AJAX for Java Web Applications

Session: 3

AJAX Client Frameworks-I

Objectives

- 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

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.

- For example: Google Web Toolkit (GWT)

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, exception and error handling, and a lot of extra features.

- For example: Prototype, DOJO Toolkit, Rico, Script.aculo.us

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.

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

Advantages of Client-Side Frameworks

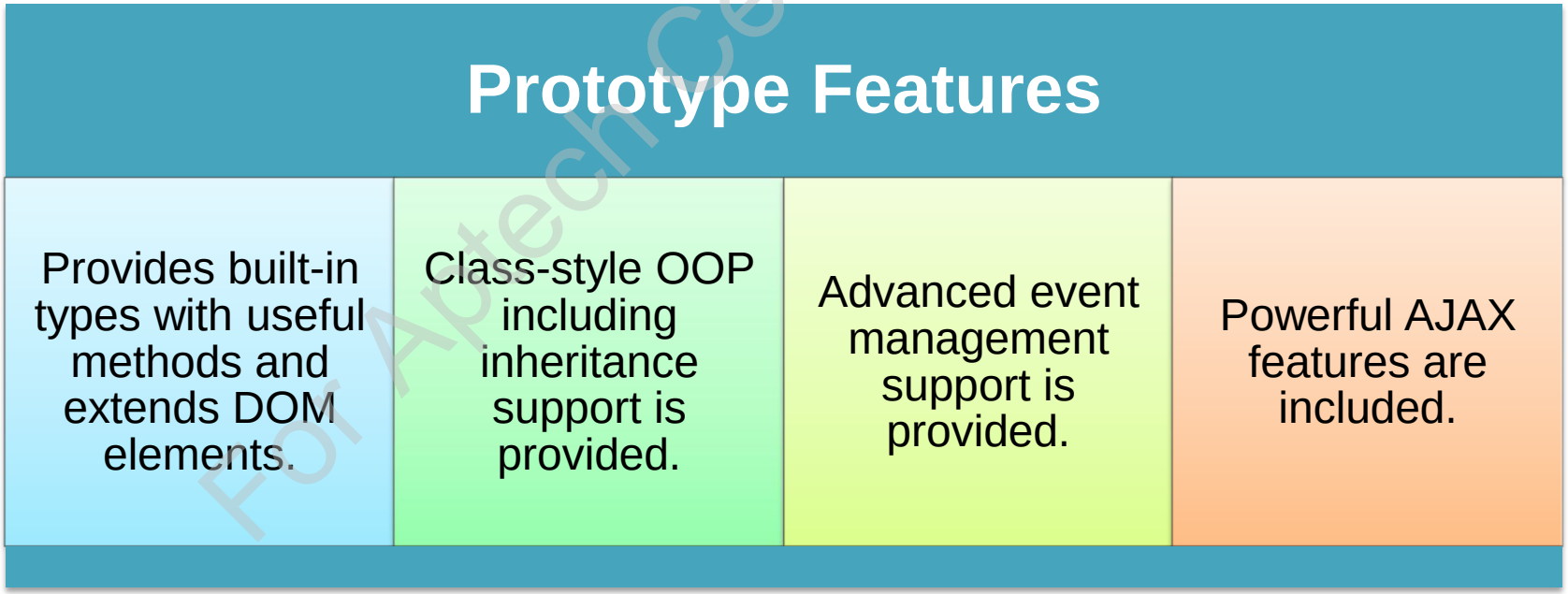
Complete flexibility of development.

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 using AJAX components with drag and drop style.

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.



AJAX Response Callbacks 1-2

- 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 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]);`

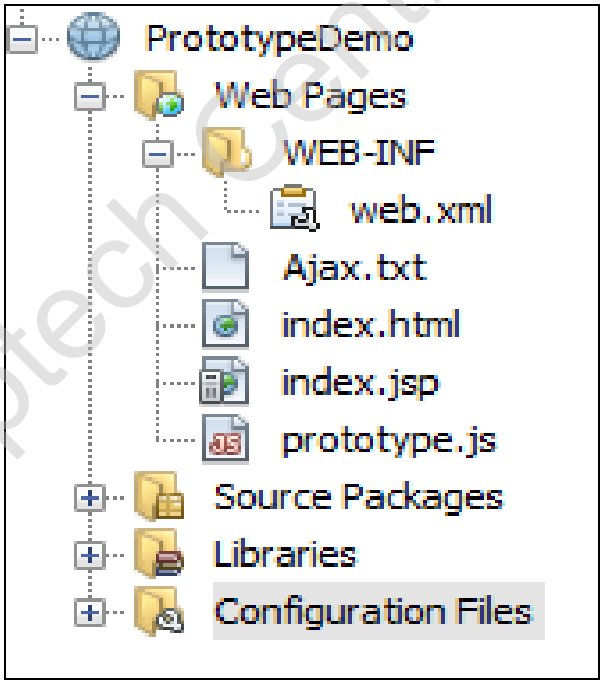
- The request is initiated as soon as the object is created, then it processes throughout its life-cycle. The life-cycle has the following states:

AJAX Response Callbacks 2-2

- Callback functions for Prototype's 'Ajax'.
  - onCreate
  - onUninitialized
  - onLoading
  - onLoaded
  - onInteractive
  - onXYZ
  - onComplete
- Some common options of the callbacks.
  - asynchronous
  - contentType
  - encoding
  - method
  - parameters
  - postBody
  - requestHeaders
  - evalJS
  - evalJSON
  - sanitizeJSON

Steps to Use Prototype Framework 1-3

1. • Download the prototype JavaScript from https://ajax.googleapis.com/ajax/libs/prototype/1.7.2.0/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 the following figure:



Steps to Use Prototype Framework 2-3

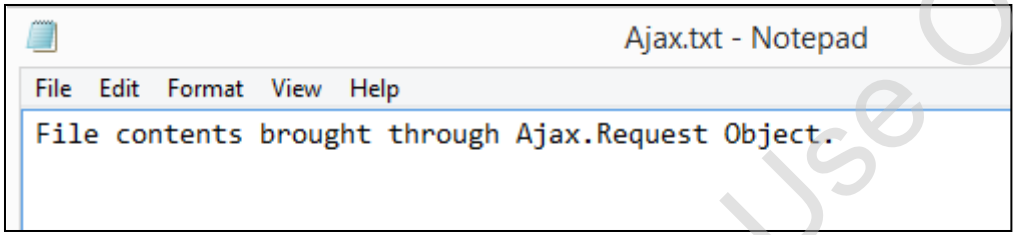
4. • For using the prototype.js include the following code in the JSP page:
   • `<script type="text/javascript" src="prototype.js"/>`

- Consider a Web page where on click of a button, the data from a file placed on the server is retrieved asynchrounously using Ajax.Request object.
- Following Code Snippet demonstrates the code of index.jsp file using the Ajax.Request object:

```
...
<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) {
        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>
...
```

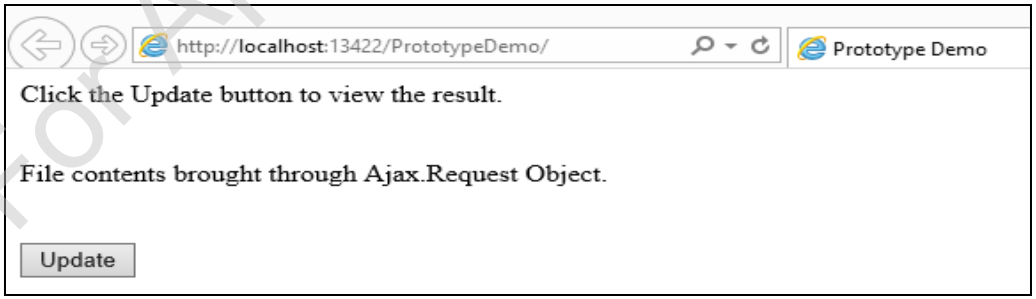
Steps to Use Prototype Framework 3-3

- Following figure shows the contents of the Ajax.txt file:



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

- On click of the Update button, the text in the tag changes and the file contents are brought asynchronously as shown in the following figure:



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.

insertion(string)

- By default, the whole content of a container is updated.

- Following Code Snippet demonstrates the use of Ajax.Updater:

```
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.

- **Methods of Ajax.Response Object**
  - getHeaderName()
  - getAllHeaders()
  - getReponseHeader(name)
  - getAllReponseHeader()

- **Properties of Ajax.Response Object**
  - status
  - statusText
  - readyState
  - reponseText
  - reponseXML
  - reponseJSON
  - headerJSON
  - Request
  - Transport

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 the following table:

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

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.
- 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 ] )`

- 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"/>
```

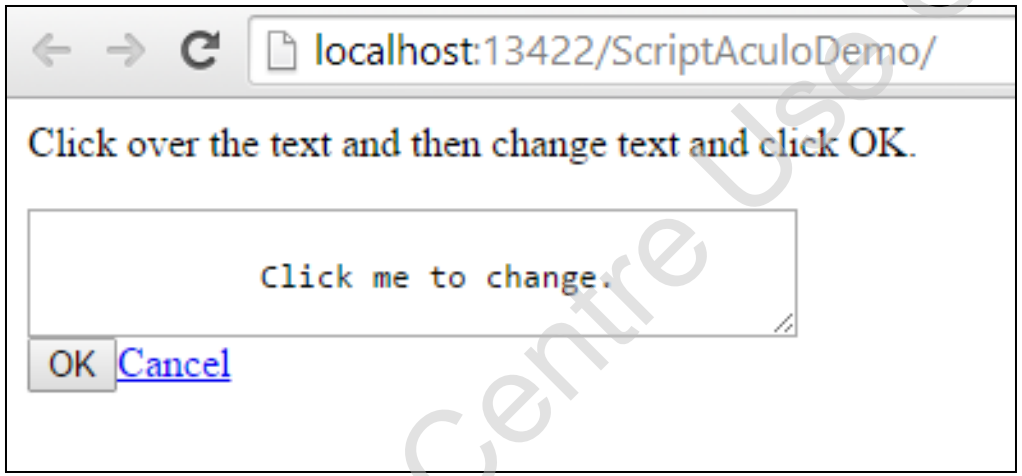
Using Ajax.InPlaceEditor() 1-2

- Consider a Web page which gets updated with the contents based on updates in the transform.php file.
- Following Code Snippet demonstrates the use of Ajax.InPlaceEditor().
- The file name is index.jsp.

```
...
<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=ef
fects,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>
...
```

Using Ajax.InPlaceEditor() 2-2

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



- Specify the text and click OK. The php script changes the text to upper case and displays the modified text. The content of the transform.php script is as follows:

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

Ajax.AutoCompleter() and Ajax.InPlaceCollectionEditor()

- **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);
```

- **Ajax.InPlaceCollectionEditor()**
  - The contents of the select box are changed on the fly.
  - **Syntax**:
```
new Ajax.InPlaceCollectionEditor(
element, url, { collection: [array],
[moreOptions] } );
```

Effects 1-3

- 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
  - Effect.Scale
  - Effect.Move
  - Effect.Highlight
  - Effect.Morph
  - Effect.Parallel
- For example, consider a Web page where on clicking the page the background color and the other properties get changed.

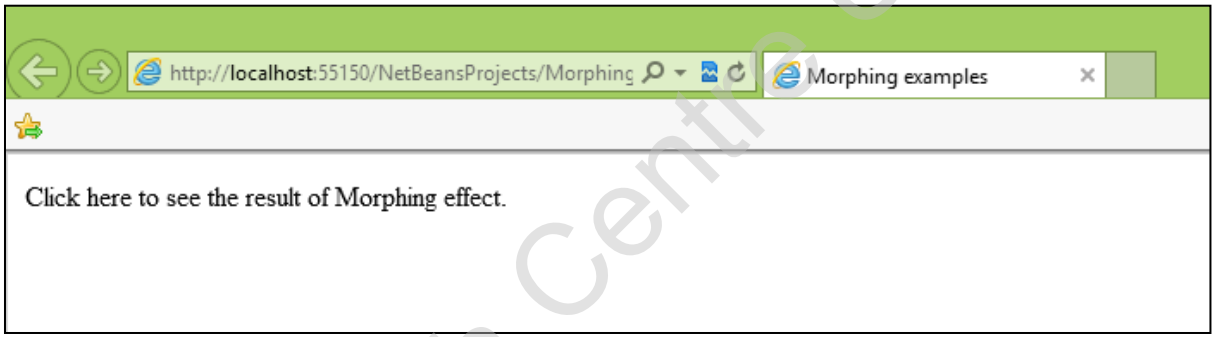
Effects 2-3

- Following Code Snippet shows the morphing effect with the Morphing.jsp file:

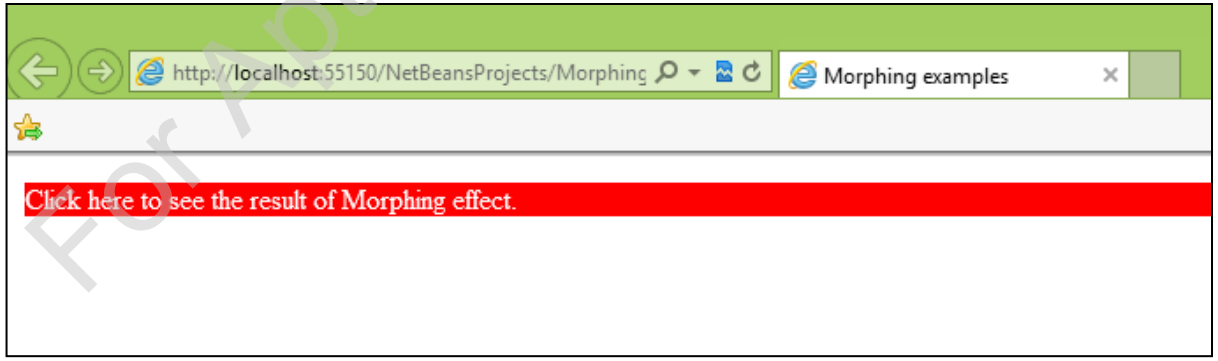
```
...
<script type="text/javascript" src="scriptaculous-js-
1.9.0/lib/prototype.js"></script>
<script type="text/javascript" src="scriptaculous-js-
1.9.0/src/scriptaculous.js?load=effects,controls"></script>
<script type="text/javascript">
function Morphing(element){
    new Effect.Morph(element,
    {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>
...
```

Effects 3-3

- On click of the div tag the Morphing function is called.
- 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 the following figure:



- On clicking the text, the text CSS properties changes and the output is as shown in the following figure:



Combination Effects

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

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.
- Following figure shows the architecture of Dojo:

Architecture of Dojo

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

Language Utilities

- It is the first layer that improves and simplifies the coding technique of JavaScript developers while developing Web sites.

Dojo Event System

- It is the second layer that contains language libraries to process Web application events. These events allow widgets to interact with each other.

Packaging System

- It is the third and the last layer in the hierarchy.
- It helps the 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.
- Widgets are combined into groups called namespaces.

Dojo Programming Model

- **Declarative model**

```
<!-- Creating a Button widget declaratively -->
<button data-dojo-type="dijit.form.Button" id="btnExit">
Exit
</button>
```

  - 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.

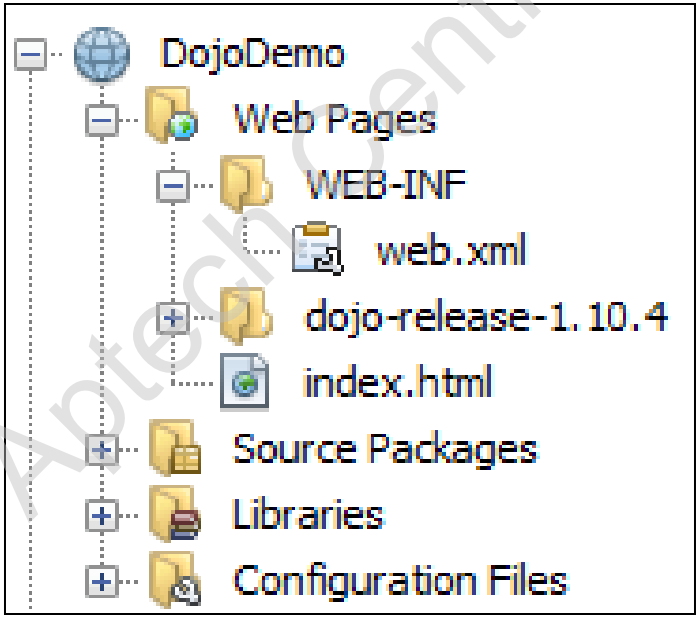
- **Programmatic model**

```
<!-- Creating a Button widget programmatically -->
var btnExit = new dijit.form.Button('Button',((label : 'Exit')),
dijit.byId("btnExit"));
```

  - 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.

Setting Up Dojo 1-2

- To use Dojo toolkit in Web applications, download the latest copy of the Dojo toolkit from http://dojotoolkit.org/download/
- Copy the libraries to the Web application's root folder.
- The Dojo toolkit comprises the modules dojo, digit, dojox, and util.
- Following figure shows the Dojo toolkit added to the Web Pages folder:



Setting Up Dojo 2-2

- Following Code Snippet shows the steps to set up and load Dojo in a Web page:

```
…
<!-- 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>
…
```

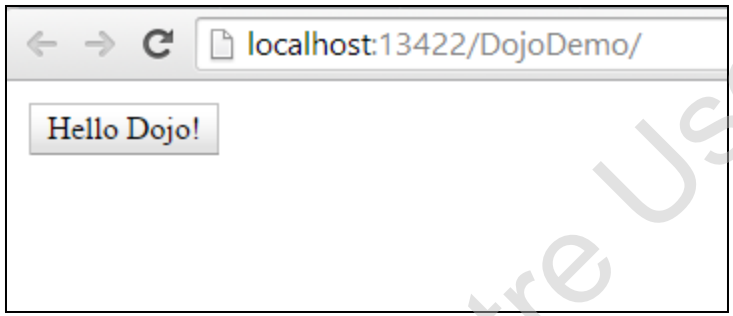
Creating and Connecting a Button Widget to an Event 1-2

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

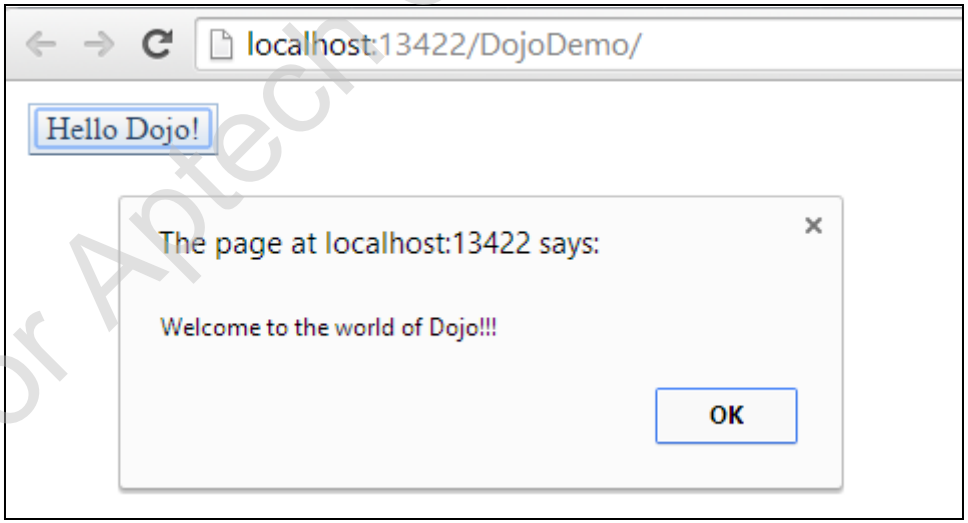
```
…
<link rel="stylesheet" href="./dojo-release-
1.10.4/dijit/themes/tundra/tundra.css"/>
        <script>dojoConfig = {parseOnLoad: true};</script>
        <script src="./dojo-release-1.10.4/dojo/dojo.js"
type="text/javascript"></script>
        <script>
            require(["dijit/form/Button"]);
        </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>
…
```

Creating and Connecting a Button Widget to an Event 2-2

- The output of the file on execution is as shown in the following figure:



- Following figure shows the output after event is handled for the Button widget:



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.

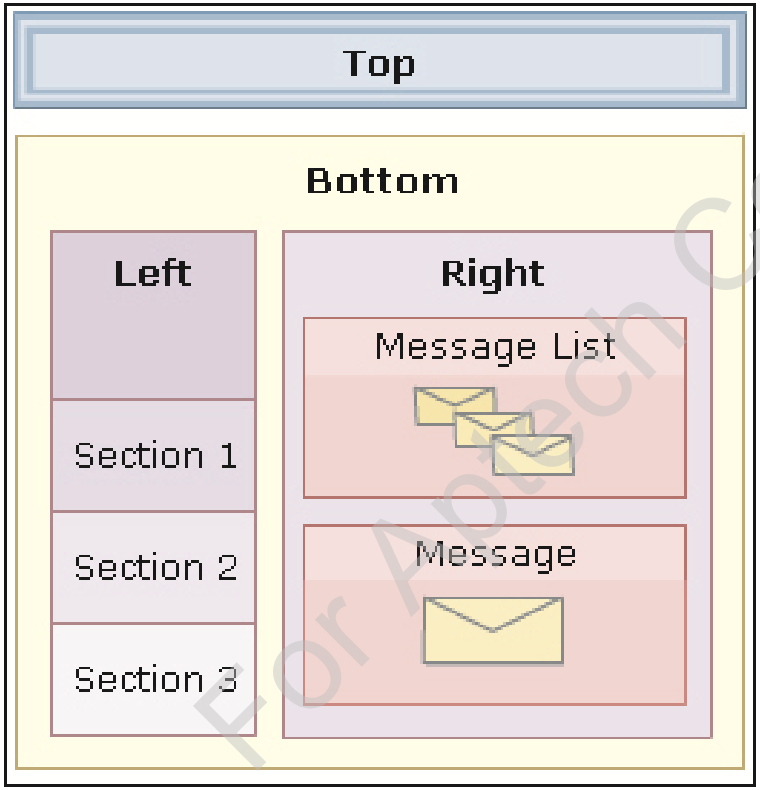
Dijit can either be used declaratively by using special attributes within regular HTML tags or programmatically by using JavaScript.

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.

Dijit Layout

- Dijit has multiple layout widgets that are combined together in a hierarchy.
- 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.
- Following figure shows the hierarchy of Dijit:

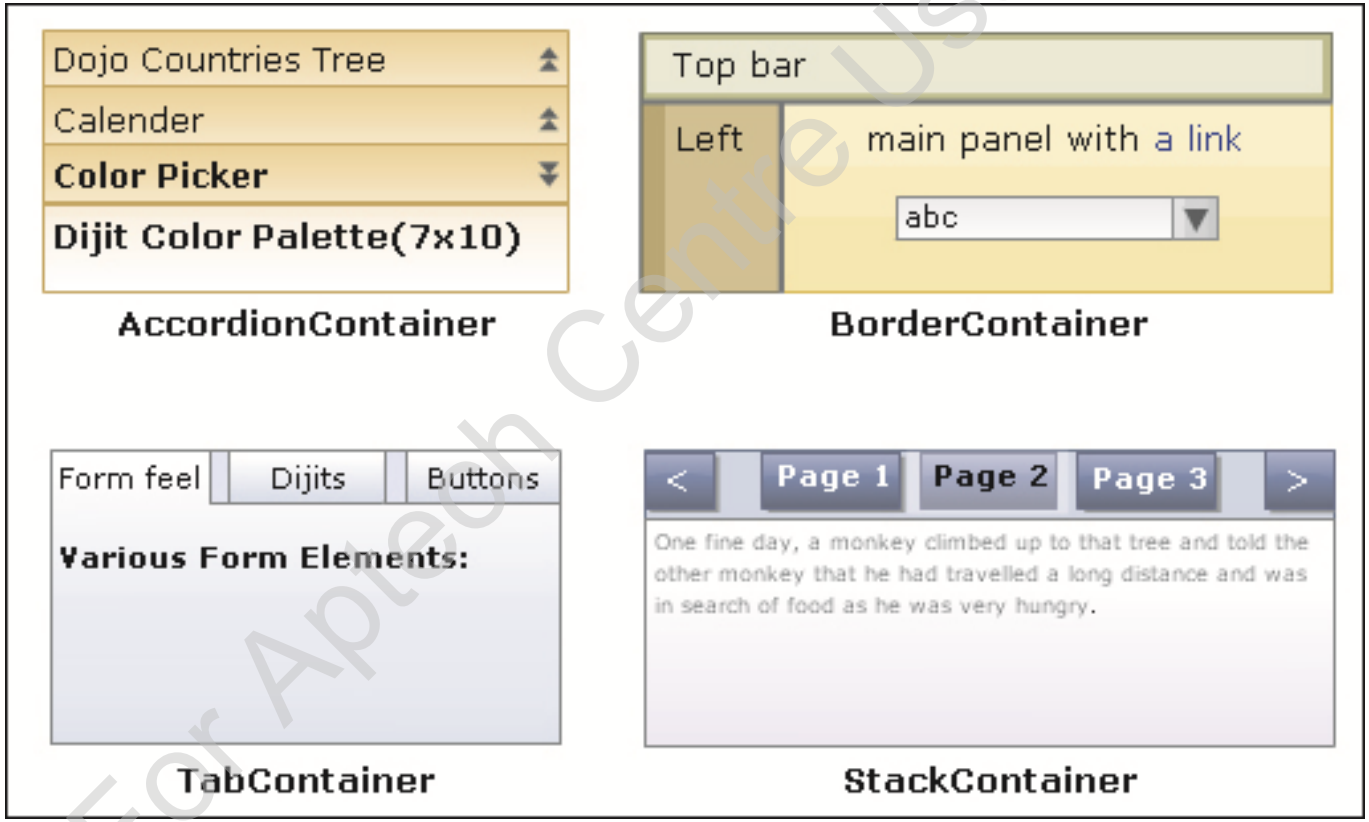


Conceptually, a Dijit is a set of containers that contain following three types of elements:

- 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.
- The third type of elements are the leaf nodes that contain only the content.

Dijit Layout Widgets 1-2

- Dijit's layout widgets are stored in the dijit.layout subpackage.
- Following figure shows the layout widgets available in the dijit.layout subpackage:



Dijit Layout Widgets 2-2

- 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.
BorderContainer	• The BorderContainer layout divides the container into top, left, bottom, right, and center sections.
ContentPane	• The ContentPane layout resembles an internal frame, but contains additional design features.
LayoutContainer	• The LayoutContainer arranges the child nodes in top, left, bottom, right, and client sections.
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.
TabContainer	• The TabContainer layout resembles a tabbed folder. To display the content of a particular tab, click the corresponding tab title.

Check Box and Radio Button Dijits 1-2

- Check boxes are used when you want to allow a user to select zero or more options from a set of options.
- 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.
- You can import the CheckBox and RadioButton classes by using dijit.form.* or by using the require function.
- Following Code snippet demonstrates creation of check box and radio button using dijit:

```
…
<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"/>
```

Check Box and Radio Button Dijits 2-2

```
<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>
…
```

- Following figure shows the creation of check box and radio button using dijit:

AutoCompleter Combo Box Dijit 1-2

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.

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.

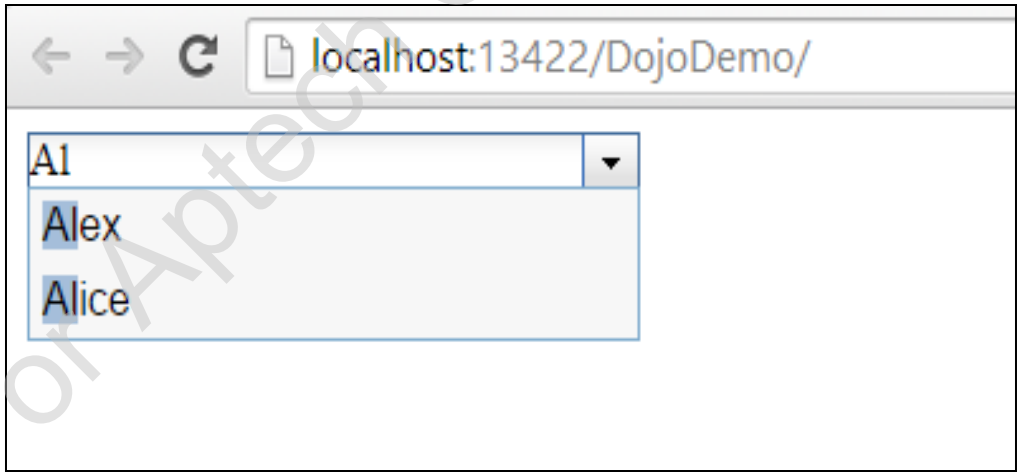
- Following Code Snippet shows the creation of AutoCompleter combo box using dijit in the file Autocompleter.jsp:

```
…
<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>
```

AutoCompleter Combo Box Dijit 2-2

```
<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>
…
```

- The output for the Autocompleter.jsp page is as shown in following figure:



Using dojo.xhrGet() Function 1-3

- Dojo provides a function named dojo.xhrGet() to send and receive data asynchronously.

Following are the options used with the method:				
url	handleAs	load	error	response

- Following Code Snippet demonstrates the code to send an AJAX request using Dojo's dojo.xhrGet() method:

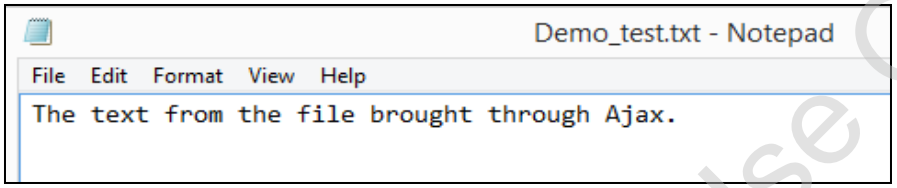
```
…
<link rel="stylesheet" href="./dojo-release-
1.10.4/dijit/themes/tundra/tundra.css"/>
        <script>dojoConfig = {parseOnLoad: true};</script>
        <script src="./dojo-release-1.10.4/dojo/dojo.js"
type="text/javascript"></script>
        <script>
            require(["dijit/form/Button"]);
        </script>
        <script type="text/javascript">
            function FileReading() {
                dojo.xhrGet({
                    url: "Demo_test.txt",
```

Using dojo.xhrGet() Function 2-3

```
                    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>
…
```

Using dojo.xhrGet() Function 3-3

- The content of the Demo_test.txt file is as shown in the following figure:



- The output for the Code Snippet is as shown in the following figure:

- To load the file contents asynchronously, click the Read File.
- The contents are loaded from the file asynchronously and displayed in the div as shown in the following figure:

Defining the load and error Functions

- Following Code Snippet demonstrates the use of the functions loginCallback() and loginError():

```
…
function loginCallback(data, ioArgs){
 alert(data.getElementByTagName("name")[0].childNodes[0].nodeValue);
}

function loginError(data, ioArgs){
  alert('Error when retrieving data from the server!');
}
```

- The loginCallback() function is used to process the response received from the server.
- DataServlet returns an XML response containing the request parameter's value enclosed in a name element.
- The loginError() function is used to display an error message if AJAX request could not be processed.

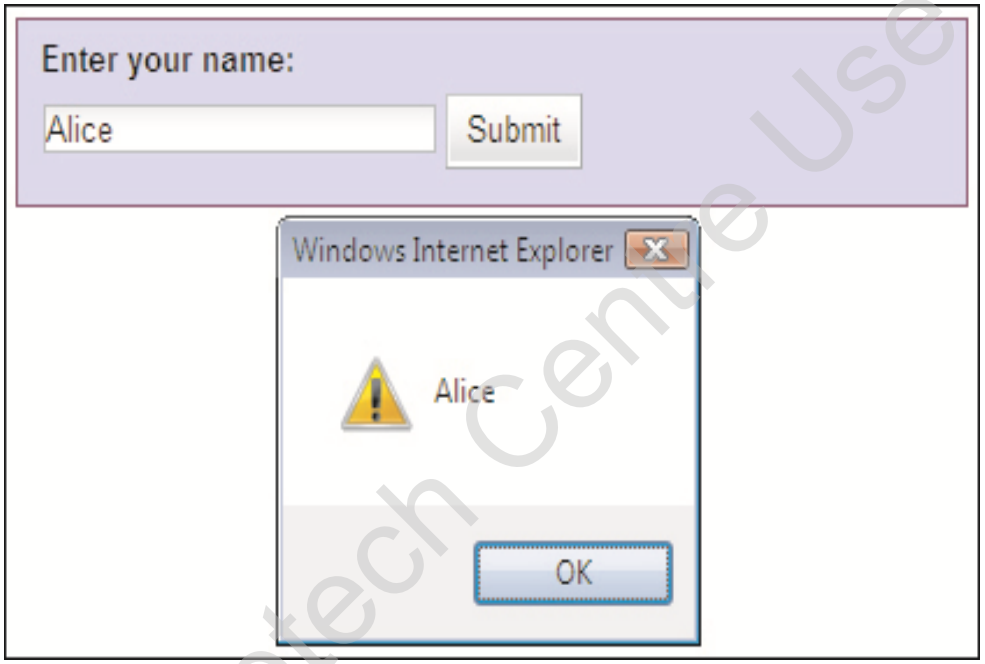
Working with Server-Side Component 1-2

- 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**
  - Following Code Snippet 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:

```
…
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());
 }
}
```

Working with Server-Side Component 2-2

- Following figure shows the implementation of the doGet() method at the server-side:



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.