v1.0/jsp" %>
...
<a:widget name="dojo.table"
args="{columns: { 'title' : 'Title', 'author': 'Author', 'bookId':
'BookID', 'price': 'Price'}}"
value="{rows:[
['JavaScript by Dummies', 'Alex John','A101', '450'],
['Ajax with Java', 'Jean Thomas','A102', '650']
]}" />
```

The name attribute specifies the widget name. Dot notation specifies the directory structure containing the widget's resource files. The args attribute contains the column description whereas, the value attribute contains the values for each column. The structure of the table has been separated from the data. The jMaki widgets accept data in JSON format and the data is provided using the value attribute.

4.1.7 Loading Data

jMaki widgets can be populated with data. There are three ways by which data can be loaded onto a widget. They are as follows:

- Referring to a static file that contains JSON data
- Referring to the data in a bean by using an Expression Language (EL) expression in the tag's value attribute.
- Referring to the data provided by a JSP page or servlet using the widget's service attribute 'All', the data needs to be passed to jMaki widget's in JSON format.

Session 04

AJAX Client Frameworks–II

In other words, data from a bean needs to be converted into JSON format. Data conversion is performed using JSON APIs.

Three steps to be followed for adding data into a widget using EL expression are: creation of a bean class that represent a single object, conversion of the data into JSON format, and loading of the data from the bean into a widget.

- **Creation of a Bean Class**

Code Snippet 5 demonstrates the creation of a Bean class.

Code Snippet 5:

```
public class Book {
private int ID;
    private String bookName;
    private String author;
    private String price;

     public Book(int num, String bname, String auth, String
price){
        ID = num;
        bookName = bname;
        author = auth;
        this.price = price;
    }

    public int getID() {
        return ID;
    }
    public void setID(int ID) {
        this.ID = ID;
   }
...
}
```

The Book class is created with a parameterized constructor to initialize the data members. The getter and setter methods for ID attribute have been added.

- **Data Conversion into JSON Format**

Code Snippet 6 demonstrates the code that converts data into JSON format.

Code Snippet 6:

```
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import org.json.simple.JSONArray;
public class BookApplicationBean {
```

```
    public List addBooks() throws Exception
    {
        ArrayList booklist = new ArrayList();
         Book bookObj = new Book(201, "Who Moved My Cheese",
"Alfred John", "450");
        booklist.add(bookObj);
        return booklist;
    }
…
    public JSONArray displayBookData() throws Exception {
        JSONArray booksArray = new JSONArray();
        JSONArray book = new JSONArray();
        ArrayList bookList = (ArrayList) addBooks();
        Iterator itr = bookList.iterator();
        while (itr.hasNext())
        {
           Book bookData = (Book) itr.next();
            book.add(bookData.getID());
            book.add(bookData.getBookName());
            book.add(bookData.getAuthor());
            booksArray.add(book);
            book = new JSONArray();
        }
        return booksArray;
    }
}
```

A method named addBooks() has been defined to add the Book object to the ArrayList named bookList. An instance of the Book class has been initialized with appropriate values for the data members. The add() method of the ArrayList class has been used to store an instance of the Book class in the ArrayList object. The return statement returns the ArrayList object to the calling method.

The dispalyBookData() method has been defined to convert the book data to JSON format.

A new JSONArray has been created. The addbooks() method is invoked and the ArrayList object returned from the method is stored in an ArrayList object named bookList.

The iterator loops through the result using the hasNext() method to retrieve the object stored in the ArrayList class. The object is converted to Book type by explicitly casting the record. Next, the ID of the book is retrieved and stored in the JSON array. The object is added to the JSON array that converts the data into JSON format. Finally, JSON array containing the book data in JSON format is returned to the calling method.

- **Populating the Widget**

Code Snippet 7 demonstrates how to populate the widget.

Code Snippet 7:

```
<jsp:useBean   id="bookBean"   scope="session"   class="src.
BookApplicationBean" />

    <a:widget name="dojo.table"
              value="{columns:[ {label : 'ID', id : 'id'},
              {label : 'Name', id : 'name'},
              {label : 'Author', id : 'author'},
              {label : 'Price', id : 'price'} ],
              rows:${bookBean.displayBookData()}}"
              />
```

The useBean tag is used to access the property of the bean, BookApplicationBean.

The widget, Dojo table is added to the Web page. The widget name is specified using the name attribute of the widget tag.

The column data has been added as the displayBookData() method does not create the column data. JSONObject API uses HashMap which inserts data in any order. To maintain insertion order, the column data is entered directly in the tag.

The row data is obtained by referencing the method from the rows attribute. The displayBookData() method returns the row data in JSON format.

4.1.8 Data Models

Data models specify the type of data expected by various widgets. jMaki uses standardized data model to simplify communication between various widgets.

Following are some of the features of data models:

- Data models are standard for widgets, such as combo boxes, menus, trees, tables, and so on, across toolkits.
- Data model of a widget can be used in any toolkit without changing the format of the data. When you want to use a widget in another toolkit, the widget wrappers convert the jMaki data model as per the data requirement of the new toolkit.
- Data model of a widget includes publishers and subscribers. These allow and simplify dynamic update and communication between widgets.
- Data models across various widgets have similar properties and events. If you learn one data model, most of the information of the data model will be applicable for other data models too.

Figure 4.7 shows the data model of jMaki.

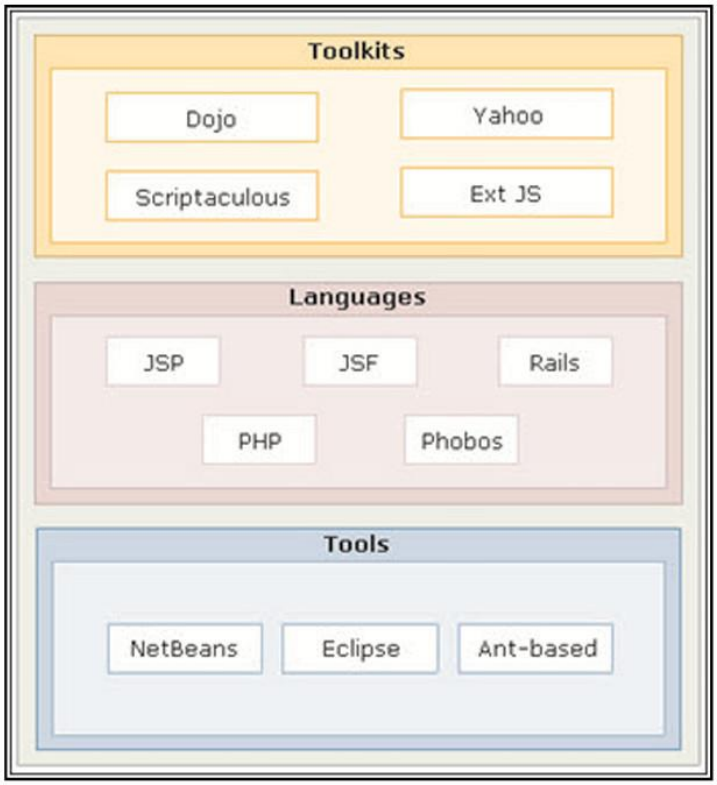


Figure 4.7: Data Model of jMaki

4.1.9 Types of Data Models

The data models supported by various widgets are listed in table 4.1.

Data Model	Widget Supporting the Data Model
jMaki Menu	Yahoo Menu, jMaki Menu, jMaki Tab Menu, jMaki Accordion Menu
jMaki Table	Yahoo Datatable, Dojo Table
jMaki Tree	Yahoo Tree, Dojo Tree
jMaki Combobox	Dojo Combobox
jMaki Multiview	jMaki Dynamic Container, Dojo Accordion, Dojo Tabbedview, Yahoo
Tabbedview	
jMaki Fisheye	Dojo Fisheye
jMaki Drawer	Dojo Drawer
jMaki Map	Yahoo Map, Google Map

Table 4.1: Widgets Supporting Data Models

Session 04

AJAX Client Frameworks–II

4.1.10 Common Properties

There are a set of properties which are common among the different data models. They are as follows:

- **id** - Indicates the identifier of items such as table row, tree node, tab, and so on.
- **label** - Indicates the title of items such as table column, tree node, tab, and so on.
- **href** - Indicates that the string will act as a hyperlink. Clicking the link will navigate to the specified url.
- **include** - Indicates that the content from the specified url will be included in the page.
- **action** - Indicates that the object communicates an action to be performed by a widget.
- **targetId** - Indicates the id of an element on which a specific action is to be performed. The id of the target element and targetId of the data model should be the same.

4.2 JQuery

JQuery is a concise and fast JavaScript library. It was developed by John Resig in 2006 to do more and write less. It simplifies event handling, animating, document traversing, and AJAX interactions implementation.

The core features of jQuery are as follows:

- **Lightweight**: It is a very lightweight library. It is about 19 KB in size.
- **DOM Manipulation**: DOM elements can be selected, traversed, and modified easily in jQuery. It uses the cross-browser open source selector engine called as Sizzle.
- **Cross Browser Support**: jQuery has cross-browser support and works well in Fire Fox 2.0+, IE 6.0+, Chrome, Safari 3.0+, and Opera 9.0+.
- **Handling Events**: jQuery captures a wide variety of events elegantly such as a user clicking a link.
- **AJAX Support**: jQuery helps in developing a feature-rich and responsive site using AJAX technology.
- **Other Technology Support**: jQuery supports basic XPath syntax and CSS3 selectors.
- **Animations**: jQuery provides several built-in animation effects which can be used in Websites.

4.2.1 JQuery Selectors

The Cascading Style Sheets (CSS) selectors are harnessed even in JQuery library for easy and quick access to the elements or groups of elements in the Document Object Model (DOM).

Table 4.2 lists some of the commonly used selectors.

Fault Subelement	Description
$(#id)	It selects the element with given id. For example, $("#container"), will select the element with id as container.
$(.class)	It selects the element with given class. For example, $(". name"), will select the element with name as container.
$('*')	This selector selects all elements in the document.
$(element)	It selects all the elements of the type mentioned on the Webpage respectively. For example, ("<p>"), will select all the <p> elements.
$("element:first") and $("element:last")	It will select the first and last element on the Webpage of the specified element. For example, $("p:first"), will select first <p> element.
$("element:odd") and $("element:even")	It will select the odd and even positioned elements on the Webpage of the specified element. For example, $("tr:odd"), will select all odd <tr> element.

Table 4.2: Commonly Used Selectors

Following is the process for using the JQuery library:

1. Download the latest version of jQuery from http://jquery.com/.
2. To use the JQuery library, directly place the file in the directory where the Web page resides.
3. For using it in the particular Web page, use the script tag.

   Consider a JSP page where number of tags are to be counted on the click of a button.

   Code Snippet 8 demonstrates the use of selectors. The index.jsp page uses the jQuery library.

Code Snippet 8:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
         <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>Selectors Example</title>
     <script type="text/javascript" src="jquery-1.11.1.js"></
script>
       <script type="text/javascript" language="javascript">
            $(document).ready(function() {
                $("button").click(function() {
    var count = $("li").length;
    $("#div1").text("There are " + count + " li tags");
```

```
                });
            });
        </script>
    </head>
    <body>
        <div id="container"> Name of the students
            <ul>

                <li>Alex</li>
                <li>Tony</li>
                <li>Alice</li>
                <li>Steven</li>
                <li>Andrew</li>
            </ul>
        </div>
        <button> Click Me </button>
        <br/>
        <div id="div1"> </div>
    </body>
</html>
```

In Code Snippet 8, the <li> tags on the index.jsp page are counted using the $("li") selector. The data is placed inside a div tag which is accessed using the $("#div") selector. Also, note the events such as load and click are bounded to the script using selectors $(document).ready and $("button").click.

The output for the index.jsp is as shown in figure 4.8. At first there are no contents in the tag with id div1.

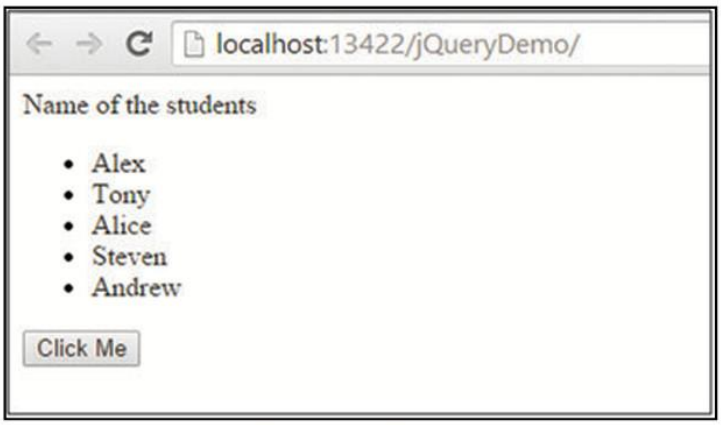


Figure 4.8: JSP Page

On click of the button, the function bounded with the click event is called and the count of the <li> tag is returned in the tag as shown in figure 4.9.

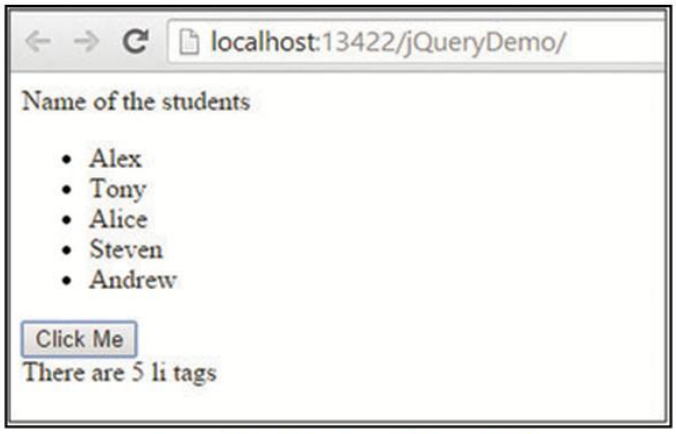


Figure 4.9: Output on Button Click

4.2.2 jQuery Callback Functions

The statements in JavaScript are executed line by line. However, with effects, the next line of code can be executed even if the effect is not completed. This can lead to errors due to incomplete effects.

A callback function can be created to prevent this. The callback function is executed once the effect is fully completed. The example of the callback function is as follows:

```
$(selector).hide(speed,callback);
```

The callback function is executed after the hide() method execution is completed.

```
$("button").click(function(){
  $("p").hide("slow",function(){
    alert("The paragraph is now hidden");
  });
});
```

Here, the callback parameter, that is a function, will be executed after the hide effect is completed.

4.2.3 JQuery Properties

The JQuery object is associated with properties. Table 4.3 shows the commonly used properties.

Property	Description
jquery	Contains the version number of jQuery.
jQuery.fx.interval	It changes the rate of firing animation in milliseconds.
jQuery.fx.off	It enables/disables animation globally.

Property	Description
jQuery.support	It contains properties of different browser features and bugs which can be used internally by jQuery.
length	It is the count for elements in a jQuery object.

Table 4.3: Common jQuery Properties

4.2.4 JQuery AJAX

jQuery provides several functions for implementing AJAX functionality. The text, XML, HTML, or JSON content can be requested with JQuery AJAX methods from a remote server. The data can also be loaded in the selected HTML element directly. A powerful and simple method of JQuery used for AJAX is the load() method that loads data from a server. The returned data is displayed/added in the selected element. The syntax of the load method is as follows:

Syntax:

```
$(selector).load(URL,data,callbackfunction);
```

The URL parameter specifies the URL from where the data is to be loaded.

The data parameter is optional and specifies the information to be sent along with the request.

The callback parameter is optional and indicates the name of a function which is to be executed after the load() method is completed.

Code Snippet 9 demonstrates the use of jQuery library and the AJAX technology within the index.jsp Web page.

Code Snippet 9:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
    <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>JQuery Demo</title>
        <script type="text/javascript" src="jquery-1.11.1.js">
</script>
        <script>
        $(document).ready(function(){
          $("button").click(function(){
            $("#container").load("Demo_test.txt");
          });
        });
        </script>
```

```
    </head>
    <body>
        <div id="container">
            <h2>Let jQuery AJAX change this text.</h2>
        </div>
        <br/>
        <button>Get File Content</button>
    </body>
</html>
```

In Code Snippet 9, the data is asynchronously brought from a file on click of the button.

The load() method is used for bringing the contents from the Demo_test.txt file. The contents are loaded in the tag which is referred using the selector, $("#container").

The contents of the Demo_test.txt file are shown in figure 4.10.

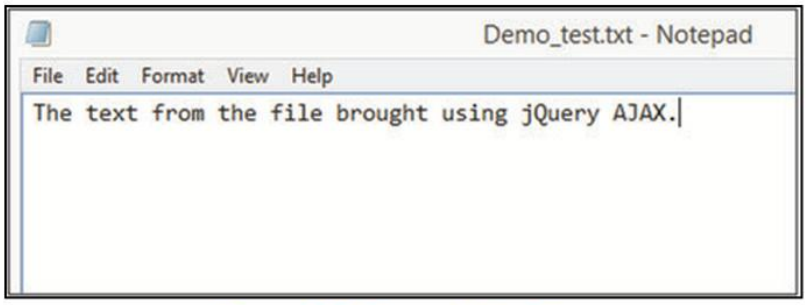


Figure 4.10: Demo_test.txt File

The output for the index.jsp page is as shown in figure 4.11.

Figure 4.11: Output on Page Load

On clicking the button, the data is brought from the file and the content on the div is modified as shown in figure 4.12.

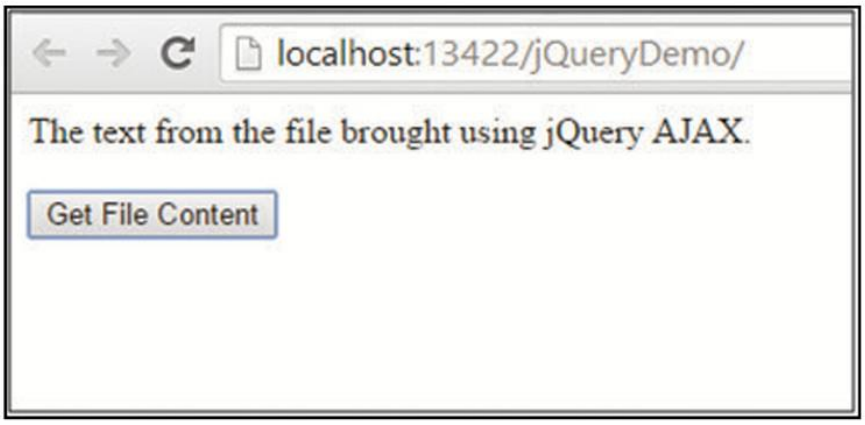


Figure 4.12: Output after Button Click

4.3 YUI

YUI is a robust, fast, and scalable framework. It is a JavaScript library which runs on mobile, server, or desktop browsers. It is a world-class platform for application developers, hackers, and novices. It provides convenient APIs which are fast and lightweight, with comprehensive suite of tools.

There are currently two supported versions of YUI namely, YUI 2 launched in 2006 and YUI 3 which was released in 2009. YUI 3 has a completely new syntax, which improves its ease of use.

Following are the YUI features that have made it popular:

- A huge library of widgets, including one of the most feature-complete datatables.
- Stellar documentation whereby each component and widget has detailed instructions, examples, and api documentation.
- YUI provides a number of rich development tools including a profiler, in-browser console, and testing framework.
- Flexible event handling with built-in support for touch and gesture events.

4.3.1 Sandboxing

This is a new feature in YUI 3. It states each instance in YUI is protected, limited, and self-contained. Thus, the other YUI instances are segregated and specific need for functionality is tailored properly.

The process of creating isolated iframe sandboxes is simplified by sandbox where tasks such as unit testing and profiling are done. Code Snippet 10 demonstrates the use of sandbox.

Code Snippet 10:

```
<script>
// Create a YUI sandbox.
YUI().use('node', 'event', function (Y) {
    // The Node and Event modules are loaded and can be used.
});
</script>
```

4.3.2 Selectors

The Selector class can be used to provide support for using the CSS selector for querying the DOM.

It returns an HTML Element. The methods of the Selector class are shown in table 4.4.

Methods	Descriptions
query()	It is used for attaching an event to number of elements.
test()	It returns true or false depending on the node, if matched with selector provided.
filter()	It returns a set of nodes based on a simple selector.

Table 4.4: Methods of Selector Class

All these event methods present in YUI2 are deprecated.

4.3.3 Custom Events

The DOM events are also handled in YUI using the Event Utility API's.

The YUI Event Utility provides APIs for working with DOM model. The YUI Custom Event System allows to define and use events which are not included in the DOM — events that are specific to an application. Custom Events work much like DOM events. They can bubble, pass event facades, have their default behaviors and propagation suppressed, and so on.

The EventTarget class provides the API for the custom events. EventTarget is extended by all other classes, but if the custom event APIs are required, user can augment or extend classes with EventTarget directly.

4.3.4 Nodes and NodeList

The two methods provided by YUI for accessing DOM nodes through its Node module are as follows:

- Y.one('selector') - returns a YUI Node representing a DOM Node.
- Y.all('selector') - returns a YUI NodeList of all node matches.

Session 04

AJAX Client Frameworks–II

The steps required for using YUI framework are as follows:

1. Download the seed file for YUI from http://yuilibrary.com/.
2. Place it in the Web Pages folder for using it.

Consider the scenario where all features of the YUI listed in the session have been used. Code Snippet 11 demonstrates use of events, selectors, and sandboxing.

Code Snippet 11:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<!DOCTYPE html>
<html>
    <head>
            <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>JSP Page</title>
       <script src="yui-min.js"></script>
        <script>
            YUI().use("node", function(Y) {
                Y.one('ul').delegate("click", function(em) {
             var itemList = em.container.all('li');
                    var node = em.currentTarget;
                          alert("You clicked link " + itemList.
indexOf(node));
                    node.addClass("Clicked");
                    alert(node.getAttribute("class"));
                }, 'li');
            });
        </script>
    </head>
    <body>
        <h2>List for Students</h2>
        <ul>
            <li>Alex</li>
            <li>Tony</li>
            <li>Alice</li>
            <li>Steven</li>
            <li>Andrew</li>
        </ul>
    </body>

</html>
```

In Code Snippet 11, sandboxing feature is used by using YUI().use. The nodes are accessed using the 'all' function. On click of a node, a message is shown about the node.

Figure 4.13 shows the output of the index.jsp page.

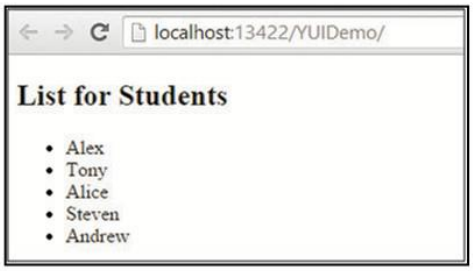


Figure 4.13: Output

On clicking any name, the index of the node will be displayed in the message box as shown in figure 4.14.

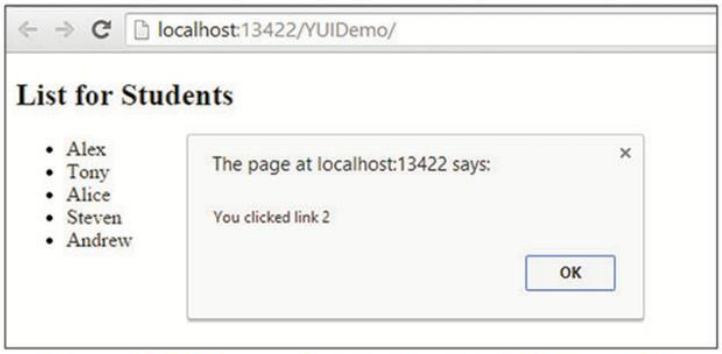


Figure 4.14: Message Box Displayed on Clicking a Link

On clicking Alice, this message box is shown and the event delegation is seen when the next message box shows the class name of the li tag.

Figure 4.15 shows event delegation output.

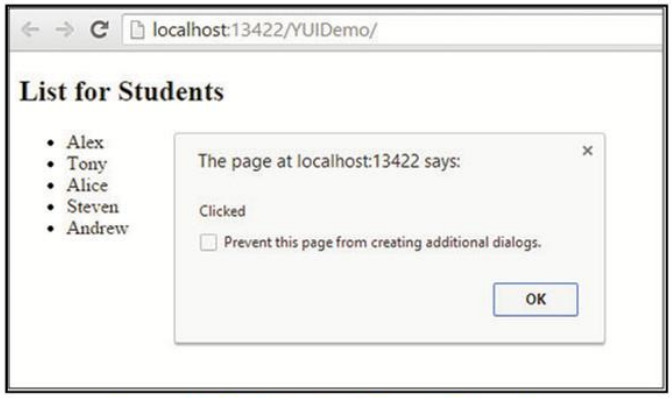


Figure 4.15: Output of Event Delegation

Session 04

AJAX Client Frameworks–II

Check Your Progress

1. Match the descriptions with their appropriate technology.

Description		Technology	
(a)	Uses HTML and CSS standards for creating Web applications	(1)	jMaki Widget Model
(b)	Provides APIs for performing XMLHttpRequest	(2)	jMaki Layouts
(c)	Provides a component model for widgets	(3)	jMaki Server Runtime
(d)	Binds the server-side runtime with the client runtime	(4)	XMLHttpProxy
(e)	Allows access to the external services	(5)	jMaki Client Services

(A)	a-2, b-3, c-1, d-5, e-4	(C)	a-3, b-2, c-5, d-1, e-4
(B)	a-4, b-3, c-1, d-5, e-2	(D)	a-5, b-4, c-3, d-2, e-1

2. Which of the following statements describing the features of jMaki are true?

(a)	AJAX components are wrapped as a JSP tags or JSF components.
(b)	Developers cannot reuse the widgets present in different libraries.
(c)	jMaki does not support the multiple server technologies.
(d)	jMaki standardizes the data and event model.
(e)	jMaki follows 80/20 rule.

(A)	a, b, c	(C)	b, d, e
(B)	a, d, e	(D)	c, d, e

Session 04

AJAX Client Frameworks–II

Check Your Progress

3. Which of the following statements about jQuery are true?

(a)	JQuery is a concise and fast JavaScript library.
(b)	It simplifies event handling, animating, document traversing, and AJAX interactions implementation.
(c)	jQuery does not provide cross-browser support.
(d)	It is a very heavyweight library.

(A)	a, b	(C)	b, d
(B)	c, d	(D)	a, c

4. Which feature segregates the specific functionality and targets in YUI?

(A)	Isolation	(C)	Sandboxing
(B)	Nodelists	(D)	Selectors

5. Which open source engine is used by jQuery?

(A)	JVM	(C)	Apache
(B)	Google	(D)	Sizzle

Session
04

AJAX Client Frameworks–II

Answers

1.	A
2.	B
3.	A
4.	C
5.	D

Session 04

AJAX Client Frameworks–II

Summary

- jMaki is a lightweight client-server framework used for creating AJAX applications. It wraps the functionality of JavaScript technology.
- jMaki Architecture comprises client and server components.
- The three core resource files for a widget are component.css, component.js, and component.htm.
- jQuery is a concise and fast JavaScript library developed by John Resig in 2006 to do more and write less.
- The callback function in jQuery is executed once the effect is fully completed.
- YUI is a robust, fast, and scalable framework. It is JavaScript library which runs on mobile, server, or desktop browsers.
- The process of creating isolated iframe sandboxes is simplified by sandbox where tasks such as unit testing and profiling are done.

05 Google Web Toolkit

Welcome to the Session, **Google Web Toolkit**.

This session explains about Google Web Toolkit (GWT) used for developing Rich Internet Applications (RIAs). It describes the features Google Web toolkit, Google maps, and introduces the Google AJAX search API.

In this Session, you will learn to:

- Describe Google Web Toolkit
- Work with Google Maps
- Describe Google AJAX search API

Session 05

Google Web Toolkit

5.1 Google Web Toolkit

Google Web Toolkit (GWT), pronounced as 'gwit', is an open source development kit provided by Google. It allows Java developers to develop rich applications without much expertise on client-side technologies such as JavaScript, CSS, and so on. When using GWT, everything appears as Java objects. Now, one might question that 'How can Java replace JavaScript?'. The answer is the GWT compiler as shown in figure 5.1.

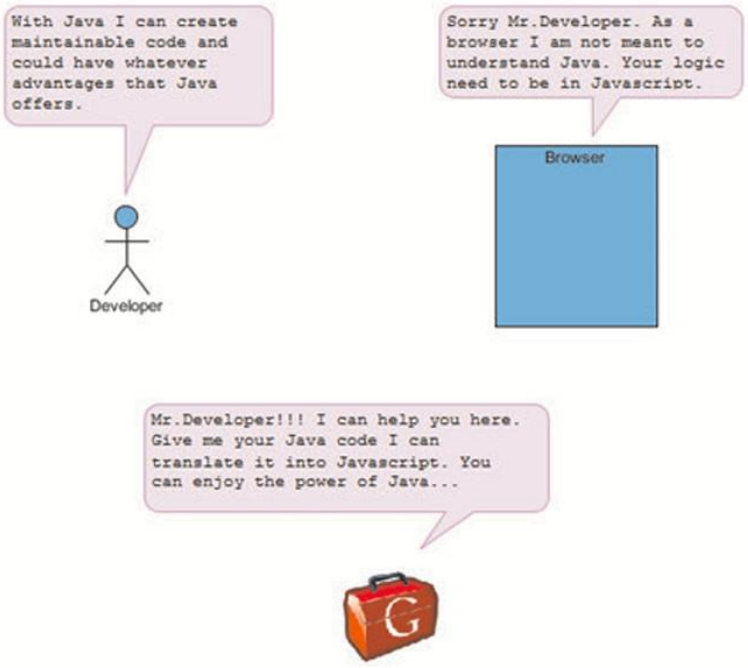


Figure 5.1: Google Web Toolkit

Anything that is written in Java, is translated by the GWT compiler into JavaScript/CSS/ HTML as applicable. Ultimately, the developer receives the client-side code with the functionalities that were required using Java.

The application written in GWT is cross-browser compliant. GWT automatically generates JavaScript code suitable for each browser. GWT is completely free and licensed under the Apache License version 2.0.

Thus, GWT is a framework for developing large scale, high performance, and easy-to-maintain Web applications.

Advantages of GWT

The advantages of GWT are as follows:

- Since GWT is Java based, one can use Java IDEs such as Eclipse, NetBeans, and so on for developing a GWT application.

- This helps developers to use the code auto-complete, navigation, refactoring, project management, and all other features of the IDEs. Hence, GWT has a low learning curve for Java developers.
- GWT also provides debugging capability so that the client-side application can be debugged just as a Java application.
- GWT provides easy integration with JUnit and Maven.
- GWT generates browser-specific and optimized JavaScript code.
- GWT comes with a Widgets library that helps to implement most of the functionality required in an application.
- GWT is also extensible and allows creation of custom widgets to cater to application-specific requirements.

Moreover, applications made using GWT can run on all major browsers as well as Smartphones including Android and iOS based phones/tablets.

Disadvantages of GWT

Along with all the advantages, GWT comes with few drawbacks as follows:

- Web pages generated by GWT cannot be indexed by search engines because these applications are generated dynamically.
- If JavaScript is disabled in the browser then, user will only be able to view the basic page and nothing else.
- GWT is not suitable for Web designers who prefer using plain HTML with placeholders for inserting dynamic content at later point in time.

GWT Components

The GWT framework can be divided into the following three major parts:

- **GWT Java to JavaScript Compiler**: This is the most significant component of GWT that makes it a powerful tool for building RIAs. The compiler translates all the application code written in Java into JavaScript.
- **JRE Emulation Library**: GWT consists of a library that emulates a subset of the Java runtime library. The list includes java.lang, java.math, java.sql, java.lang.annotation, java.io, java.util, and java.util.logging.
- **GWT UI Building Library**: This component of GWT consists of many subparts such as the actual UI components, History management, RPC support, and so on.

Additionally, GWT provides a **GWT Hosted Web Browser** that allows executing the GWT applications in hosted mode, wherein the code runs as Java in the Java Virtual Machine without compiling to JavaScript.

A GWT application consists of the following four important parts:

- Module descriptors
- Public resources
- Client-side code
- Server-side code

The last component is optional but first three components are mandatory.

5.1.1 Google Web Toolkit Setup

To use Google Web Toolkit, it is recommended to use the latest version of the NetBeans IDE.

Steps to Install the Google Web Toolkit Plugin

The steps to install the GWT plugin are as follows:

1. Download the latest plugin from http://plugins.netbeans.org/plugin/44509/gwt4nb.
2. Go to Tools → Plugins.
3. Click the Downloaded tab and then click Add Plugins.
4. Select the plugin file that has just been downloaded, as shown in figure 5.2.

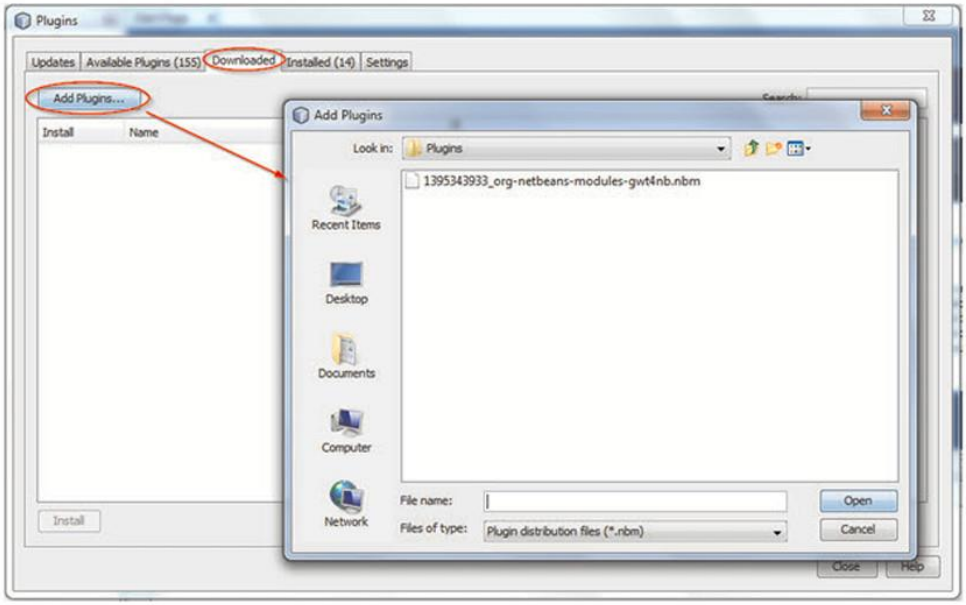


Figure 5.2: Add Plugins Dialog Box

5. Once the plugin is added, click Install to install it as shown in figure 5.3.

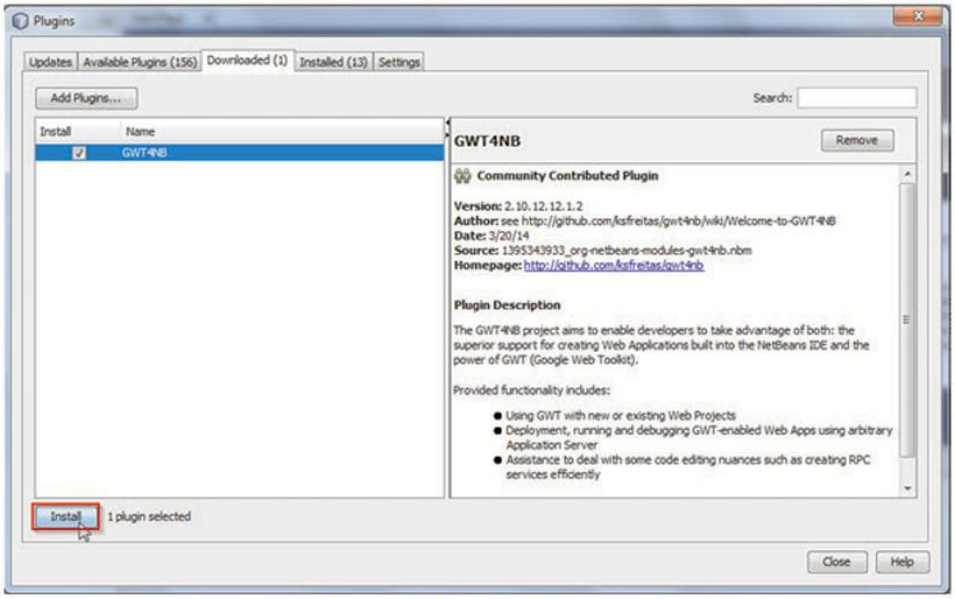


Figure 5.3: Install Plugins

Perform a regular installation and ignore any warning or message. A restart will be required after the plugin install.

6. Download and unzip the GWT SDK, gwt-2.7.0 from http://www.gwtproject.org/download.html?csw=1.

This contains the core libraries, compiler, and development server that you need to write Web applications.

Steps to Create a Google Web Toolkit Project

To create a GWT project, perform the following steps:

1. Create a Java Web Application project by clicking File → New Project.
2. Select Java Web → Web Application as shown in figure 5.4 and click Next.

Session 05

Google Web Toolkit

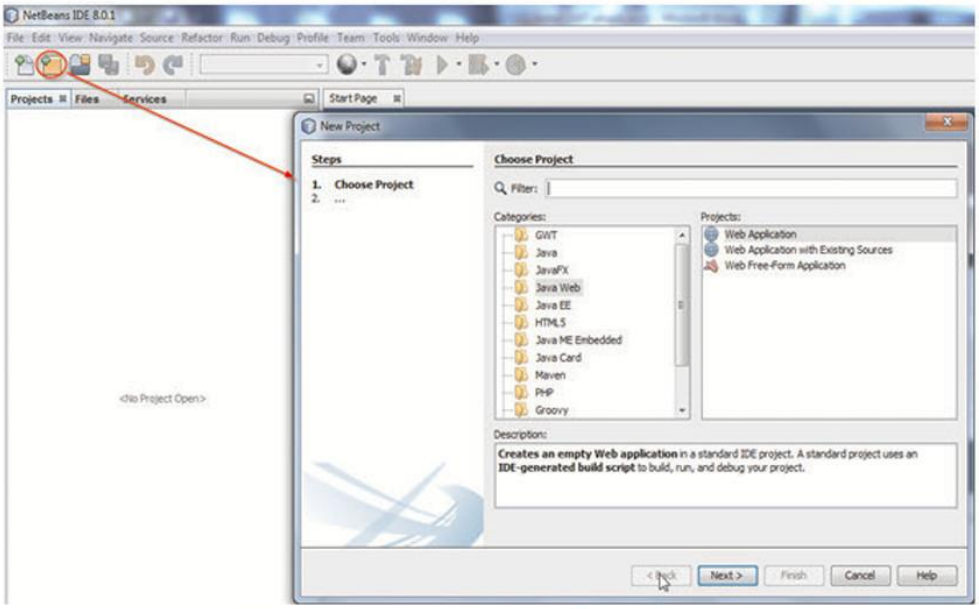


Figure 5.4: Creating a Java Web Application

3. Specify the Project Name as shown in figure 5.5 and click Next.

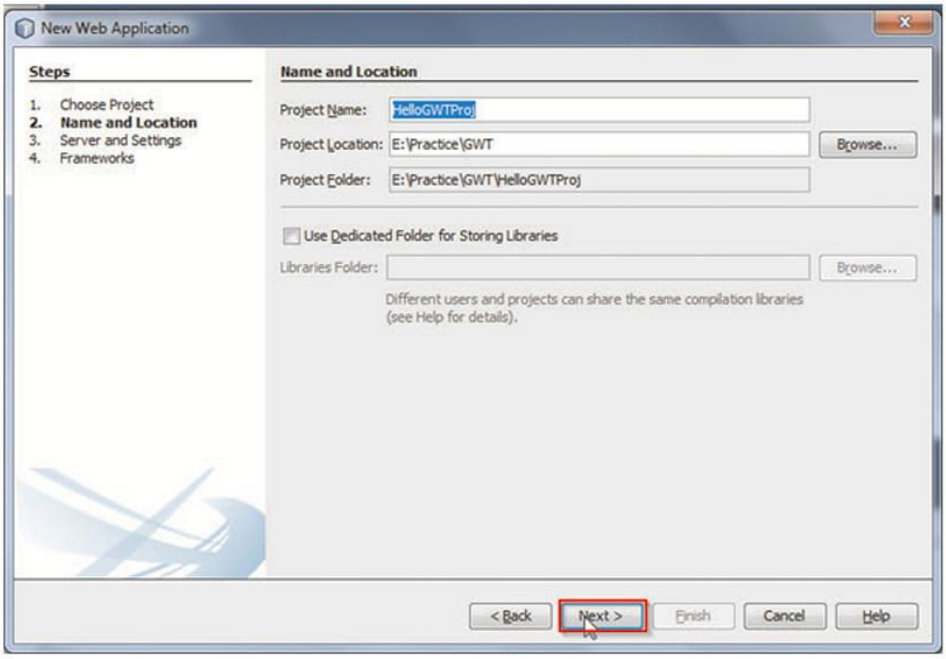


Figure 5.5: Adding the Project Name

4. Select the application server GlassFish Server 4.1, as shown in figure 5.6.

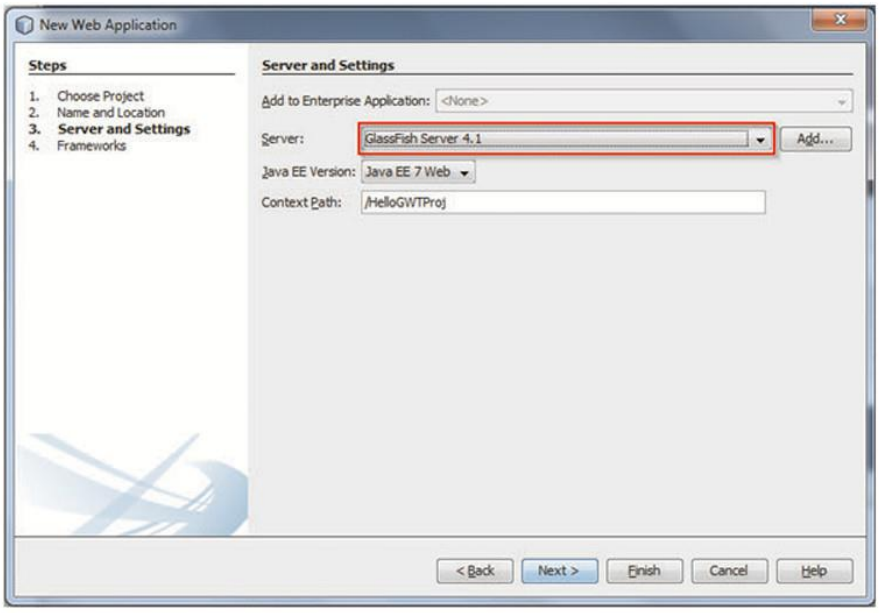


Figure 5.6: Selection of Application Server

5. Select the latest GWT version that has been installed from the Frameworks list, set the location of GWT SDK, and specify the GWT Module name as shown in figure 5.7. Click Finish.

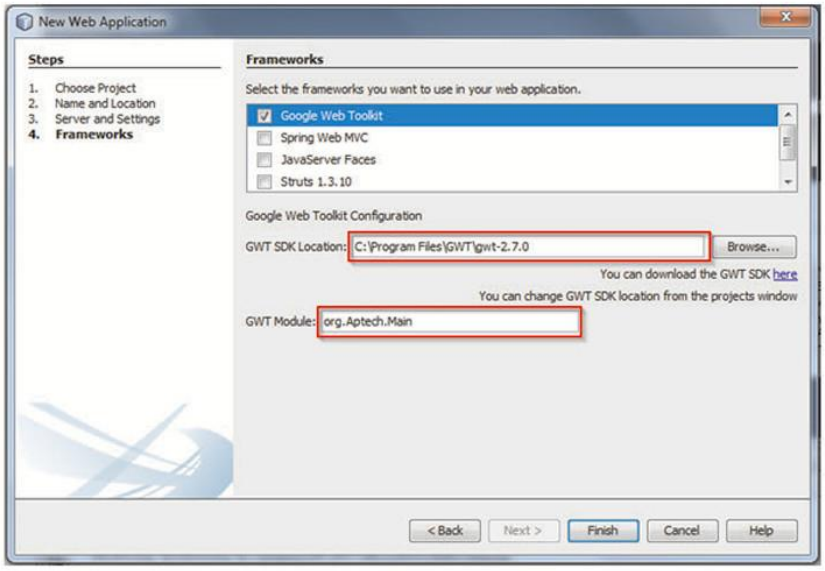


Figure 5.7: Selection of Module Name

Note - One can also create the application as a GWT project, but that will be created as a Maven project.

6. Right-click the project and open the project properties.

7. Change the GWT version 2.7 to version 2.6 since version 2.7 got released recently and is not supported by the GWT plugin. Figure 5.8 shows how to change the version of GWT.

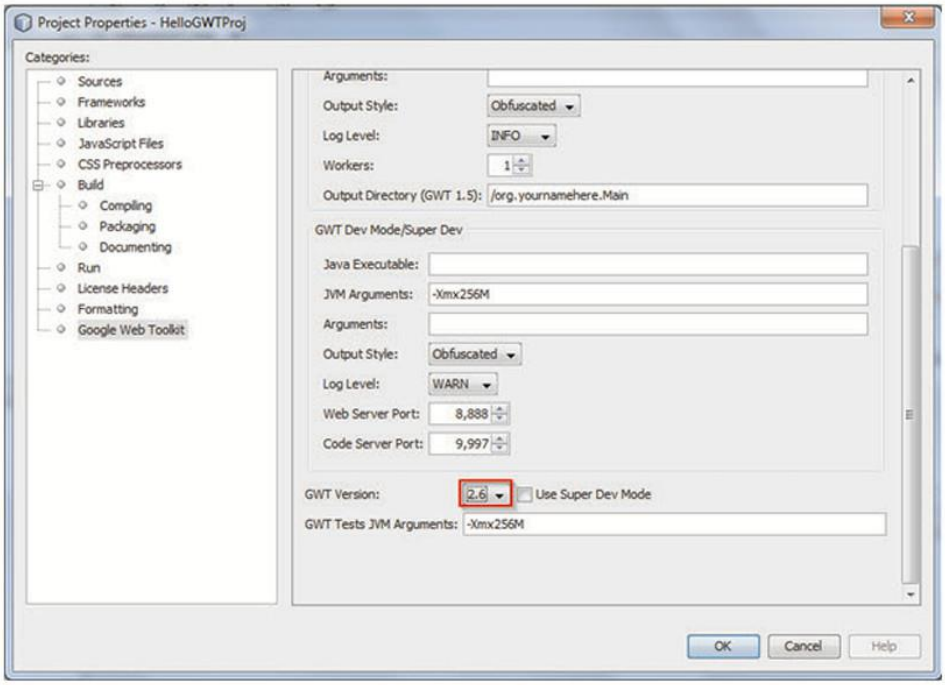


Figure 5.8: Project Properties

8. Click OK.

5.1.2 GWT Basics

To work with GWT, first it is required to understand the following building blocks of GWT and GWT terminologies:

1. **EntryPoint**

   The EntryPoint interface belongs to the com.google.gwt.core.client package. It must be implemented by the class that represents the entry point of a module. Any GWT application must have atleast one class that implements com.google.gwt.core.client. EntryPoint. A class that implements the EntryPoint interface to a GWT application is similar to the main() method of a Java program. The only difference is that there can be multiple classes that implement the EntryPoint.

In other words, it allows writing the logic that needs to be converted into JavaScript. Thus, entry point is the class that implements the EntryPoint interface, which will hook-up the GWT code with any HTML, JSP, Servlet, or any other client.

The logic starts with the onModuleLoad() method of the class that implements the EntryPoint interface. Whatever code is written here will be converted into JavaScript and can be included wherever required. The included script takes the responsibility of performing the logic written within the onModuleLoad() method.

Also, ensure that the EntryPoint class contains a default constructor. That is, if there is no constructor defined, it will do, but if there are any parameterized constructors added to the class, a default constructor must also be added.

2. **Module**

A module is an entity that contains one or more entry points. A GWT application can be considered as a collection of one or more modules.

3. **Module Descriptor**

For every module, there is an .xml file called Module Descriptor. For example, for a module with a name, ModuleName, the Module Descriptor will be in the format of ModuleName.gwt.xml.

Note - Though the name Module Descriptor is not a term that is defined in the API, it is a commonly used to refer to the xml file that defines the module.

4. **Host Page**

The logic from the EntryPoint classes is converted into JavaScript files. Hence, there must be some HTML or other server pages that include the scripts and hook-up the GWT functionalities into the application. These pages are referred as Host Pages as shown in figure 5.9.

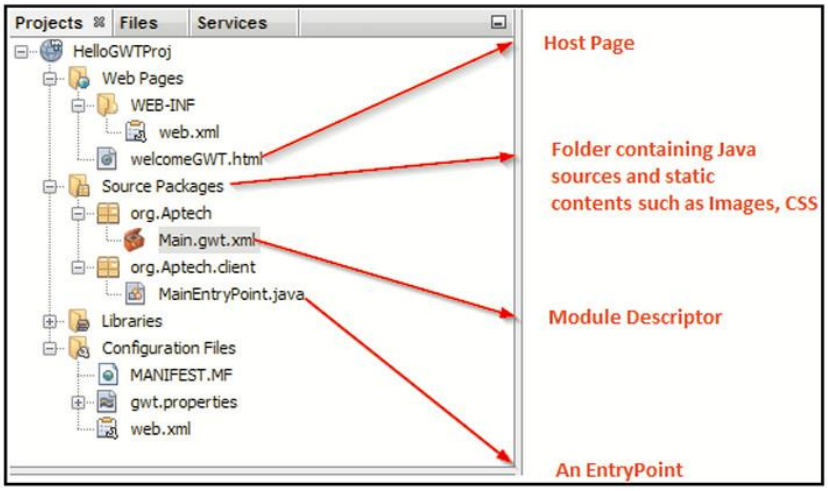


Figure 5.9: Host Pages

5. **Bootstrap file**

This refers to the JavaScript file that has been generated as an equivalent for the GWT module. This is where all the functionality written in GWT is stored. This will be located with the relative path as follows:

```
<script type="text/JavaScript" src="org.Aptech.Main/org.
Aptech.Main.nocache.js"></script>
```

- **CSS Styling APIs**

There are several APIs available to handle CSS settings for any GWT widget. Some important APIs that are helpful in Web programming using GWT are as follows:

1. **public void setStyleName(java.lang.String style)**

   This method removes any existing styles and sets the widget style to the new CSS class provided using style parameter.

2. **public void addStyleName(java.lang.String style)**

   This method adds a secondary or dependent style name to the widget. This name is an additional style name. If there were any previous style names applied, they are retained.

3. **public void removeStyleName(java.lang.String style)**

   This method removes given style from the widget and retains any others associated with the widget.

4. **public java.lang.String getStyleName()**

   This method gets all of style names of the object, as a space-separated list.

5. **public void setStylePrimaryName(java.lang.String style)**

   This method sets the primary style name for the object and updates all dependent style names.

   For example, one can define two new styles which can be applied to a text as follows:

```
.gwt-Big-Text{
 font-size:150%;
}
.gwt-Small-Text{
 font-size:75%;
}
.gwt-Red-Text{
 color:red;
}
```

Now, the setStyleName(Style) method can be used to change the default setting to the new setting. After applying the following rule, a text's font will become larger:

```
txtWidget.setStyleName("gwt-Big-Text");
```

Session 05

Google Web Toolkit

5.1.3 GWT Widgets

GWT Widgets are UI components which can be used as per requirement on the Web page. In GWT, a simple Button, Text box, a Date Picker, or a Rich Text Area, everything is a Widget. Widgets are added to the panels which handles user interaction.

Table 5.1 shows the examples of few widgets that are offered by GWT by default and their purpose.

Widget Class	Purpose
com.google.gwt.user.client.ui.Button	Creates an HTML <button> element.
com.google.gwt.user.client.ui.TextBox	Creates a <input type="text"> element.
com.google.gwt.user.client.ui.Tree	Creates a tree widget. A tree widget is the component similar to the one used to navigate to the project in NetBeans or other similar IDEs.
com.google.gwt.user.client.ui.RichTextArea	Rich Text Area refers to the text area that allows entering text and facilitates other options such as formatting, and so on. The Text Area along with the sophisticated toolbar.
com.google.gwt.user.client.ui.PushButton	Push button is similar to a button but has two display texts. One in normal state and one when the button is pushed.
com.google.gwt.user.client.ui.RadioButton	A mutually-exclusive selection radio button widget that creates a radio button for the given group with the given name.
com.google.gwt.user.client.ui.CheckBox	Creates a check box with the given name.
com.google.gwt.user.datepicker.client.DatePicker	Creates a Date Picker element.
com.google.gwt.user.client.ui.PasswordTextBox	Creates a normal text box. That is, <input type="password">.
com.google.gwt.user.client.ui.TextArea	Creates a TextArea, that is, <Textarea></Textarea>.
com.google.gwt.user.client.ui.ToggleButton	Creates a toggle button. A toggle button is similar to a Push button. However, a Push button when clicked, it will be in a pressed state and when it is clicked again it comes back to a normal state.

Table 5.1: Widgets

To use the Widgets, perform the following steps:

- Create an instance of the Widgets and change the characteristics of the widget as required.

- Add the widgets to the panel(s), so that they can be positioned in the proper location of the document. Then, the widget will handle the interaction with the user for which it has been created.

5.1.4 GWT Event Handling

GWT is basically event driven. That is, the action mechanism will be based on the event that occurs on the client-side.

GWT Events are similar to the events in other Java UI frameworks such as AWT or Swing. A UI component informs the other instances through events. For example, when a click occurs in a button, an instance for Click Event will be created that can be made available to other Java instances as applicable.

Note - It is recommend to read about the 'Observer Design Pattern' to have good understanding Event handling.

Following are the building blocks of event handling:

- The UI component in which the event occurs. For example, Button.
- The Implementation class, whose instance needs to be notified by the component when some event occurs. This can be any class that handles the Event.
- A handler interface, which must be implemented by the Class that will handle the event. For example, ClickHandler.
- The event instance, which will be passed by the component. The required details can be retrieved from that instance. For example, ClickEvent.

Observer/Listener Design Pattern

Listener design pattern is a solution for any scenario in which an instance needs to inform one or many instances on some events.

For example, when a button is clicked, some dialog box needs to be displayed and there exists an instance that validates something. Consider that there are three different entities performing all these tasks as follows:

1. The button instance, on which the click happens. Assume an instance of a class MyButton and a method buttonClicked() is being called when the button is clicked.
2. The instance which will show the Dialog. Consider that the instance name is Instance1 and class for that entity is DialogDispatcher. A method named onClick needs to be called when the button is clicked.
3. The instance which does the validation, say the instance name is Instance2 and the class for that entity is Validator. A method named onClick needs to be called when the button is clicked.

The instances of Validator and DialogDispatcher can be set as instance variables of the Button class. Whenever the method buttonClicked() is called in the MyButton class, the onClick event for the member variables of Validator and DialogDispatcher can be invoked.

This works fine when there are only two instances for which the event needs to be updated. However, what if there are 20 such instances interested to know about the click? In such a case, one must update the class definition of MyButton to add a member variable and the code to call the onClick method. This needs to be done every time a new instance has to be updated on the click event.

This situation can be solved by the following Observer/Listener design pattern:

1. Define an interface named, MyButtonClickListener, with a method onClick(MyClickEvent).
2. In the MyButton class, declare a member variable which holds the list of MyButtonClickHandler.
3. In the MyButton class, define a method addClickHandler(MyButtonClickHandler) which adds the instance in the argument to the member variable defined in step 2.
4. Whenever an instance of some class needs to be notified on the button's click, the implementer of the class is asked to implement the interface too. Also, it is the responsibility of the implementing class to decide what is to be done within the onClick method.
5. Whenever a click happens on the button, the MyButton class will iterate through the list of MyButtonClickHandler and if there are any instances, onClick will be called for that instance.
6. One can use MyClickEvent to hold additional data about the clicks, if required.

Hence, with the implementation shown in these steps, it will not be required to update MyButton class every time. This type of implementation for event handling is called as Observer/Listener Pattern.

Steps for Creating a Module

The steps to create a module are as follows:

1. Right-click the project and select New → Other.
2. In the dialog box that appears, select the GWT module, as shown in figure 5.10 and click Next.

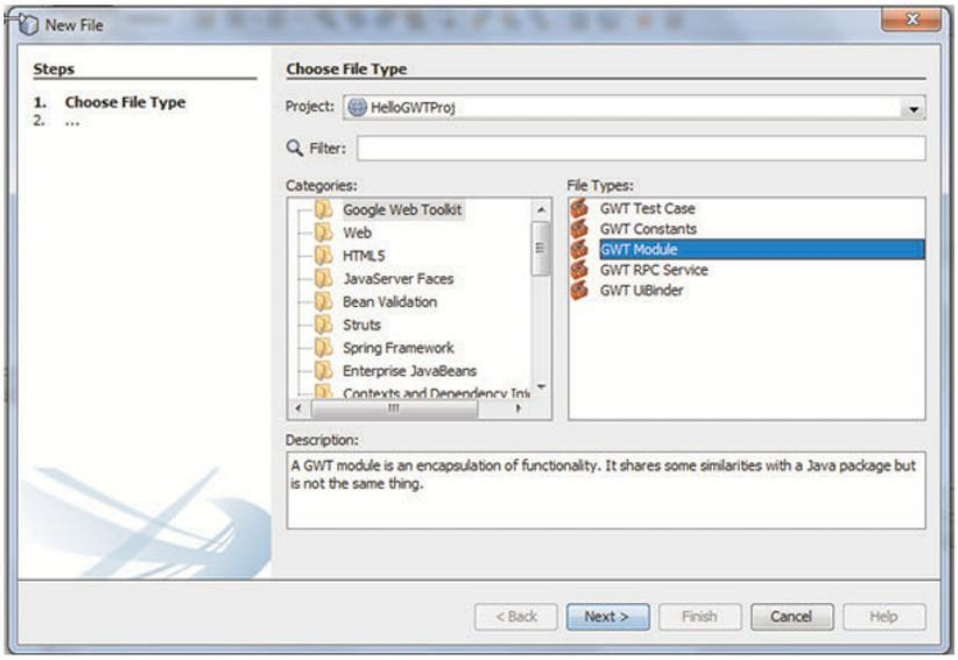


Figure 5.10: Selecting GWT Module

3. In the next screen that appears, specify the Module name and click Finish as shown in figure 5.11.

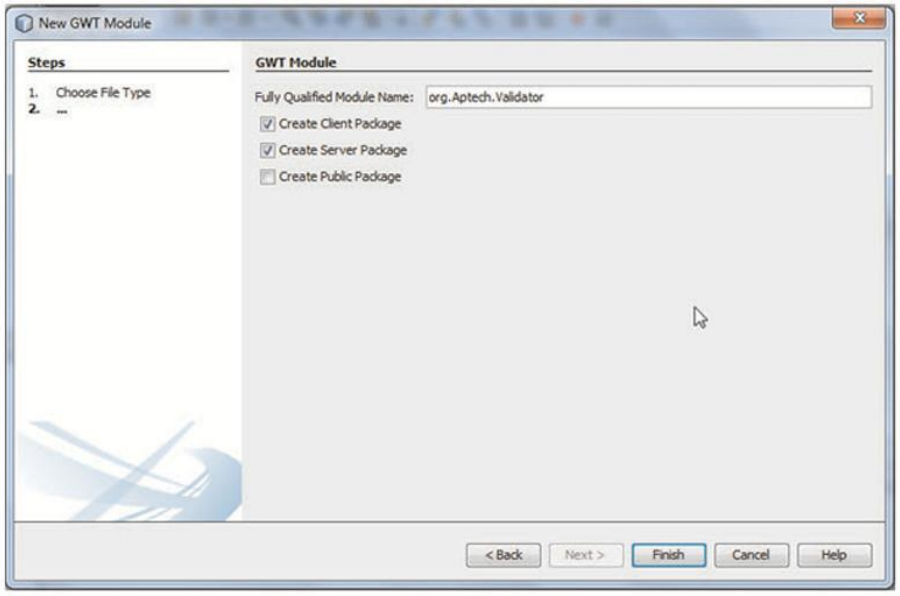


Figure 5.11: New GWT Module

The module descriptor gets created in the source package.

Steps for Creating an Entry Point

The steps to create an entry point are as follows:

1. Right-click the client package and click New → Java Class.
2. Specify the Class Name as shown in figure 5.12 and click Finish.

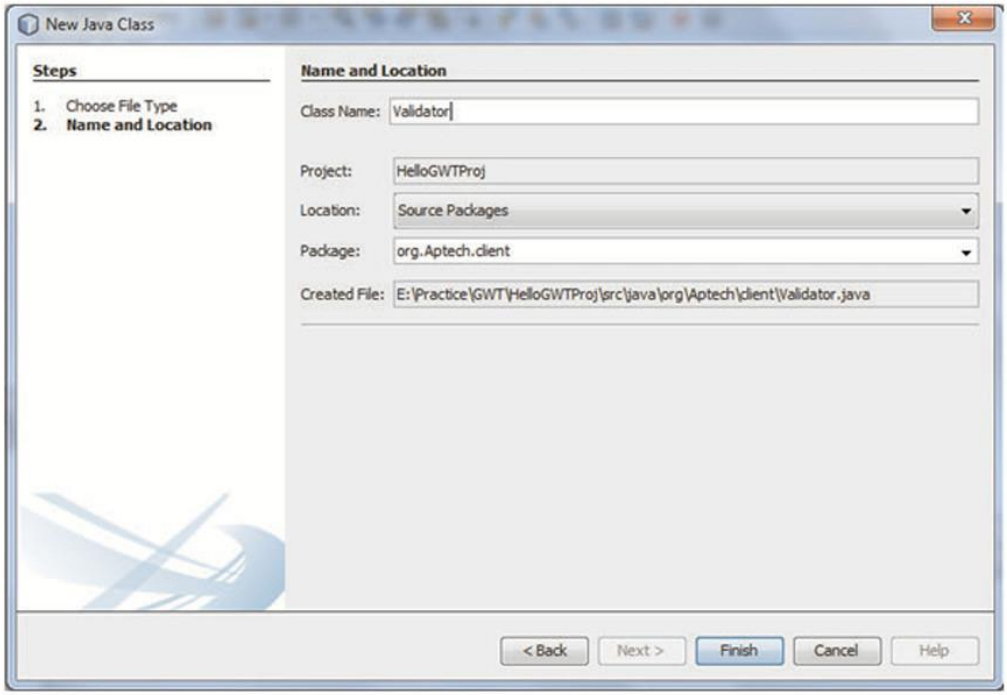


Figure 5.12: New Java Class

Ensure that the Entry Point class is created within the client package or any sub-package of the client.

3. Implement the EntryPoint interface in the Validator class and override the onModuleLoad() method as demonstrated in Code Snippet 1.

Code Snippet 1:

```
public class Validator implements EntryPoint{
    @Override
    public void onModuleLoad()    {
        // TODO: The implementation starts here.
    }
}
```

- **Steps to Add Entry Point to a Module**

  Add the entry for the EntryPoint class in the module descriptor as demonstrated in Code Snippet 2.

Code Snippet 2:

```
<!-- Specify the app entry point class.  -->
 <entry-point class="org.Aptech.client.Validator"/>
```

Note, in this case, the entry point class is org.Aptech.client.Validator.

- **Steps to Hook-up a Module in a Host Page**

  To get the logic written in the Module to work, a bootstrap file must be included in the page in which the module needs to be added. The bootstrap file is present in the following location: <module_name>/<module_name>.nocache.js

  For example, if the GWT module org.Aptech.client.Validator functionality needs to be added in some HTML file, a line must be added to the HTML page as shown in Code Snippet 3.

Code Snippet 3:

```
<script  type="text/JavaScript"    src="org.Aptech.validator/
org.Aptech.Validator.nocache.js">
</script>
```

5.1.5 GWT RPC

GWT Remote Procedure Calls (RPC) is a mechanism with which GWT allows to call the server method from the client code, asynchronously.

From a developer perspective,

- A servlet is created (by extending RemoteServiceServlet). This servlet will contain the method that will be invoked through RPC.
- A client-side interface will be created (by extending RemoteService) which will be responsible for invoking the method in server.
- The Java POJOs will be created for carrying data in this communication.

The serialization of the POJO instances in this communication will be handled by GWT.

The sequence of events is as follows:

- The server-side code remains as it is.
- The client-side code will be translated into equivalent JavaScript. This JavaScript will be responsible for making calls to the server-side servlet. This call is an AJAX call.
- GWT takes the responsibility of making appropriate translations in this communication.

Widget Example

Code Snippet 4 demonstrates an example of creating and adding a simple Toggle button to a Web page.

HelloWidget.html – Present in Web Pages folder of the Project

Code Snippet 4:

```
<html>
    <head>
        <title>Hello Widget!!</title>
        <meta charset="UTF-8">
       <meta name="viewport" content="width=device-width, initial-
scale=1.0">
      <script type="text/JavaScript"  src="org.Aptech.Widget/org.
Aptech.Widget.nocache.js"></script>
    </head>
    <body>
    </body>
</html>
```

Code Snippet 5 demonstrates the module descriptor file Widget.gwt.xml.

<Source Package>/org.Aptech/Widget.gwt.xml

Code Snippet 5:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit
2.0.0//EN"          "http://google-Web-toolkit.googlecode.com/svn/
tags/2.0.0/distro-source/core/src/gwt-module.dtd">
<module>
<!-- Inherit the core Web Toolkit stuff.  -->
     <inherits name='com.google.gwt.user.User' />
<!-- Inherit the default GWT style sheet.  You can change   -->
<!-- the theme of your GWT application by uncommenting   -->
<!-- any one of the following lines.   -->
<inherits name='com.google.gwt.user.theme.standard.Standard' />
<!--   <inherits   name='com.google.gwt.user.theme.chrome.Chrome'/>
-->
<!-- <inherits name='com.google.gwt.user.theme.dark.Dark'/>  -->
<!-- Other module inherits -->

<!-- Specify the app entry point class.  -->
 <entry-point class="org.Aptech.client.MyWidgetDemo"/>
</module>
```

Code Snippet 6 demonstrates the entry point class for the application.

Entry Point Class – MyWidgetDemo.java

Code Snippet 6:

```
package org.Aptech.client;
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.ToggleButton;
public class MyWidgetDemo implements EntryPoint{
    @Override
    public void onModuleLoad()    {
        // Create Instance for your Widget
     ToggleButton aToggleButton = new ToggleButton("Normal State",
"Clicked State");
    // Apply required style as per your wish
        aToggleButton.setPixelSize(150, 20);
        aToggleButton.setTitle("Click to Toggle");
      // Add it to the panel of your wish
      RootPanel.get().add(aToggleButton);
}
}
```

The code creates a ToggleButton with values of strings to be toggled every time the button is clicked. Figure 5.13 shows the output of the code.

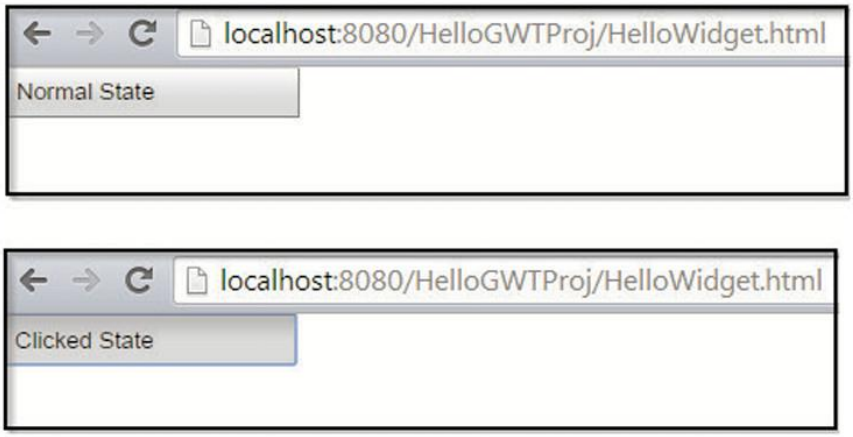


Figure 5.13: Output

Event Handling Example

In this example, a text box will be used to accept the color from the user. Once the user enters it, the keyboard event will be handled and the color of the body of the current document will be set based on the value entered by the user.

Code Snippet 7 demonstrates the creation of the host page.

EventDemo.html - Host Page

Code Snippet 7:

```
<!DOCTYPE html>
<html>
 <head>
  <title>Event Demo..</title>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-
scale=1.0">
  <script type="text/JavaScript" src="org.Aptech.EventDemo/org.
Aptech.EventDemo.nocache.js"></script>
 </head>
 <body>
  <div>
   <p>Enter a valid color :</p>
   <input type="text" id="ipColor" name="ipColor"/>
  </div>
 </body>
</html>
```

Code Snippet 8 demonstrates the module descriptor.

EventDemo.gwt.xml

Code Snippet 8:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit
2.0.0//EN"          "http://google-Web-toolkit.googlecode.com/svn/
tags/2.0.0/distro-source/core/src/gwt-module.dtd">
<module>
<!-- Inherit the core Web Toolkit stuff.  -->
<inherits name='com.google.gwt.user.User' />
<!-- Inherit the default GWT style sheet.  You can change -->
<!-- the theme of your GWT application by uncommenting    -->
<!-- any one of the following lines.   -->
<inherits name='com.google.gwt.user.theme.standard.Standard' />
<!--  <inherits  name='com.google.gwt.user.theme.chrome.Chrome'/>
-->
<!-- <inherits name='com.google.gwt.user.theme.dark.Dark'/>     -->
<!-- Specify the app entry point class. -->
   <entry-point class="org.Aptech.client.EventHandle"/>
</module>
```

Code Snippet 9 demonstrates the entry point class for handling the event.

org.Aptech.client.EventHandle

Code Snippet 9:

```
package org.Aptech.client;
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.event.dom.client.KeyUpEvent;
import com.google.gwt.event.dom.client.KeyUpHandler;
import com.google.gwt.user.client.DOM;
import com.google.gwt.user.client.ui.RootPanel;
import com.google.gwt.user.client.ui.TextBox;

public class EventHandle implements EntryPoint{
    private final static String itsTextId = "ipColor";
    @Override
    public void onModuleLoad(){
        // Get the text box as a Java instance
                  final  TextBox  aTextBox  =  TextBox.wrap(DOM.
getElementById(itsTextId));
        // Add the KeyUpHandler to the text box
        aTextBox.addKeyUpHandler(
    // Anonymous implementation of KeyUpHandler which handles //
the key event
        new KeyUpHandler(){
            @Override
// Set the background color of the <body> element based on the //
value from the text box
     RootPanel.getBodyElement().setAttribute("style","background-
color:"+aTextBox.getValue());
            }
        }
    );
}
}
```

The code creates an instance of the TextBox widget and adds a key event handler to it for recognizing the key up event. The background color of the body tag will be set based on the value entered in the TextBox.

Figure 5.14 shows the output when user enters the color as Orchid.

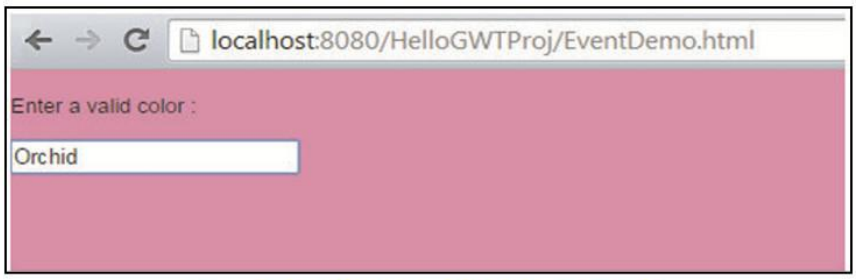


Figure 5.14: Output – Color Given as Text

Figure 5.15 shows the output when hex color code is specified.

Figure 5.15: Output – Hex Color Code

Figure 5.16 shows the output when color is specified in RGB format.

Figure 5.16: Output – RGB Color Code

RPC Example

Code Snippet 10 demonstrates the creation of the RPCMessage class.

RPCMessage

Code Snippet 10:

```
/**
 * The POJO for our RPC call which carries data from server to
client.
 * This class must be serialized and must contain a no-argument
constructor.
 */
public class RPCMessage implements Serializable{
    String itsMessage;
    private static final long serialVersionUID = 1L;
    public RPCMessage (){
    }
        public RPCMessage (String theMessage){
       itsMessage = theMessage;
    }
    public String getServerMessage(){
        return itsMessage;
    }

}
```

Code Snippet 11 demonstrates the creation of the DemoRPCService class.

DemoRPCService

Code Snippet 11:

```
package org.Aptech.client.rpc;
import com.google.gwt.user.client.rpc.RemoteService;
import com.google.gwt.user.client.rpc.RemoteServiceRelativePath;
/**
* The interface which defines the method that can be used by the
client.
 * This will be implemented by the target servlet in the server.
 */
@RemoteServiceRelativePath("rpcdemo")
public interface DemoRPCService extends RemoteService {
    RPCMessage getServerMessage();
}
```

The RemoteService interface declares the getServiceMessage() method to return the message from the server.

Code Snippet 12 demonstrates the creation of the DemoRPCServiceImpl class.

DemoRPCServiceImpl

Code Snippet 12:

```
package org.Aptech.client.rpc.server;

import com.google.gwt.user.server.rpc.RemoteServiceServlet;
import java.util.Date;
import org.Aptech.client.rpc.DemoRPCService;
import org.Aptech.client.rpc.RPCMessage;
/**
 * The Servlet, which is the target for the RPC call.
* This is where the methods which is called through RPC lies.
*/
public class DemoRPCServiceImpl extends RemoteServiceServlet
implements DemoRPCService{
    /**
     * The target method that we will invoke from the client
     * @return instance of RPCMessage which is actually needed
     * in the client.
   */
    @Override
    public RPCMessage getServerMessage(){
      RPCMessage aMessage = new RPCMessage("The current date is: "+
new Date());
       return aMessage;
    }
}
```

The message to be returned from the server is set in the RPCMessage object.

Code Snippet 13 demonstrates the creation of the DemoRPCServiceAsync class.

Code Snippet 13:

```
package org.Aptech.client.rpc;
import com.google.gwt.user.client.rpc.AsyncCallback;

/**
 * The client side interface which will be used in client side.
 * There will be an auto-generated class of this type in client
side.
 * The instance for the auto-generated class will be given by GWT.
This interface variable will be used to invoke the server methods.
 *
 */
```

```
public interface DemoRPCServiceAsync{
  void getServerMessage(AsyncCallback<RPCMessage> theCallback);
}
```

Code Snippet 14 demonstrates the creation of the RPCMessageCallBack class.

RPCMessageCallBack

Code Snippet 14:

```
package org.Aptech.client.rpc;
import com.google.gwt.user.client.Window;
import com.google.gwt.user.client.rpc.AsyncCallback;
/**
 * Implementation of call back. After the RPC call to the server
 * onSuccess will be called if the invocation is successful.
 * onFailure method is called otherwise.
 */
public class RPCMessageCallBack implements
AsyncCallback<RPCMessage>{
    @Override
    public void onFailure(Throwable caught) {
        Window.alert("Error while communicating the server.");
    }

    @Override
    public void onSuccess(RPCMessage result){
    Window.alert("Message from server"+result.getServerMessage());
    }
}
```

Code Snippet 15 demonstrates the creation of the RPCEntryPoint class.

RPCEntryPoint

Code Snippet 15:

```
package org.Aptech.client;

import org.Aptech.client.rpc.DemoRPCService;
import com.google.gwt.core.client.EntryPoint;
import com.google.gwt.core.client.GWT;
import com.google.gwt.event.dom.client.ClickEvent;
import com.google.gwt.event.dom.client.ClickHandler;
import com.google.gwt.user.client.ui.Button;
import com.google.gwt.user.client.ui.HasHorizontalAlignment;
import com.google.gwt.user.client.ui.HorizontalPanel;
import com.google.gwt.user.client.ui.RootPanel;
```

```
import org.Aptech.client.rpc.DemoRPCServiceAsync;
import org.Aptech.client.rpc.RPCMessageCallBack;
/**
 * Entry Point class for sample RPC
 *
 */
public class RPCEntryPoint implements EntryPoint{
        private DemoRPCServiceAsync itsMessageService = GWT.
create(DemoRPCService.class);

    @Override
    public void onModuleLoad()    {
        Button aButton = new Button("Test RPC!!!");

        HorizontalPanel aVPanel = new HorizontalPanel();
        aVPanel.add(aButton);

        RootPanel.get().add(aVPanel);

        aButton.addClickHandler(new ClickHandler(){
            @Override
            public void onClick(ClickEvent event){
     itsMessageService.getServerMessage(new RPCMessageCallBack());
            }
        });
    }
}
```

Code Snippet 16 demonstrates the module descriptor file.

RPC.gwt.xml

Code Snippet 16:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE module PUBLIC "-//Google Inc.//DTD Google Web Toolkit
2.0.0//EN"          "http://google-Web-toolkit.googlecode.com/svn/
tags/2.0.0/distro-source/core/src/gwt-module.dtd">
<module>
<!-- Inherit the core Web Toolkit stuff.  -->
<inherits name='com.google.gwt.user.User' />

<!-- Inherit the default GWT style sheet.  You can change       -->
<!-- the theme of your GWT application by uncommenting          -->
<!-- any one of the following lines.    -->
<inherits name='com.google.gwt.user.theme.standard.Standard' />
```

```
<!--  <inherits  name='com.google.gwt.user.theme.chrome.Chrome'/>
-->
<!-- <inherits name='com.google.gwt.user.theme.dark.Dark'/>    -->

<!-- Other module inherits  -->

<!-- Specify the app entry point class.  -->
     <entry-point class="org.Aptech.client.RPCEntryPoint"/>
</module>
```

Code Snippet 17 demonstrates the RPCTest.html file.

RPCTest.html

Code Snippet 17:

```
<!DOCTYPE html>
<html>
    <head>
        <title>RPC Test</title>
        <meta charset="UTF-8">
      <meta name="viewport" content="width=device-width, initial-
scale=1.0">
        <script type="text/JavaScript"  src="org.Aptech.RPC/org.
Aptech.RPC.nocache.js"></script>
    </head>
    <body>
        <div></div>
    </body>
</html>
```

Code Snippet 18 demonstrates the deployment descriptor.

web.xml

Following entry needs to be added in web.xml, to add reference to the newly created servlet:

Code Snippet 18:

```
<servlet-mapping>
     <servlet-name>DemoRPCServiceImpl</servlet-name>
     <url-pattern>/org.Aptech.RPC/rpcdemo</url-pattern>
</servlet-mapping>
<servlet>
  <servlet-name>DemoRPCServiceImpl</servlet-name>
 <servlet-class> org.Aptech.client.rpc.server.DemoRPCServiceImpl</
servlet-class>
</servlet>
```

Figure 5.17 shows the output of the code when the call is successfully executed.

Figure 5.17: Output

Tip 1: To test the failure case, open the html page and then, stop the server from NetBeans IDE to check if RPC is working. Since the server is down, the call will fail.

Tip 2: Since the call for RPC is through AJAX, one can use **Developer Tools** or **Firebug** to test the call.

5.2 Google Maps

Nowadays, everyone uses Google Maps off and on to find the location on the map. It allows to view the earth's satellite image and is commonly used for navigation. Google provides the API that allows a developer to use the map in an application.

All one has to do is to include the Google's JavaScript to the page. The library has some pre-defined objects. The user needs to specify to those objects which place to show and how to present the map as well as where to show the map in the page. The library that is included performs the tasks for the developer. Following library needs to be included in the page: http://maps.googleapis.com/maps/api/js

5.2.1 Map Options

The Map Options need to be set to specify the location and display details for the map. Table 5.2 lists few basic Map Options.

Option Key	Type	Description
Center	LatLng	LatLng, that is, latitude and longitude, is a data type that is included in the API. This option specifies which point should be the centre of the Map.
keyboardShortcuts	Boolean	Indicates if the Key-board shortcut needs to be enabled or not.

Option Key	Type	Description
Zoom	Number	The initial default zoom level when the map is loaded.
zoomControl	Boolean	Indicates if the zoom control is needed or not.
zoomControlOptions	zoomControlOptions	Indicates how control options for zoom should be.
Maptypeid	MapTypeId	Specifies how the default Map type should be. The map can be Hybrid, Road map, Satellite, and Terrain.

Table 5.2: Option Key

There are approximates 30 more Options on how the map should be displayed.

5.2.2 Longitude and Latitudes

Longitude and Latitudes are the numbers which denotes a particular point in the earth. They are similar to the **X** and **Y** co-ordinates that denote a point on a 2-D graph. The latitudes and longitudes are imaginary, but standard. That is, Latitude and Longitude of a particular point remains the same always.

To get the Latitude and Longitude of a particular point,

- Launch Google Maps (https://maps.google.com).
- Search for the place of interest.
- Right–click the location and click 'What's here?'. The latitude and longitude will appear in the top–left corner of the page as shown in figure 5.18.

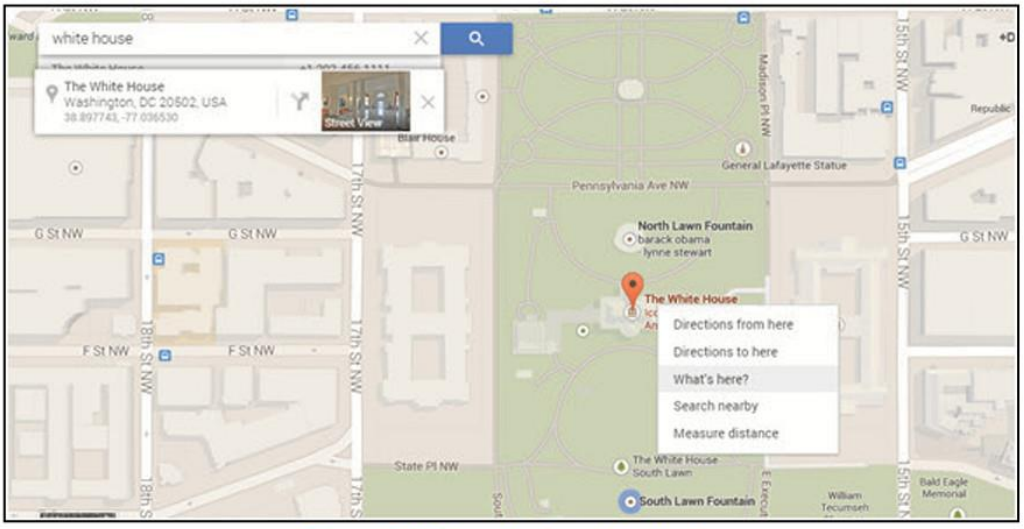


Figure 5.18: Longitude and Latitude

5.2.3 Map Object

The instance of google.maps.Map represents a Map. So whenever a map is needed, the instance of google.maps.Map must be created as follows:

```
new google.maps.Map(document.getElementById("divToContainMap"),
aVariableWithMapOptions);
```

Code Snippet 19 demonstrates the use of a map in a div of dimension 400X400 with the centre of the map as White House.

WhiteHouse.html

Code Snippet 19:

```
<!DOCTYPE html>
<html>
<head>
<script src="http://maps.googleapis.com/maps/api/js"></script>
<script>
function init(){
  // Give the Latitude and longitude of White House and the initial
zoom level
  var aMapPosition = {
     center:new google.maps.LatLng(38.897743, -77.036530),
    zoom:18
  };
// The map will be displayed in the div with id googleMap
  new google.maps.Map(document.getElementById("googleMap"),aMapPos
ition);
}

// The call to show the map will happen on window load.
google.maps.event.addDomListener(window, 'load', init);
</script>
</head>
<body>
 <div id="googleMap" style="width:400px;height:400px;"></div>
</body>
</html>
```

Figure 5.19 shows the output pointing to the location on the map based on the latitude and longitude specified in the code .

Session 05

Google Web Toolkit

Figure 5.19: Output

5.3 Google AJAX Search API

The Google AJAX Search API allows including Google Search in Web pages with JavaScript. That is, one can embed a simple, dynamic search box, and display search results in Web pages or use the results in innovative and programmatic ways.

The Google AJAX Search API allows developers to integrate Web Search, News Search, and Blog Search into their Web site. The developers can add Local Search results to the Web site, or integrate them with their Google Maps API mashup. It also allows adding YouTube Videos and Google Image Search results to a Web site or blog. This API is now deprecated and the functionality of this API is now covered by the Google Custom Search API.

Developers can now exploit the full power of ideas to dynamically generate Custom Search Engines. One can host the CSE specification on the Web site and include the url for this specification in the CSE search request. Google retrieves the CSE specification from the Website when a user searches in the CSE.

There are two editions of Google Custom Search:

Session 05

Google Web Toolkit

- **Custom Search Engine (basic edition)**

  With Google Custom Search, developers can add a search box to the homepage to help people find required content on their Website.

- **Google Site Search (business edition)**

  Google Site Search includes the same search technology that is applied in Google.com in the developer's Website, to display relevant results with lightning speed.

Session 05

Google Web Toolkit

Check Your Progress

1. _____ is a Google's development kit that allows Java developers to develop rich applications without much expertise on client-side technologies such as JavaScript and CSS.

(A)	Google Web Toolkit (GWT)	(C)	Google drive
(B)	Google Maps	(D)	JavaScript

2. _____ pattern is a solution whenever there is a scenario in which an instance needs to inform one or more instances about an event.

(A)	Listener design	(C)	JQuery
(B)	Framework	(D)	Robust

3. GWT _____ is a mechanism that allows calling the server method from the client code, asynchronously.

(A)	RemoteServiceServlet	(C)	Remote Procedure Calls
(B)	POJO	(D)	AJAX call

4. Since the call for RPC is through AJAX, one can use _____ to test the call.

(A)	Google's JavaScript	(C)	Programming language
(B)	API	(D)	Firebug

5. Longitude and Latitude are the numbers which denote a particular point on the earth. They are similar to the X and Y co-ordinates that denote a point in a _____.

(A)	4-D dimensions	(C)	3-D graph
(B)	2-D graph	(D)	Launch

Session 05

Google Web Toolkit

Answers

1.	A
2.	A
3.	C
4.	D
5.	B

Session 05

Google Web Toolkit

Summary

- The GWT compiler translates Java code into JavaScript/CSS/HTML as applicable.
- Any GWT application must have atleast one class that implements com.google.gwt.core.client.EntryPoint.
- A module is an entity that contains one or more Entry Points. A GWT application can be said as a collection of one or more modules.
- There must be some HTML or other server pages that include the scripts and hook-up those GWT functionalities into the application.
- Web pages generated by GWT cannot be indexed by search engines because these applications are generated dynamically.
- GWT Events are similar to the events in other Java UI frameworks such as AWT or Swing.

06 AJAX with JavaServer Faces (JSF) and Struts

Welcome to the Session, **AJAX with JavaServer Faces (JSF) and Struts**.

This session explains about using AJAX with technologies such as JavaServer Faces (JSF) and Struts 2. It describes the new features of the frameworks that can be used to implement AJAX in the Web applications.

In this Session, you will learn to:

- Explain the features of Struts 2
- Use AJAX with Struts 2
- Describe the features of JSF
- Use AJAX with JSF

Session 06

AJAX with JavaServer Faces (JSF) and Struts

6.1 Introduction

Two of the most widely used Web frameworks are Struts and JSF. However, both frameworks send and receive data synchronously. Therefore, while the server processes the user's request, the user cannot interact with the Web page. AJAX gives these frameworks the ability to send and receive data asynchronously. Thus, the user can interact with the Web page while the server is processing the request.

To enable AJAX in a Web application, JavaScript code is used to send AJAX request. This code is written in a JSP page. A server-side component, such as a servlet processes the AJAX request.

Today developers prefer using Web frameworks such as Struts and JSF to process AJAX requests. This is because these frameworks provide a higher-level of abstraction above the servlet API. Moreover, AJAX requests are similar to HTTP requests. The only difference is that an AJAX request is sent asynchronously. Therefore, using a Web framework with AJAX will allow taking the benefits of both, the framework and AJAX. For example, AJAX will enable improvement of the response time and provide rich look and feel to Struts and JSF applications.

6.2 AJAX in Struts Application

Consider a Struts or a JSF Web application that allows downloading articles from a Web site only after the user is registered. To become a registered user, creation of an account is required. The Web application displays the account registration page. The registration page contains text boxes to accept username and password, and a Submit button. When a user enters the details and clicks the Submit button, the page waits for a response from the Web server. If the username already exists, the user will need to repeat the process of filling details in the registration page. This is because Struts and JSF follow the click-wait-refresh cycle.

If AJAX based applications are used, the users can be notified about the validation of the username and password as soon as the text box looses focus.

6.2.1 Features of Struts 2

Struts 2 is a popular Web application framework and a complete rewrite of Struts 1. It is based on the MVC design pattern.

To simplify Web development for the developers, the WebWork framework began with Struts framework to offer an improved and enhanced framework built on Struts.

After some time, the Struts and the WebWork framework community together created the Struts 2 framework.

Session 06

AJAX with JavaServer Faces (JSF) and Struts

The features of Struts 2 are as follows:

- **Support for AJAX** - Struts 2 has integrated AJAX support by creating AJAX tags into the product, that function in the same manner as the standard Struts 2 tags.
- **Integration simplified** - Struts 2 provides integration with other frameworks such as Tiles, Spring, and SiteMesh.
- **Tag Modification made easy** - In Struts 2, using Freemarker templates, tag markups can be tweaked. There is no requirement of Java or JSP knowledge. Basic XML, HTML, and CSS knowledge is sufficient to modify the tags.
- **Plugin support** - The behavior of core Struts 2 can be augmented and enhanced by using plugins. Several plugins are available for Struts 2.
- **POJO forms and actions** - With Struts 2, POJO can be used to receive the form input. Similarly, POJO can be used as an Action class.
- **Profiling** - Integrated profiling for the application is offered by Struts 2. Also, integrated debugging is offered in Struts 2 with the help of built in debugging tools.
- **Promote less configuration** - Less configuration is required in Struts 2 as it uses default values for various settings. Configuring something is not required in Struts 2 unless it deviates from the default settings used by Struts 2.
- **Tag support** - The form tags are improved in Struts 2 and thus, the developers are allowed to write less code by using these tags.
- **Template Support** - Generating views is supported in Struts 2 using templates.
- **View Technologies** - Multiple view options such as JSP, Velocity, Freemarker, and XSLT are supported by Struts 2.

6.2.2 AJAX in Struts 2

To add the Struts 2 plugin, perform the following steps:

1. Download the Struts 2 plugin from http://sourceforge.net/projects/struts2nbplugin and install it using Tools → Plugins in NetBeans IDE 8.0.

2. Create a Web application and use the Struts 2 Framework for it as shown in figure 6.1.

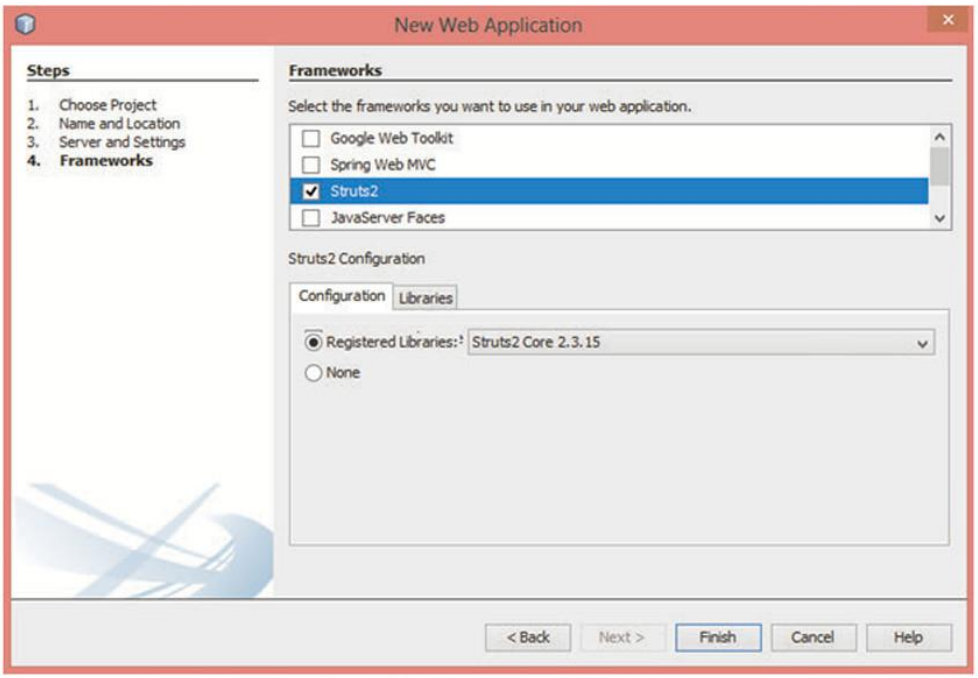


Figure 6.1: Frameworks Screen

AJAX Implementation in Struts 2 Using JQuery and JSON

Following files need to be created for the application:

- **web.xml**

  The configuration settings, filter-name, filter-class, and mapping details are specified here. Code Snippet 1 demonstrates the web.xml file.

Code Snippet 1:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app   version="3.1"   xmlns="http://xmlns.jcp.org/xml/ns/
javaee"  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee http://
xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd">
    <filter>
        <filter-name>struts2</filter-name>
       <filter-class>
      org.apache.struts2.dispatcher.ng.filter.
StrutsPrepareAndExecuteFilter</filter-class>
    </filter>
```

```
    <filter-mapping>
        <filter-name>struts2</filter-name>
       <url-pattern>/*</url-pattern>
    </filter-mapping>
    <session-config>
        <session-timeout>
            30
        </session-timeout>
    </session-config>
    <welcome-file-list>
        <welcome-file>Registration.jsp</welcome-file>
    </welcome-file-list>
</web-app>
```

- **struts.xml**

Code Snippet 2 depicts the struts.xml file that connects the html code and Java code by using action tags.

Code Snippet 2:

```
<!DOCTYPE struts PUBLIC
"-//Apache Software Foundation//DTD Struts Configuration 2.0//
EN"
"http://struts.apache.org/dtds/struts-2.0.dtd">
<struts>
    <!-- Configuration for the default package. -->
      <package  name="default"  extends="struts-default,json-
default">
         <action name="blurAction" class="ajax.TestAjaxCall"
method="check">
            <result type="json"></result>
        </action>
    </package>
</struts>
```

The action named 'blurAction' is called and the TestAjaxCall class is bounded with the action.

The output received is in the form of JSON so it is required to add the Struts2-JSON jar file. Download the struts2-JSON plugin from http://mvnrepository.com/artifact/org.apache.struts/struts2-json-plugin/2.3.16.3 and add it to the Libraries folder in the project.

- **Registration.jsp**

The Registration.jsp page takes input from the user and if the username already exists, it displays a message to the user in the span tag. The message is retrieved from the TestAjaxCall.java class.

Code Snippet 3 demonstrates the Registration.jsp page.

Code Snippet 3:

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>
<%@taglib prefix="s" uri="/struts-tags"%>
<!DOCTYPE html>
<html>
   <head>
          <meta http-equiv="Content-Type" content="text/html;
charset=UTF-8">
        <title>JSP Page</title>
        <script type="text/javascript" src="jquery-1.11.1.js">
</script>
    <script type="text/javascript">
        $(document).ready(function(){
            $(".requiredF").blur(function(){
                $.ajax({type: "POST",
                    url: 'blurAction',
                    data: {
                       uname : $("#username").val(),
                        pwd : $("#password").val()
                    },
             success: function(data){
            console.log(data.message);
  document.getElementById("message").innerHTML=data.message;
                        $("#username").text="";
                        $("#password").text="";
                    },
              error: function(error){
                        console.log(error);
                    }
                });
            });
        });
    </script>
    </head>
<body>
  <s:form action="registerAction" theme="simple" method="post"
validate="true">
            <table>
                <tr>
                    <td>
                     UserName:<s:textfield cssClass="requiredF"
name="username" id="username" label="Username"></s:textfield>
<span id="message" style="color:red"></span><br/>   </td>
```

```
                </tr>
                <tr>
                    <td>
             Password:  <s:textfield cssClass="requiredF"
name="password" id="password" label="Password" />
                     </td>
                </tr>
                <tr>
                    <td>
                <input type="button" id="btn" value="Submit">
                    </td>
                </tr>
            </table>
        </s:form>
    </body>
</html>
```

Note that to use struts tags, the statement <%@taglib prefix="s" uri="/struts-tags"%> has been added. Also, jQuery code has been used. Hence, it is required to paste the .js library in the Web Pages folder. Here, jquery-1.11.1.js has been used.

Whenever there is blur event for the 'requiredF' class, the $.ajax method is called. The data passed to the server are username and password values. The url 'blurAction' is mentioned in the struts.xml file and the check method of TestAjaxCall.java class associated with it is invoked. Based on the value of the username, the 'message' property is set and can be accessed using Struts.

The success and error functions are specified which are executed based on the response.

Note - The blur event is called when the text box loses focus.

- **TestAjaxCall.java**

The TestAjaxCall.java class consists of the 'check' method which is called for verifying the username. The class is created under a package named 'ajax' as demonstrated in Code Snippet 4.

Code Snippet 4:

```
package ajax;
import static com.opensymphony.xwork2.Action.SUCCESS;
public class TestAjaxCall {
    private String uname;
    private String pwd;
private String message;
```

```
        public String check() throws Exception {
         String[] unames= new String[4];
         unames[0]="jsmith";
         unames[1]="ashandrews";
         unames[2]="johnsmith";
         unames[3]="maryjoe";
         for (String name : unames) {
            if(name.equalsIgnoreCase(uname)){
              message= "Username already exists";
              break;
            }
            else
               message= "Available";
       }
     return SUCCESS;
    }
    public String getMessage() {
        return message;
    }
    public void setMessage(String message) {
        this.message = message;
    }
        public String getUname() {
        return uname;
    }
   public void setUname(String uname) {
        this.uname = uname;
    }
    public String getPwd() {
        return pwd;
    }
    public void setPwd(String pwd) {
        this.pwd = pwd;
    }
}
```

The 'check' method is called through AJAX and Struts 2 framework. It checks for the contents in the username field and returns the message accordingly. The message is directly accessed in the Registration.jsp page.

The output for the Registration.jsp is as shown in figure 6.2.

Figure 6.2: Output of Registration.jsp Page

When the user enters a username which already exists, the output of onblur event after entering the data is as shown in figure 6.3.

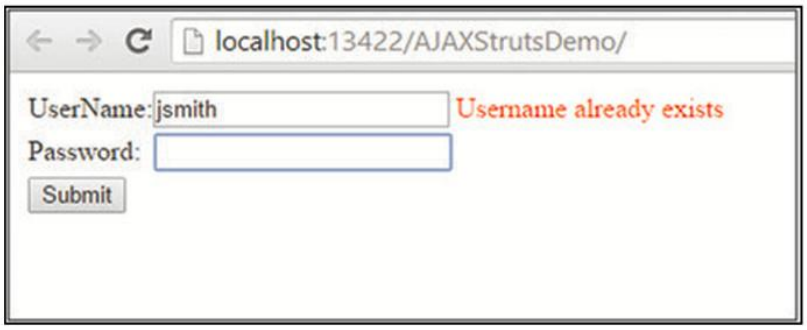


Figure 6.3: Output After onblur Event

When the blur event is called on the username text box, the request is sent to blurAction class where the 'check' method is invoked from the associated TestAjaxCall. java class. As the username specified in the text box already exists, the message 'Username already exists' is displayed.

The output when an unused username is specified is as shown in figure 6.4.

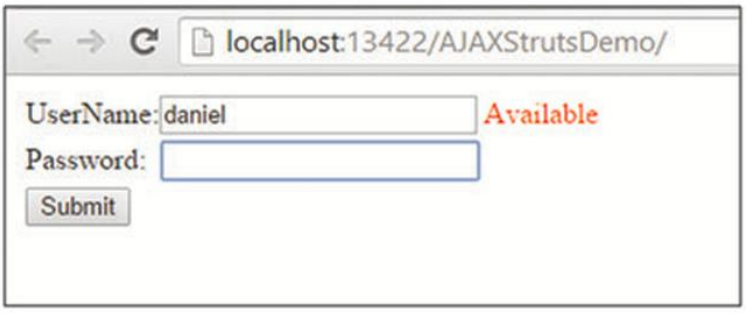


Figure 6.4: Output for Another Username

Since the username has not been used yet, the message 'Available' is displayed. Thus, the user need to go through the click-wait-refresh cycle as the message gets displayed asynchronously as soon as the text box loses focus.

Session 06

AJAX with JavaServer Faces (JSF) and Struts

6.3 JSF and AJAX

JSF is a component UI framework. It provides a rich set of UI components, validators, and converters to create Web applications. However, JSF also follows the click-wait-refresh cycle. In other words, JSF pages do not allow interaction with the page until the response is received from the server.

AJAX can be used in JSF applications to send and receive data asynchronously. There are several approaches to include AJAX functionality in JSF applications. The easiest approach is to include the AJAX JavaScript code in JSF pages and use a servlet or a phase listener to process the AJAX request. However, this requires the page author to have sound knowledge of JavaScript. JSF provides the option of creating custom components. All the JavaScript code can be included in the custom component. The page author only needs to know how to use the custom component in the page.

6.3.1 New Features in JSF 2.0 and 2.2

Following are the new features in JSF 2.0:

- **AJAX support within tags** – This feature allows JSF pages to directly communicate with the server without having to go through the full-page refresh of the browser.
- **Annotations** may remove the need for XML. This is because a new Java language annotation is now available for almost every XML element appearing in the faces-config.xml file. These features of annotation and navigation without XML navigation rules, will allow JSF applications to be built entirely without using the faces-config.xml file.
- **Composite components** – It enables creating JSF components by aggregations of other JSF components. It is easy to build custom JSF components and to refactor existing views into reusable components, complete with attributes, listeners, and events.
- **Partial State Saving** –The amount of memory consumed by JSF was the major issue before, but it is solved by maintaining the view state between requests. This has drastically reduced the amount of memory utilized. The API for handling state also provides easy way to implement this, while developing custom components.
- **View Parameters** –JSF insists on using POST for all inter-page navigations is the other complaint against JSF. However, with JSF 2.0, one can use View Parameters to enable the page to accept parameters even on an initial GET request, or on a page transition request, and at the same time preserve the JSF conversion and validation model. It is possible to include view parameters using implicit navigation as well as in the XML navigation rules.
- **Exception Handling** – JSF 2.0 has a central ExceptionHandler. With this, an error page that uses JSF components can be easily constructed.
- **Expression Language (EL) Enhancements** –The EL supports method invocation on arbitrary Java methods and several new implicit objects have being introduced.

Also, parameter passing is allowed.

- **Validation** – A new Java specification has been developed for validation known as JSR-303 Bean Validation and its works fine with JSF 2.0.
- **FacesContext** -During application startup and shutdown, **FacesContext** access is now available.
- **New scopes** – JSF 2.0 also provides a mechanism to define custom scopes.

Following are the new features in JSF 2.2:

- Stateless Views for improved performance.
- HTML5 friendly markup passed straight to the browser. JSF Renderer handles decode from browser.
- JavaServer Faces has a new bean scope that is wider than request scope, narrower than session scope and different from view scope: Flow Scope. Faces flows helps to modularize behavior. It is built on navigation concepts. Navigation is no longer just between pages but between flow 'nodes'. Multiple node types include View, Method call, Switch, Flow call, and Flow return.
- Resource Library Contracts to modularize appearance. It is built on facelets concepts. The faces-config.xml file controls which parts of the application are allowed to use which contracts.

6.3.2 Custom Component

The first step in enabling AJAX in a JSF application is to create a custom component or a custom tag. To create a JSF custom component, create or use the following files:

- **UI Component Class** – contains the core logic of the component. This class is derived from either UIInput to create an input component, UIOutput to display static text, or UICommand class to create an event-based component.
- **Renderer Class** – contains the code to render the markup of the component. Usually, most simple custom components include the rendering code in the UI Component class itself.
- **UI Component Tag Class** – is the tag handler class and associates the custom tag with the component and renderer classes.
- **Tag Library Descriptor File** – describes the usage of custom component in JSP pages and associates the tag in JSP page with the UI Component tag class.
- **faces-config.xml** – specifies the name of custom component and the renderer.

Figure 6.5 shows the JSF components.

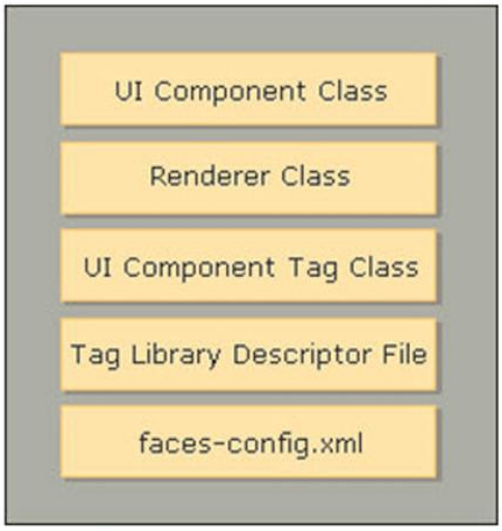


Figure 6.5: JSF components

6.3.3 Using AJAX with JSF

To use AJAX with JSF, perform the following steps:

1. Create a Web Application with the JSF (2.0 or 2.2) framework. Figure 6.6 shows the use of JavaServer Faces framework.

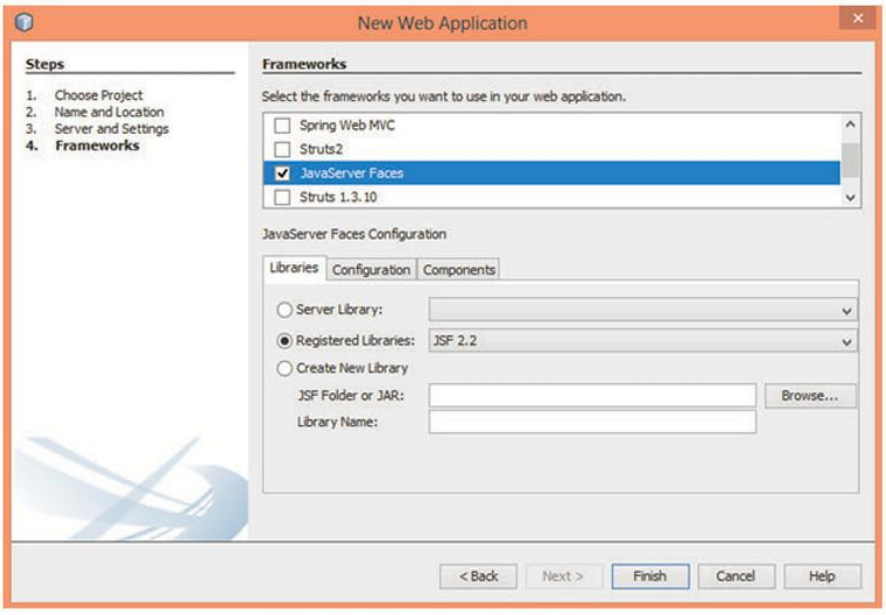


Figure 6.6: Framework Window

The files used in the JSF projects are as follows:

- **index.xhtml**

Code Snippet 5 demonstrates the index.xhtml JSF page.

Code Snippet 5:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:h="http://xmlns.jcp.org/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      >
   <h:head>
        <title>Facelet Title</title>
    </h:head>
    <h:body>
        <h1>JSF with AJAX</h1>
    <h:form>
            <h:panelGrid columns="2">
                Select a country:
             <h:selectOneMenu value="#{country.localeCode}" >
                            <f:selectItems  value="#{country.
countryInMap}" />
                 <f:ajax event="change" listener="#{country.
countryLocaleCodeChanged()}" render="countryname" />
                </h:selectOneMenu>
                Selected country is:
             <h:outputText id="countryname" value="#{country.
message}" >
                </h:outputText>
            </h:panelGrid>
        </h:form>
    </h:body>
</html>
```

The index.xhtml page includes a drop-down which will be filled with the names of countries from a JSF Managed Bean named CountryBean. <h:ajax> is the tag used for implementing AJAX functionality in JSF. The attributes event, listener, and render are used for specifying the event on which the action is to be triggered, the handler function in the Managed Bean class, and the class or control which is to be rendered with the output respectively. The tags used are JSF tags. Here, the outputText tag with the id 'countryname' will be rendered with the response received from the Managed Bean.

CountryBean.java

Code Snippet 6 demonstrates the JSF Managed Bean named CountryBean. Note that while creating the bean, the scope should be set to Session.

Code Snippet 6:

```
package com;
import javax.inject.Named;
import javax.enterprise.context.SessionScoped;
import java.io.Serializable;
import java.util.LinkedHashMap;
import java.util.Map;
import javax.faces.bean.ManagedBean;
import javax.faces.event.ValueChangeEvent;
/**
 *
 * @author Dhrutis
 */
@ManagedBean(name = "country")
@SessionScoped
public class CountryBean implements Serializable {
    private static Map<String, String> countries;
    private String message;
   private String localeCode = "en"; //default value
    static {
        countries = new LinkedHashMap<String, String>();
       countries.put("United Kingdom", "en"); //label, value
        countries.put("French", "fr");
        countries.put("German", "de");
        countries.put("China", "zh_CN");
    }
   public void countryLocaleCodeChanged() {
//        localeCode = e.getNewValue().toString();
        if (localeCode.equals("en")) {
           message = "United Kingdom";
        } else if (localeCode.equals("fr")) {
            message = "France";
        } else if (localeCode.equals("de")) {
            message = "Germany";
        } else {
            message = "China";
        }
    }
    public Map<String, String> getCountryInMap() {
        return countries;
    }
```

```
    public String getLocaleCode() {
        return localeCode;
    }
    public void setLocaleCode(String localeCode) {
        this.localeCode = localeCode;
    }
    public String getMessage() {
        return message;
    }
}
```

The Code Snippet uses the annotations @ManagedBean and @SessionScoped which indicates that the class is a Managed Bean with session scope. The CountryBean class can be accessed by the name 'country' in the index.xhtml file.

The method countryLocaleCodeChanged is accessed on the change event of the selectOneMenu component. In the method, the property 'message' is set according to the value selected in the selectOneMenu.

Figure 6.7 shows the output on page load.

Figure 6.7: Output on Page Load

On selecting a country from the drop-down menu, an asynchronous request is sent to the Managed Bean that returns the appropriate country name. The name is displayed in the outputText tag as shown in figure 6.8.

Figure 6.8: Selecting a Country from the Drop-down Menu

Session 06

AJAX with JavaServer Faces (JSF) and Struts

Check Your Progress

1. Match the files and/or classes required to create custom component in JSF with their corresponding descriptions.

Description		File/Class	
(a)	Renders the component's markup.	(1)	UI Component
(b)	Describes the usage of the custom component.	(2)	faces-config.xml
(c)	Contains core logic of the custom component.	(3)	Tag Class
(d)	Specifies the name of the custom component and renderer.	(4)	Renderer
(e)	Associates the custom tag, component and the renderer.	(5)	Tag Library Descriptor

(A)	a-2, b-5, c-1, d-3, e-4	(C)	a-3, b-5, c-1, d-2, e-4
(B)	a-4, b-5, c-1, d-2, e-3	(D)	a-5, b-4, c-3, d-2, e-1

2. Which of the following statements about AJAX and Web frameworks are true?

(a)	Struts and JSF use servlets to send HTTP requests.
(b)	AJAX-based applications can respond faster.
(c)	Web frameworks such as Struts and JSF provide an abstraction layer above the servlet API.
(d)	Web applications write AJAX code in JSP pages.
(e)	Web frameworks such as Struts and JSF improve the response time of Web applications.

(A)	a, c, d	(C)	b, d, e
(B)	b, c, d	(D)	c, d, e

Session 06

AJAX with JavaServer Faces (JSF) and Struts

Check Your Progress

3. Match the following features of JSF with the corresponding advantages.

Advantage		Feature	
(a)	It is easy to build custom JSF components and refactor existing views into reusable components	(1)	Exception Handling
(b)	During application startup and shutdown, FacesContext access is now available.	(2)	Validation
(c)	An error page that uses JSF components can be easily constructed.	(3)	Annotations
(d)	A Java specification known as JSR-303 has been developed for validation.	(4)	Composite components
(e)	Navigation is possible even without XML configuration.	(5)	FacesContext

(A)	a-2, b-5, c-1, d-3, e-4	(C)	a-4, b-5, c-1, d-2, e-3
(B)	a-3, b-5, c-1, d-2, e-4	(D)	a-5, b-4, c-3, d-2, e-1

4. Which of the following statements about Struts 2 features are true?

(a)	Struts 2 has integrated AJAX support by creating AJAX tags into the product.
(b)	Struts 2 does not support integration with other frameworks such as Tiles, Spring, and SiteMesh.
(c)	With Struts 2, POJO can be used to receive the form input.
(d)	Generating views is supported in Struts 2 using templates.

(A)	a, b, c	(C)	a, b, d
(B)	b, c, d	(D)	a, c, d

Session 06

AJAX with JavaServer Faces (JSF) and Struts

Check Your Progress

5. Struts and the _____ framework community together created the Struts 2 framework.

(A)	WebShine	(C)	WebMaster
(B)	WebWork	(D)	WebZone

Session 06

AJAX with JavaServer Faces (JSF) and Struts

Answers

1.	B
2.	B
3.	C
4.	D
5.	B

Session 06

AJAX with JavaServer Faces (JSF) and Struts

Summary

- Web frameworks such as Struts and JSF are used to process AJAX requests as they provide higher-level of abstraction above the servlet API.
- AJAX enables improvement of the response time and provides rich look and feel to Struts and JSF applications.
- Struts 2 is a complete rewrite of the Struts architecture and not just next version of Struts 1.
- Struts 2 uses a struts.xml file for configuration of the JSP page and the Action class.
- JSF is a component UI framework with rich set of UI components, validators, and converters to create Web applications.
- Navigation in JSF 2.2 can be done without using the faces-config.xml, due to the annotations provided by it.