ALAGAPPA UNIVERSITY

**(Accredited with 'A+' Grade by NAAC (with CGPA: 3.64) in the Third Cycle and Graded
as category - I University by MHRD-UGC)
(A State University Established by the Government of Tamilnadu)**

KARAIKUDI – 630 003

DIRECTORATE OF DISTANCE EDUCATION

M.Sc (INFORMATION TECHNOLOGY)

Second Year – Fourth Semester

31341 – Web Technology

Copy Right Reserved **For Private Use only**

Author:

Dr. P .Prabhu
Assistant Professor in Information Technology
Directorate of Distance Education
Alagappa University,
Karaikudi. 630 003**.**

“**The Copyright shall be vested with Alagappa University”**

All rights reserved. No part of this publication which is material protected by this copyright notice may be reproduced or transmitted or utilized or stored in any form or by any means now known or hereinafter invented, electronic, digital or mechanical, including photocopying, scanning, recording or by any information storage or retrieval system, without prior written permission from the Alagappa University, Karaikudi, Tamil Nadu.

SYLLABI - BOOK MAPPING TABLE
Web Technology

Unit No.	Contents	Mapping in Book
	BLOCK 1 : HTML, JAVA SCRIPT and XML	
1	**HTML Common tags:** List, Tables, images, forms, Frames; Cascading Style sheets.	9-31
2	**Introduction to Java Scripts**, Objects in Java Script, Dynamic HTML with Java Script.	32-43
3	**XML:** Document type definition, XML Schemas, Document Object model, Presenting XML, Using XML Processors: DOM and SAX	44-70
	BLOCK 2 : JAVA BEANS	
4	**Java Beans:** Introduction to Java Beans, Advantages of Java Beans, BDK, Introspection, Using Bound properties, Bean Info Interface,	71-87
5	**Constrained properties**, Persistence, Customizes, Java Beans API, Introduction to EJB's	88-109
	BLOCK 3 : SERVLETS	
6	**Web Servers and Servlets:** Tomcat web server, Introduction to Servlets: Lifecycle of a Servlet, JSDK	110-121
7	**The Servlet API,** The javax.servlet Package, Reading Servlet parameters, Reading Initialization parameters.	122-129
8	**The javax.servlet** HTTP package, Handling Http Request & Responses, Using Cookies-Session Tracking, Security Issues.	130-141
	BLOCK 4 : JAVA SERVER PAGES (JSP)	
9	**Introduction to JSP:** The Problem with Servlet. The Anatomy of a JSP Page, JSP Processing. JSP Application Design with MVC Setting Up and JSP Environment: Installing the Java Software Development Kit, Tomcat Server & Testing Tomcat.	142-154
10	**JSP Application Development:** Generating Dynamic Content, Using Scripting Elements Implicit JSP Objects, Conditional Processing – Displaying Values Using an Expression to Set an Attribute, Declaring Variables and Methods	155-167
11	**Error Handling and Debugging:** Sharing Data Between JSP pages, Requests, and Users Passing Control and Date between Pages – Sharing Session and Application Data – Memory Usage Considerations	168-176
	BLOCK 5 : DATABASE ACCESS AND STRUCTS FRAMEWORK	
12	**Database Access:** Database Programming using JDBC, Studying Javax.sql.* package, Accessing a Database from a JSP Page	177-200
13	**Application – Specific Database Actions**, Deploying JAVA Beans in a JSP Page	201-213
14	**Introduction to struts framework.**	214-223

CONTENTS

PAGE NUMBER

BLOCK 2 : JAVA BEANS

BLOCK 3: SERVLETS

BLOCK 4 : JAVA SERVER PAGES (JSP)

BLOCK – I

HYPER TEXT MARKUP LANGUAGE (HTML) JAVASCRIPT and XML

UNIT 1

HTML - Common Tags and Cascading Style Sheets (CSS)

Structure

1. 0 Introduction

Web technology refers that by which computers can communicate with each other using markup languages like HTML, DHTML and multimedia tools. It gives us a way to interact with computers in World Wide Web. HTML (Hypertext Markup Language) is used to create document on the World Wide Web. Cascading Style sheets are used to add style to web documents. Javascript is a scripting language used to create web pages in clientside. Serverside programming such as servlet and JSP provides client server communication. Database connectivity can be achieved through Java Database Connectivity JDBC.

In this block you will be able to learn about HTML markup language and cascading styles sheets used to create web pages.

1.1 Objectives

After going through the unit, you will be able to;

- Create web page using Common HTML Tags
- Create Tables and use of its various attributes and Table tags
- Work with Frames and its attributes
- Use forms to accept data from the user
- Design a web page using HTML and Javascript

1.2 HTML Common tags

HTML (Hypertext Markup Language) is used to create document on the World Wide Web. It is simply a collection of certain key words called ‘Tags’ that are helpful in writing the document to be displayed using a browser on Internet. It is a platform independent language that can be used on any platform such as Windows, Linux, Macintosh, and so on. To display a document in web it is essential to mark-up the different elements (headings, paragraphs, tables, and so on) of the document with the HTML tags. To view a mark-up document, user has to open the document in a browser.

A browser understands and interpret the HTML tags, identifies the structure of the document (which part are which) and makes decision about presentation (how the parts look) of the document. HTML also provides tags to make the document look attractive using graphics, font size and colors. User can make a link to the other document or the different section of the same document by creating Hypertext Links also known as Hyperlinks.

The essential Common tags that are required to create a HTML document are:

<HTML>.............</HTML>

<HEAD>.............</HEAD>

<BODY>.............</BODY>

Most HTML tags have two parts, an opening tag and closing tag. The closing tag is the same as the opening tag, except for the slash mark e.g</HTML>. The slash mark is always used in closing tags.

HTML Tag <HTML>

The <HTML> tag encloses all other HTML tags and associated text within your document. It is an optional tag. You can create an HTML document that omits these tags, and your browser can still read it and display it. But it is always a good form to include the start and stop tags. The format is:

```
<HTML>
Your Title and Document (contains text with HTML tags) goes here
</HTML>
```

An HTML document has two distinct parts HEAD and BODY.

HEAD Tag <HEAD>

HEAD tag comes after the HTML start tag. It contains TITLE tag to give the document a title that displays on the browsers title bar at the top. The Format is:

```
<HEAD>
<TITLE>
Your title goes here
</TITLE>
</HEAD>
```

BODY Tag <BODY>

The BODY tag contains all the text and graphics of the document with all the HTML tags that are used for control and formatting of the page. The Format is:

```
<BODY>
Your Document goes here
</BODY>
```

An HTML document, web page can be created using a text editor, Notepad or WordPad. All the HTML documents should have the extension .htm or .html. It requires a web browser like Internet Explorer or Netscape Navigator/Communicator to view the document.

Attributes used with <BODY>

BGCOLOR: used to set the background color for the document,for Example:

```
        <BODY BGCOLOR="yellow">
Your document text goes here.
```

</BODY>

TEXT: used to set the color of the text of the document

Example: <BODY TEXT="red">

Document text changed to red color

</BODY>

MARGINS: set the left hand/right hand margin of the document

LEFTMARGIN: set the left hand margin of the document Example:

<BODY LEFTMARGIN="60">
This document is indented 60 pixels from the left hand side of the page.
</BODY>

TOPMARGIN: set the left hand margin of the document

Example:

<BODY TOPMARGIN="60">
This document is indented 60 pixels from the top of the page.
</BODY>

BACKGROUND: It is used to point to an image file (the files with an extension .gif, .jpeg) that will be used as the background of the document. The image file will be tiled across the document.

Example:

<BODY BACKGROUND="filename.gif">
Your document text goes here
</BODY>

Example: An HTML document attribute.html that shows the use of attributes with the <BODY> tag

```
<HTML>
<HEAD>
<TITLE>
Use of Attributes with the Body Tag
</TITLE>
</HEAD>
```

```
<BODYBACKGROUND="logo.gif" text="blue"
TOPMARGIN ="75"
LEFTMARGIN="75">
Your document text will be indented 75 pixels from the left hand and 75 pixels from the top of the page. The background image computer.gif will be tiled across the document. Your image and HMTL document should present at the same place.
</BODY>
</HTML>
```

Note : All Html codings are saved with extension of .html and can be invoked from any web browser like Internet explorer and Netscape navigator.

Container and Empty Tags

All HTML tags are enclosed in angle brackets ‘<’ and ‘>’ i.e. Opening Tag: <HTML> and closing tag: </HTML> same as opening tag except a / (slash) mark. Tags are not case-sensitive i.e. there is no difference in small and capital letters in tags. There are two types of tags:

1. Container Tags:

Tags which have both the opening and closing i.e. <TAG> and </TAG> are called container tags. They hold the text and other HTML tags in between the tags. The <HTML>, <HEAD>, <TITLE> and <BODY> tags are all container tags.

Example:

<TAG> this is a container tag. It holds both the text and HTML tag </TAG>

2. Empty Tags:

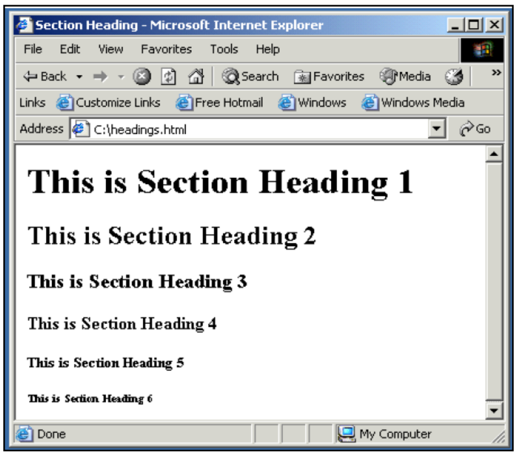
Tags, which have only opening and no ending, are called empty tags. The <HR>, which is used to draw Horizontal, rule across the width of the document, and line break <BR> tags are empty tags.

HTML tags used for formatting a web page are:

SECTION HEADING: <H1>.............<H6>

HTML has six header tags <H1>, <H2>...........<H6> used to specify section headings. Text with header tags is displayed in larger and bolder fonts than the normal body text by a web browser. Every header leaves a blank line above and below it when displayed in browser. Example: An HTML document, headings.html shows the different section headings

```
<HTML>
<HEAD>
<TITLE>
Section Heading
</TITLE>
</HEAD>
<BODY>
<H1>This is Section Heading 1 </H1>
<H2>This is Section Heading 2 </H2>
<H3>This is Section Heading 3 </H3>
<H4>This is Section Heading 4 </H4>
<H5>This is Section Heading 5 </H5>
<H6>This is Section Heading 6 </H6>
</BODY>
</HTML>
```



Section Heading - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Links Customize Links Free Hotmail Windows Windows Media
Address C:\headings.html
Go
Done
My Computer

```
This is Section Heading 1
This is Section Heading 2
This is Section Heading 3
This is Section Heading 4
This is Section Heading 5
This is Section Heading 6
```

Using paragraph tag: <P>

This tag <P> indicates a paragraph, used to separate two paragraphs with a blank line. Example:

```
<P> Welcome to the world of HTML </P>
<P> First paragraph.
Text of First paragraph goes here </P>
```

Output:

```
Welcome to the world of HTML
```

First paragraph. Text of First paragraph goes here Here, two paragraphs are separated with a line. But web browser ignores the line breaks in the second paragraph that can be controlled by putting <BR> tag.

Using Line Break Tag: <BR>

The empty tag <BR> is used, where the text needs to start from a new line and not continue on the same line. To get every sentence on a new line, it is necessary to use a line break.

Using Preformatted Text Tag: <PRE>

<PRE> tag can be used, where it requires total control over spacing and line breaks such as typing a poem. Browser preserves your space and line break in the text written inside the tag.

Using Horizontal Rule Tag: <HR>

An empty tag <HR> basically used to draw lines and horizontal rules. It can be used to separate two sections of text.

Character Formatting Tags

The character formatting tags are used to specify how a particular text should be displayed on the screen to distinguish certain characters within the document.

The most common character formatting tags are:

Boldface <B>: displays text in BOLD

Example: Welcome to the <B> Internet World </B>

Output: Welcome to the Internet World

Italics <I>: displays text in Italic

Example: Welcome to the <I> Internet World </I>

Output: Welcome to the Internet World

- Subscript <SUB>: displays text in Subscript
- Superscript <SUP>: displays text in Superscript
- Small <SMALL>: displays text in smaller font as compared to normal font
- Big <BIG>: displays text in larger font as compared tonormal font

Font Colors and Size:<FONT>

By using <FONT> Tag one can specify the colors, size of the text.

Example: <FONT> Your text goes here </FONT>

Attributes of <FONT> are:

COLOR: Sets the color of the text that will appear on the screen. It can be set by giving the value as #rr0000 for red (in RGB hexadecimal format), or by name.

Example: <FONT COLOR="RED">Your text goes here </ FONT>

SIZE: Sets the size of the text, takes value between 1 and 7, default is 3. Size can also be set relative to default size for example; SIZE=+X, where X is any integer value and it will add with the default size.

FACE: Sets the normal font type, provided it is installed on the user's machine.

1.3 List Tags

HTML Supports several ways of arranging items in lists. The most commonly used are:

- Ordered List (Numbered List)
- Unordered List (Bulleted List)

Ordered List <OL>

Ordered list also called as Numbered list, is used to present a numbered list of item in the order of importance or the item (paragraph) is marked with a number.An ordered list must begin with the <OL> followed by an <LI> list item tag.

Example: An HTML document orderedList.html shows the use of Ordered List

```
<HTML>
<HEAD>
<TITLE>
```

An Ordered List

```
</TITLE>
</HEAD>
<BODY>
<H1><U> Various Terms Used In Internet</U></H1>
<OL>
<LI> WWW-World Wide Web
<LI> URL-Uniform Resource Locator
<LI> HTTP-Hypertext Transfer Protocol
<LI> FTP-File Transfer Protocol
<LI> HTML-Hypertext Markup Language
</OL>
</BODY>
</HTML>
```

Attributes of <OL> tag:

- COMPACT: render a list in compact form.
- TYPE : allows marking list items with different types. By default the list Item markers are set to numbers 1,2,3… so on.

Other values of TYPE attribute are:
Attribute Description

Type = A Capital letter eg. A, B, C………
Type = a Small letter eg. a, b, c,………
Type = I Uppercase Roman Numbers eg. I, II, III……
Type = i Lowercase Roman Numbers eg. i, ii, iii……
Type = 1 eg. 1, 2, 3………….

Unordered List <UL>

Unordered List also called as bulleted list, used to present list of items marked with bullets. An unordered list starts with in <UL>followed by <LI> (List Item) tag. Use of <UL> is very similar to <OL>(ordered list).

Example: Use of Unordered List and Various Attributes

```
<HTML>
<HEAD>
<TITLE> Use of Unordered List </TITLE>
</HEAD>
<BODY>
<UL>
<LI> FRUITS
<UL>
<LI> Banana
```

NOTES

```
<LI> Jack
<LI> Orange
</UL>
<LI> VEGETABLE
<UL>
<LI> Potato
<LI> Cabbage
<LI> Tomato
</UL>
</UL>
</BODY>
</HTML>
```

1.4 Tables

Tables are used to display data in Tabular format, containing rows and columns, on the screen. The <table> tag defines an HTML table. A simple HTML table consists of the table element and one or more tr, th, and td elements. The tr element defines a table row, the element defines a table header, and the td element defines a table cell. A more complex HTML table may also include caption, col, colgroup, thead, tfoot, and tbody elements. The table element can contain the following:

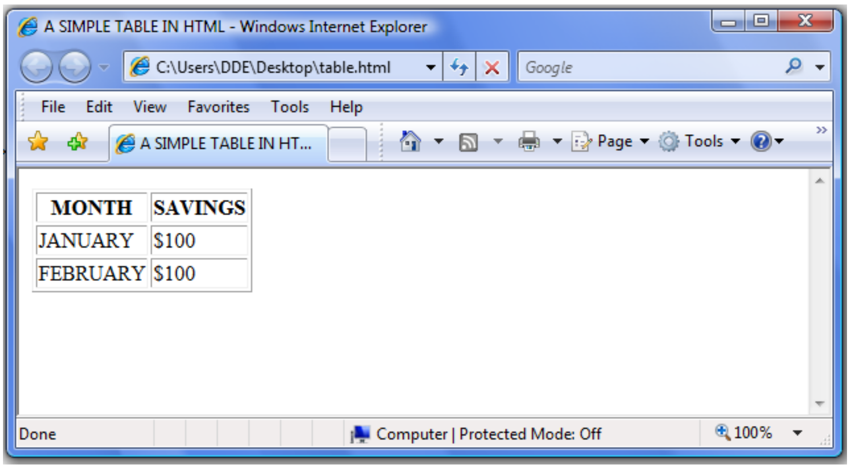
- <CAPTION>: It is used to specify the caption (Label) for the table. The CAPTION element, by default is center-aligned at the top of the Table. It’s ALIGN attribute that takes value
- left, right, center can be used to align the caption. The <CAPTION> tag should appear inside <TABLE>.
- <TR>: Table row, is to specify the row in the table. It holds <TH> Table Heading and <TD> Table Data.
- <TH>: Table Header, is used for giving Heading to Table. By default header elements contents are in bold font and center-aligned.
- <TD>: Table Data, within the row you create columns by <TD> tag. The Table data can consist of lists, images, forms and other element. The TD is a true container that can hold other HTML elements, even more tables.

Example: A simple HTML table, containing two columns and two rows:

```
<HTML>
<HEAD>
<TITLE> A SIMPLE TABLE IN HTML </TITLE>
</HEAD>
<BODY>
```

```
<TABLE BORDER="1">
  <TR>
    <TH>MONTH</TH>
    <TH>SAVINGS</TH>
  </TR>
  <TR>
    <TD>JANUARY</TD>
    <TD>$100</TD>
  </TR>
</TABLE>
</BODY>
<HTML>
```

Output :


A SIMPLE TABLE IN HTML - Windows Internet Explorer
C:\Users\DDE\Desktop\table.html
Google
File Edit View Favorites Tools Help
A SIMPLE TABLE IN HT...
Page
Tools
Done
Computer | Protected Mode: Off
100%

MONTH	SAVINGS
JANUARY	$100
FEBRUARY	$100

Attributes of <TABLE> tag

- BORDER: used to draw borders around all the Table cells. By default, tables are shown without borders i.e. BORDER=0. The size of border can set by assigning an integer value. For example BORDER=3, gives table a three pixel border.
- ALIGN: used to align a table relative to windows border. It can set to left, right or center.
- CELLSPACING: used to set the space between the cells in a table. It takes value in pixel.
- CELLPADDING: used to set the space between the cell data and cell wall in a table. It takes value in pixel.
- WIDTH: used to set the width of the table, as either an absolute width in pixels, or a percentage of the document width. For example: WIDTH=<width in pixel or percent>
- BGCOLOR: set the background color of the table. For example: BGCOLOR=red
- BORDERCOLOR: sets the color of the border of the table. For example: BORDERCOLOR=BLUE

1.5 Images

Images can be placed in a web page by using <IMG> tag. There are many image formats available today, but the most widely used among them are gif and jpeg. The gif format is considered superior to the jpeg format for its clarity and ability to maintain the originality of an image without lowering its quality. Using tools such as GIF constructor or Adobe Photoshop images can be created.

It is an empty tag (only start tag, no end tag) and is written as:

<IMG SRC = image_URL>
SRC – Source of the image file
image_URL – represents the image file with its location.

Example: <IMG SRC=file:///C:/alagappa.GIF>

Here, image_URL =file:///C:/alagappa.GIF, it means image is available in the Hard Drive C: on the local hard disk.

Other attributes used with <IMG> are: -

- ALIGN
- HEIGHT AND WIDTH
- VSPACE and HSPACE
- ALT
- BORDER

ALIGN: used to set the alignment of the text adjacent to the image. It takes the following values:

- ALIGN = LEFT - Displays image on left side and the subsequent text flows around the right hand side of that image
- ALIGN = RIGHT - Displays the image on the right side and the subsequent text flows around the left hand side of that image
- ALIGN = TOP - Aligns the text with the top of the image ALIGN = MIDDLE - Aligns the text with the middle of the image
- ALIGN=BOTTOM - Aligns the text with the bottom of the image

HEIGHT and WIDTH

Height and Width of an image can be controlled by using the HEIGHT and WIDTH attributes in the <IMG> tag as follows:

Example: <IMG SRC= alagappa.GIF HEIGHT=320 WIDTH=240>

HSPACE and VSPACE

White space around an image can be provided by using HSPACE (Horizontal Space) and VSPACE (Vertical Space) attributes of the <IMG> tag. These attributes provide the space in pixels.

Example: *<IMG SRC=alagappa.GIF VSPACE=30 HSPACE=25>*

BORDER

Border around the image can be controlled by using BORDER attribute of <IMG> tag. By default image displays with a thin border. To change the thickness or turn the border off, the value in pixels should set to BORDER attribute.

1.6 Forms

Forms are used in HTML for getting inputs from the user. It can be a registration form, feedback form, order form and so on. To help the user in data entry, form has components; text fields, radio buttons, check boxes, list boxes and so on. When data entry is complete the user submits the form by clicking the submit button on the page. On submit, the data send to server for processing. The processing is done through CGI Scripts - PHP, ASP, or servlet etc. This CGI scripts resides at the server side. After the user fills the form and click the submit button the data passes either through method POST (used to pass large amount of data) or GET (used to pass small amount of data, passed along with the URL) to the server side script that then handles data and perform appropriate action.

The Syntax of the form tag is:

<FORM>………</FORM>

Attributes of <FORM>

- ACTION: Specify the location to which the content of the form are submitted. It’s generally a URL of the CGI scripts.
- METHOD: It specifies the format in which the data send to the script. It can take two values:
  GET: The data submitted to CGI script is displayed in browser address bar for transfer.
  POST: Important data are submitted through post where data not display in browser address bar, during transfer.

NOTES

Syntax:

```
<FORM METHOD="VALUE" ACTION="URL">
</FORM>
```

Form Components

The <INPUT> Tag

This is an empty tag, no end tag. It is used to add graphical user components such as text fields, password fields, check boxes, radio buttons, reset buttons and submit buttons in the form.

Attributes of <INPUT>

- NAME: It defines name for the form components. This field is required for all the types of input except submit and clear.
- SIZE: Specifies the size of the input field in number of characters, used with text or password field.
- MAXLENGTH: Specifies maximum number of characters that can be entered into a text or password field.
- VALUE: For a text or password field, it defines the default text displayed. For a check boxes or radio button, it specified the value that is returned to the server if the box or button is selected. For submit and reset buttons, it defines the label of the button.
- CHECKED: Sets a checkbox or radio button to 'on'.
- TYPE: Set the type of input field.

The <SELECT> Tag

This is a container tag, allows user to select one of the sets of alternatives described by textual labels. Every alternatives is represented by <OPTION>, an empty tag. Attributes of <SELECT>

- MULTIPLE: Allows selecting more than item from the list.
  Example: <SELECT MULTIPLE>
- NAME: Specifies the name that will be submitted as a name-value pair.
- SIZE: Specifies the number of usable items. By default size=1 - the SELECT element treated as pull down menu. For size more than one, SELECT element treated as a list.

Example:

```
<SELECT NAME="COUNTRY">
<OPTION> INDIA
```

```
        <OPTION> USA
        <OPTION> JAPAN
        </SELECT>
```

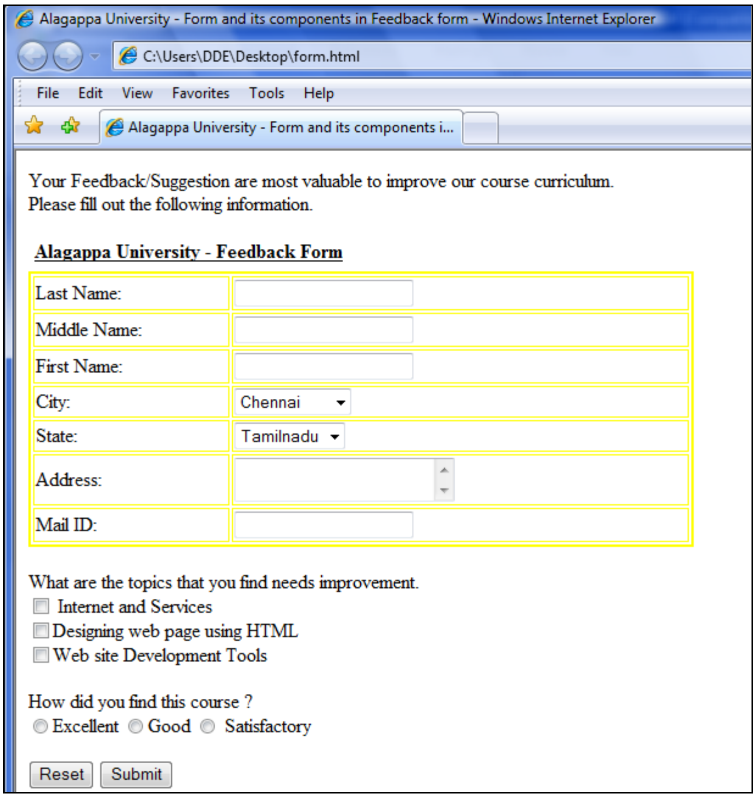
The <OPTION> Tag

An Empty tag <OPTION> can only be used within <SELECT>. Each item in <SELECT> is represented by <OPTION>.

For Example,

```
<html>
<head>
<title> Alagappa University - Form and its components in Feedback
form</title>
</head>
<body>
<form action="" method="post" name="form1">
<p>Your Feedback/Suggestion are most valuable to improve our
course curriculum. <br>
Please fill out the following information.</p>
<table width="45%" border="2" bordercolor="yellow">
<caption align="left"><b><U>Alagappa University - Feedback
Form</U></b></caption>
<tr>
<td width="30%">Last Name: </td>
<td width="70%"><input type="text" name="textfield1"></td>
</tr><tr>
<td>Middle Name: </td>
<td><input type="text" name="textfield2"></td>
</tr><tr>
<td>First Name: </td>
<td><input type="text" name="textfield3"></td>
</tr>
<tr>
<td>City:</td>
<td>
<select name="select">
<option>New Delhi</option>
<option selected>Chennai</option>
<option>Culcutta</option>
<option>Mumbai</option>
<option>Banglore</option>
<option>Hyderabad</option>
</select></td>
</tr><tr>
<td>State:</td>
<td>
```

```
<select name="select">
<option>Andhra</option>
<option>MP</option>
<option>Karnatak</option>
<option>Kerala</option>
<option selected>Tamilnadu</option>
</select></td>
</tr><tr>
<td>Address:</td>
<td><textarea name="textarea"></textarea></td>
</tr><tr>
<td>Mail ID: </td>
<td><input type="text" name="textfield4"></td>
</tr></table>
<p>What are the topics that you find needs improvement.<br>
<input type="checkbox" name="checkbox" value="checkbox"> Internet
and Services<br>
<input type="checkbox" name="checkbox" value="checkbox">Designing
web page using HTML<br>
<input type="checkbox" name="checkbox" value="checkbox”> Planning
and Designing of a web site<br>
<input type="checkbox" name="checkbox" value="checkbox">Web site
Development Tools<br><br>
How did you find this course?<br>
<input name="radiobutton" type="radio" value="radiobutton">Excellent
<input name="radiobutton" type="radio" value="radiobutton">Good
<input name="radiobutton" type="radio" value="radiobutton">
Satisfactory</p>
<p>
<input type="reset" name="Reset" value="Reset">
<input type="submit" name="Submit" value="Submit">
</p>
</form>
</body></html>
```


Alagappa University - Form and its components in Feedback form - Windows Internet Explorer
C:\Users\DDE\Desktop\form.html
File Edit View Favorites Tools Help
Alagappa University - Form and its components i...

Your Feedback/Suggestion are most valuable to improve our course curriculum.
Please fill out the following information.

Alagappa University - Feedback Form

Last Name:	
Middle Name:	
First Name:	
City:	Chennai
State:	Tamilnadu
Address:	
Mail ID:	

What are the topics that you find needs improvement.
□ Internet and Services
□ Designing web page using HTML
□ Web site Development Tools

How did you find this course ?
○ Excellent ○ Good ○ Satisfactory

Reset Submit

1.7 Frames

Frames allow the user to create the web pages in a different manner from the usual way. It provides an effective way to organize your HTML documents in one screen. By Frames, the browser screen divides into a number of panels, which might be completely independent; for instance, you might have seen screen that contains top frame to hold banner or title graphics, left frame - contains table of contents that links to different documents, right frame - to display the contents of the documents, when click over the item in the table of contents in left frame and a bottom frame - to display some copyrights, contacts information etc.

To implement frames, it needs more than one document. The MASTER document contains the frame layout that determines just what users see when they access the frame. There is not any content within the master document instead, it contains one or more FRAMESET element that define the frame layout, and FRAME element that specify just which document are supposed to be loaded into which frame. The actual documents are separate individual pages.

Frames documents are generated by the following HTML tags;

NOTES

Tag	Description
<frameset>	Defines a set of frames
<frame />	Defines a sub window (a frame)
<noframes>	Defines a noframe section for browsers that do not handle frames

<FRAMESET> Tag

Frameset tag defines the number of columns and rows in a frameset. Example:

<FRAMESET rows="20%, 70%, 10%">
The screen divide into three rows 20% row is on top, 70% row is in the middle, and 10% row is on the bottom.
<FRAMESET cols="20%, 80%">

The screen divides into two columns one 20% on left and other80% on right.

<FRAME> Tag

By using <FRAMESET>, you can only define rows and columns all you want, but, nothing shows up in them without the <FRAME> tag. The <FRAME> tag enables you to specify just what appears in row or column you defined. The FRAME tag is not a container tag, it is an empty tag, and so it has no matching end tag.

Example:

<FRAME SRC="contents.html">

SRC (Source): Specifies the Web page that you want to display in the frame.

Other attributes of <FRAME>:

- NAME: It enables you to name a frame, so that, it can function as the target for a link.
- <FRAME NAME="content">
- FRAMEBORDER: Specifies whether or not the frames has a border. You can choose 1(yes) or 0 (no). The default is 1.
  Example: <FRAME FRAMEBORDER="0">
- MARGINWIDTH: Add a left and right margin to the frame. You must specify this in pixels. The value must be greater than The default is 1.

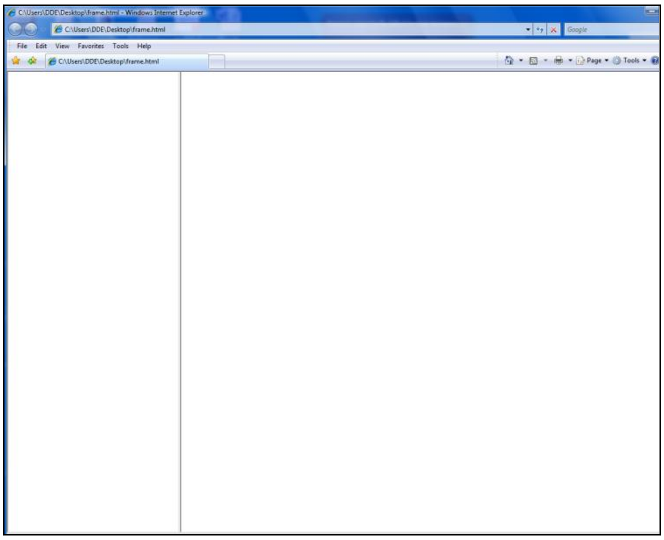
Example: <FRAME MARDINWIDTH="10">

- MARGINHEIGHT: Add a top and bottom margin to the frame. You must specify this in pixels. The value must be greater than 1. The default is 1.
  Example: <FRAME MARGINHEIGHT="10">
- NORESIZE: Fixes the frame so that it cannot be resized.
- SCROLLING: Select scrolling option. It can take the values auto (the default setting) - displays scroll bars only when the window is not large enough to accommodate the frame In the example below we have a frameset with two columns. yes - always display scroll bars, no - never displays scroll bars.
  Example: <FRAME SCROLLING="no">

The first column is set to 25% of the width of the browser window. The second column is set to 75% of the width of the browser window. The document "frame_a.htm" is put into the first column, and the document "frame_b.htm" is put into the second column:

```
<frameset cols="25%,75%">
  <frame src="frame_a.htm" />
  <frame src="frame_b.htm" />
</frameset>
```

Output:


C:\Users\DDE\Desktop\frame.html - Windows Internet Explorer
C:\Users\DDE\Desktop\frame.html
Google
File
Edit
View
Favorites
Tools
Help
C:\Users\DDE\Desktop\frame.html
Page
Tools

1.8 Cascading Style Sheets (CSS).

HTML allows designers to determine how a document is viewed in a browser over the World Wide Web (WWW) by applying tags that manipulate the appearance or assign styles to the text. The purpose of cascading style sheets (CSS) is to allow Web authors to manipulate a Web page's appearance without affecting its HTML structure.

Cascading Style Sheets (CSS) is a style sheet language used to describe the presentation semantics (the look and formatting) of a document written in a markup language. Its most common application is to style web pages written in HTML and XHTML, but the language can also be applied to any kind of XML document, including SVG and XUL.

CSS syntax

CSS syntax follows a three step process. You first select the element you want to style, and then pick a property of the element you want to style and finally the value of this property.

The basic syntax of a style is as follows:

selector { property: value; }

An example would be:

BODY { background-color: yellow; }
Selector: In this example, the selector is “body.” It specifies that this is the element you wish to apply the style to. The selector could either be an HTML tag such as <P>, <LI>, or <H1>, or an element you define yourself

Properties and Values: “background-color” is the property within the selector that will be modified. “yellow” is the value that the property is changed to. It is possible to include more then one property within each selector. In this case, a semicolon is used to separate the properties. An example of this is shown below.

H1 {background-color: transparent; border-style: solid}
selector {property: value; property: value}

There are 3 ways to use CSS.

External

This is the most common implementation of styles. The CSS code is in a separate file with a “.css” extension (NOTE: You must have the .css extension). A snippet is put in the <HEAD> section of the HTML file specifying where the style sheet is. For example, if you had a style sheet called main.css in your styles folder under your web directory, your HTML will be:

```
<HEAD>
<LINK REL="stylesheet" TYPE="text/css" HREF="./styles/main.css" />
</HEAD>
```

The CSS file would then just contain your CSS code AND NOTHING ELSE. For example if all your style sheet did was set the background color of your pages to yellow, the CSS file would look like this:

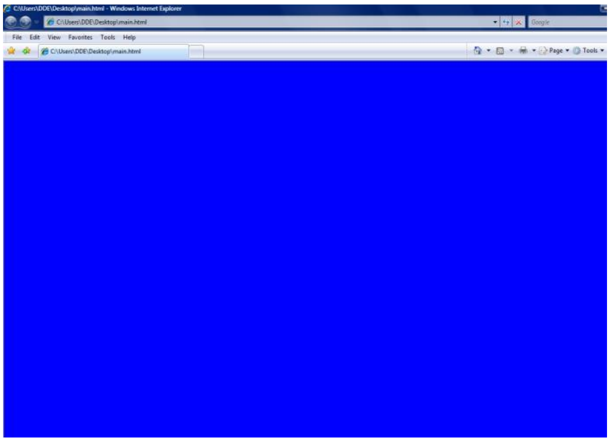
```
BODY {background-color: yellow}
```

Using external style sheets makes it easy to apply the style sheet to multiple pages. Even better, any changes you make to the source style sheet cascades and updates the styling of all your pages.

Internal/Embedded

Say you apply an external style sheet to your page, but then want just one page to have a blue background. Then you can include the page specific CSS code within the <HEAD> section of your page. While other styles of your external style sheet come through, the background color style of the external sheet will be overridden by the internal stylesheet in the page. Now the CSS code needs to be wrapped with special <STYLE> tags in the HTML:

```
<head>
<style type="text/css">
<!--BODY { background-color: blue;}-->
</style>
</head>
```


C:\Users\DDE\Desktop\main.html - Windows Internet Explorer
C:\Users\DDE\Desktop\main.html
Google
File Edit View Favorites Tools Help
C:\Users\DDE\Desktop\main.html
Page
Tools

Inline

Inline uses of CSS is generally not recommended and is slowly being faded out. Inline CSS is where you stick the style directly inside a

HTML tag. For example:

<P STYLE=”color:green”>
The text in this paragraph would then be green.
</P>

The only time you would use Inline CSS is if you need one instance of CSS, say highlighting a sentence or something that would be difficult to do with other HTML methods.

1.9 Check Your Progress

Check Your Progress 1

1. Write True or False for the following:

(a) WIDTH specifies the height of a cell.
(b) The BORDER attribute of the <TABLE> element can be used to create Borderless tables.
(c) Table in HTML is defined row by row. The <TR> tags define a new row.

Check Your Progress 2

2. Write True or False for the following:

(a) RADIO is used for attributes.
(b) Drop down menu is not supported in HTML.
(c) Text fields may be used for multi-line input.

1.10 Answers to Check Your Progress

1. (a) False (b) True (c) True
2. (a) True (b) False (c) False

1.11 Summary

In this lesson you learnt about features of HTML - mainly how to create web page using common controls, tables and use its various attributes. You also learnt about Frames and use of different attributes of Frames and forms. By understanding this unit you may design your own web sites for various applications.

1.12 Key Words

HTML stands for Hyper Text Markup Language. It is not a programming language but a markup language, A markup language is a set of markup tags, It uses markup tags to describe web pages.

CSS stands for Cascading Style Sheets.

1.13 Self-Assessment Questions and Exercises

1. Explain frames. What are advantages of using Frames? Write down the syntax of Frameset and Frame tags with its attributes.
2. Create a HTML document with two frames, which contains a navigation bar on the left-hand. The right side frame is the target of left frame that should display appropriate contents reference to links on the left.
3. Write a CSS code to design your web site.
4. What are the various ways to define CSS? Explain.

1.14 Further Readings

1. Thomas A. Powell, HTML: the complete reference, McGraw-Hill, 2001.
2. Chris Bates, Web Programming, Building Internet Applications, 2nd Edition, Dreamtech
3. Patrick Naughton and Herbert Schildt, The complete Reference Java 2, 5th Edition, Tata McGraw Hill.
4. Jason Hunter Java Servlet Programming, O'Reilly
5. Hans Bergsten, Java Server Pages, O'Reilly
6. Ramesh Bangia,Web Technology, Firewall media, 2006.

UNIT 2

INTRODUCTION TO JAVASCRIPT

Structure

2. 0 Introduction

HTML a merely markup language and is limited to what it can do. HTML creates static web pages that have fixed contents (text, audio, and graphics). It does not allow user to interact with the web page. The increase in expectations, requirements has resulted in continual improvement of HTML and advent of powerful set of language called ‘Scripting Language’.

2. 1 Objectives

After going through the unit, you will be able to;

- Understand introduction to Javascript
- Know about various Objects in JavaScript
- Study about various characteristics of Javascript
- Write a code for client-side validation
- Understand about DHTML (Dynamic Hypertext Markup Language)
- Learn about various javascript programming constructs

2.2 Introduction to Javascript

Scripting language allows user to create an active and dynamic web page that can display dynamic information such as display an appropriate greeting – "Good Morning" or "Good Evening" depending on the time, making web page interactive.

JavaScript developed by Netscape Communication Corporation is the most popular platform independent (run in windows, Linux, Macintosh…etc) Scripting Language. All most all browsers (Internet Explorer, Netscape Navigator) support JavaScript i.e. JavaScript Language Interpreter becomes part of browser.

Characteristics of Java Script

Scripts are embedded in a web page (HTML document) in specifically demarcated section. When browser encounters a script while opening a web page it calls a scripting interpreter, which parses and executes the scripting code.

- JavaScript has the following characteristics:
- JavaScript is a scripting language.
- A scripting language is a simple programming language having some limited features of full-blown programming language, easy to learn and easy to understand.
- JavaScript is platform independent.
- JavaScript that embedded inside HTML document are not tied to any specific hardware platform or operating system and it is interpreted by all most all browsers.
- JavaScript is object-based.
- JavaScript, depends on a collection of built-in objects (entities) for functionality
- JavaScript is event-driven.

\

It responds to user actions by the use of input devices for example: respond to the action of mouse: on mouse over, on mouse out, on mouse click.

There are two ways to use of JavaScript in HTML. One way, you can store your script in a separate simple text file and include it in your HTML document, other way, you can directly write your script inside HTML document.

The basic structure of an HTML file with JavaScript is as follows

```
<SCRIPT Language="JavaScript">
```

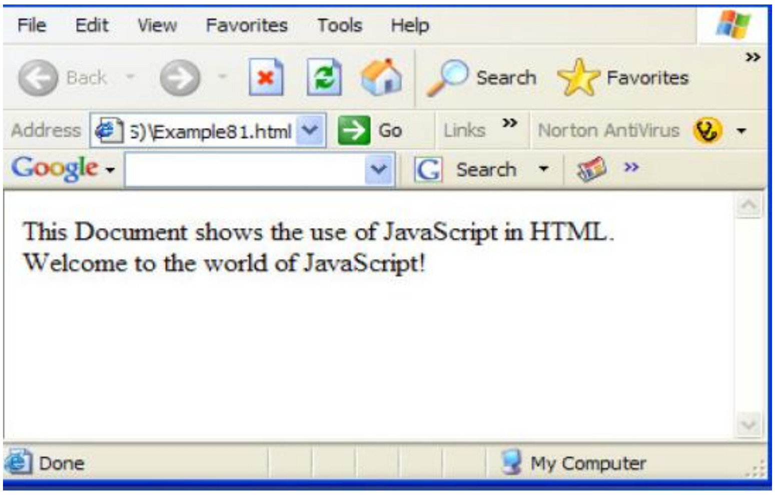
```
        Your JavaScript code goes here

        </SCRIPT>
```

A sample HTML document embedding a JavaScript shown in Example

```
        <HTML>
        <HEAD>
        <TITLE>JavaScript Example 8.1</TITLE>
        </HEAD>
        <BODY>
        This Document shows the use of JavaScript in HTML.
        <BR>
        <SCRIPT language="JavaScript">
        document.write("Welcome to the world of JavaScript!")
        </SCRIPT>
        </BODY>
        </HTML>
```

‘Document’ – Predefined Java Script Object refer to the HTML document (web page).

‘Write’ – It is a predefined method of document object that is used to write text to the HTML document.



```
This Document shows the use of JavaScript in HTML.
Welcome to the world of JavaScript!
```

Variables in JavaScript

Variables, data types, expressions and operators form the core of any programming language and JavaScript is no exception to this. Variable is an identifier used to identify a element. The value of the variable can be changed.

Declaring Variables

In JavaScript, you can declare a variable using the var keyword, as shown in below:

```
var itemName;
var itemPrice;
```

Data Types

In JavaScript, variables can store following data types

- Number
- String
- Boolean

Arrays

Arrays are used to store a sequence of values of same data type. These values are stored in the indexed location within the array. For Example: If you want to store three items of the same type, you can create an array with the name of the items and store their values in the same. The following statement declares an array of length 3:

itemNames=new Array(3)

Expressions

An expression is a part of statement that is evaluated as value. An expression can be any combination of variables, operators and other expressions. In JavaScript there are four types of expressions:

- Assignment expression
- Arithmetic expression
- String expression
- Logical expression

Assignment expression

This expression is used to assign a value to a variable.

Arithmetic Expressions

Expressions in which arithmetic operations are performed are called arithmetic expression.

String Expression

Expressions in which operations are performed on string are called string expression.

Logical Expression

These expressions evaluate to a Boolean (true or false) value.

Looping Constructs

The statements inside any program executes sequentially, line by line. This flow of execution is called as sequential execution. This flow of execution can be controlled (for example: Execute a portion of a program only once, based on a condition. Execute a portion of the program repetitively, based on condition.) by using programming constructs. This programming constructs typically are of two types:

Conditional

These constructs provide the facility to execute a portion of a program only once, based on a condition. Examples of conditional constructs are: if, if…else and switch…case.

Iterative

These constructs provide the facility to execute a portion of the program repetitively, based on a condition. Examples of iterative constructs are: while, do…while and for loops.

Conditional Constructs (if, if…else & switch…case)

1. The if and if…else statement

If and if..else is the primary decision making constructs of any programming language.

Syntax

```
if(condition)
{
Statement1
Statement2
_____________
_____________
}
```

Syntax 2

```
if(condition){
Statement(s)
}
```

```
        else{
        Statement(s)
        }
```

Example

```
<HTML>
<HEAD>
<TITLE>Example </TITLE>
</HEAD><BODY>
<SCRIPT language="JavaScript">
subject1=95
subject2=45
subject3=50
name="Kumar"
document.write("Name:"+name+" ")
document.write("<BR>")
avgMarks=(subject1+subject2+subject3)/ 3
if(avgMarks>40){
document.write("Result: PASS") }
else
{
document.write ("Result: FAIL")
</SCRIPT>
</BODY>
</HTML>
```

2. The switch Statement

The switch statement compares a value of the variable against other values. Multiple if statements can be replaced with switch.

Syntax

```
        switch(variablename)
        {
        case value1:
        statement(s)
        break
        case value2:
        statement(s)
        break
        ;}
```

Iterative Constructs

1. The while statement

The while construct is used to execute statement(s) as long as certain condition is true.

Syntax

```
while(condition(s))
{
statement(s)
}
default:
statement(s)
}
```

2. The do…while statement

This statement is very similar to the while statement, except that it first execute the statement then checking the condition whereas in while first condition checking then statement execution. So, in case of do… while the statement(s) execute at least once even if the condition is false.

Syntax:

```
do
{
        Statement(s)
}
while(condition(s))
```

3. The for statement

The for statement is also an iterative construct, used to execute statement(s) based on some condition.

Syntax:

```
for(variable initialization;checking condition;change value of variable)

{

Statement(s)

}
```

2.2.1 Objects in JavaScript

The JavaScript object model is a simple one. The bulk of these objects deal with window content are documents, links, forms, and so forth. In addition to window-content objects, JavaScript supports a small handful of "built-in" objects. These built-in objects are available regardless of window content and operate independently of whatever page Netscape has loaded.

Window Object

The window object is a top level object in the JavaScript object model. It is also default object i.e. the statement document.write("Hello! Welcome to JavaScript World") is actually expanded as window.document.write("Hello! Welcome to JavaScript World") by the JavaScript interpreter. Hierarchy of objects is as follows;

Window -> Frame -> Location.> History -> Navigator -> document

Built-in Objects

The three built-in objects are: the String object, the Math object, and the Date object. Each of these provides great functionality, and together they give JavaScript its power as a scripting language. These built-in objects are discussed in depth in this chapter and you will find many examples for use in your own projects.

String Object

String is a bona fide object, and the String keyword can be used to create new strings,

For example String Object,

```
var myString = new String();
myString = "This is a string";
alert ("This is a string".length)
```

Math Object

Math object provides built-in constants and methods for performing calculations within the script. For example, to return the value of pi, you use:

```
var pi = Math.PI;
```

Date Object

The Date object is similar to the String object in that you create new instances of the object when you assign it to a variable. It differs from the String object in that you use a statement called new to create it. Using the new statement, you create a Date object that contains information down to the millisecond at that instant.

The Syntax for Creating a New Date Object

To create a new Date object, you use the new operator. The Date object is pre-defined in JavaScript, and will build this instance filling in the current date and time from the client's system clock. Here is the syntax:

variableName = new Date(parameters)

For example, today = new Date();

You have several optional parameters that you may send to the Date object, as follows:

```
variableName = new Date()
variableName = new Date("month day, year hours:minutes:seconds")
variableName = new Date(year, month, day)
variableName = new Date(year, month, day, hours, minutes, seconds)
```

2.2.2 DHTML (Dynamic Hypertext Markup Language)

The group of technologies known as Dynamic Hypertext Markup Language (DHTML) can make your Web pages come alive. DHTML lets the Web page author add functions to the page that can change everything on the page. Instead of having to download new information from the server computer every time something on the page changes, changes can be made by the HTML document. Web pages built with Dynamic HTML are richer and more interactive, react faster, and don’t use much bandwidth.

The major elements of DHTML include HTML code, scripting languages such as JavaScript, the Document Object Model, Cascading Style Sheets, multimedia filters, and the browser itself.

“DHTML is the combination of several built-in browser features in fourth-generation browsers that enable a Web page to be more dynamic.” Dynamic HTML has also been described as a set of commands mixed with text that describe how a Web page should appear. “Dynamic HTML” is a marketing term coined by both Netscape and Microsoft to describe a series of technologies introduced in the 4.0 versions of their browsers, to enhance the “dynamic” capabilities of those browsers.”.

Advantages of DHTML

(1) DHTML makes documents dynamic. Dynamic documents:

- Allow the designer to control how the HTML displays Web pages’ content.
- React and change with the actions of the visitor.
- Can exactly position any element in the window, and change that position after the document has loaded.
- Can hide and show content as needed.

(2) DHTML allows any HTML element (any object on the screen that can be controlled independently using JavaScript) in Internet Explorer to be manipulated at any time, turning plain HTML into dynamic HTML.

(3) With DHTML, changes occur entirely on the client-side (on the user’s browser).

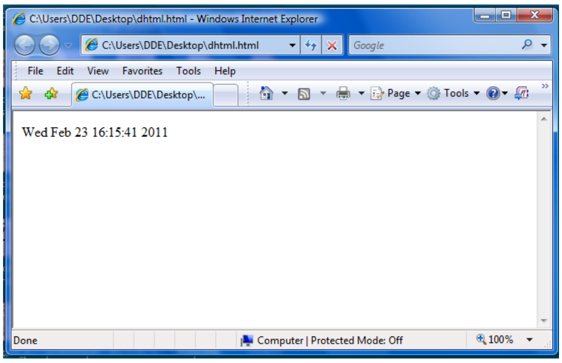
(4) Using DHTML gives the author more control over how the page is formatted and how content is positioned on the page.

JavaScript and HTML

JavaScript can create dynamic HTML content. In JavaScript, the statement: document.write(), is used to write output to a web page. The following example uses JavaScript to display the current date and time on a page:

```
<html>
<body>
<script type="text/JavaScript">
document.write(Date());
</script></body>
</html>
```

Output :



```
Wed Feb 23 16:15:41 2011
```

C:\Users\DDE\Desktop\dhtml.html - Windows Internet Explorer
C:\Users\DDE\Desktop\dhtml.html
Google
File
Edit
View
Favorites
Tools
Help
C:\Users\DDE\Desktop\...
Page
Tools
Done
Computer | Protected Mode: Off
100%

2.3 Check Your Progress

1. Write True or False for the following :

   (a) Scripting language allows user to create an static web page.
   (b) Javascript supports Window object.
   (c) DHTML, changes occur entirely on the client-side.

2.4 Answers to Check Your Progress Questions

1. (a) False (b) True (c) True

2.5 Summary

JavaScript developed by Netscape Communication Corporation is the most popular platform independent (run in windows, Linux, Macintosh…etc) Scripting Language. "DHTML is the combination of several built-in browser features in fourth-generation browsers that enable a Web page to be more dynamic." Dynamic HTML has also been described as a set of commands mixed with text that describe how a Web page should appear.

2.6 Key Words

JavaScript is used in billions of Web pages to add functionality, validate forms, communicate with the server, and much more.

2.7 Self-Assessment Questions and Exercises

1. Explain about various types of javascript objects
2. Write the advantages of DHTML.

2.8 Further Readings

1. Thomas A. Powell, Fritz Schneider, JavaScript: the complete reference, McGraw-Hill, 2001
2. Chris Bates, Web Programming, Building Internet Applications, 2nd Edition, Dreamtech
3. Patrick Naughton and Herbert Schildt, The complete Reference Java 2, 5th Edition, Tata McGraw Hill.

4. Jason Hunter Java Servlet Programming, O'Reilly
5. Hans Bergsten, Java Server Pages, O'Reilly
6. Ramesh Bangia, Web Technology, Firewall media, 2006

UNIT 3 XML - eXtensible Markup Language

Structure

3. 0 Introduction

One of the difficulties with HTML documents, word processor documents, spreadsheets, and other forms of documents is that they mix structure with formatting. This makes it difficult to manage content and design, because the two are intermingled. XML is one of the most widely used formats for sharing structured information today; between programs between computers, between people. It is a text-based markup language derived from SGML Standard Generalized Markup Language.

3. 1 Objectives

After going through the unit you will be able to;

Understand XML (eXtensible Markup Language)
Know about Document Type Definition (DTD) and XML Schema
Explain about Document Object Model (DOM)
Know how to Present XML
Use of XML Processor / Parser
Implement SAX Simple API for XML

3. 2 XML

XML stands for eXtensible Markup Language. It is designed to transport and store data. It is a markup language much like HTML but unlike HTML, XML was designed to carry data, not to display data XML tags are not predefined. You must define your own tags d designed to be self-descriptive.

XML Benefits

XML provides the following benefits, they are,

XML Holds Data, Nothing More

XML does not really do much of anything. Rather, developers can create XML-based languages that store data in a structure way. Applications can then use this data to do any number of things.

XML Separates Structure from Formatting

One of the difficulties with HTML documents, word processor documents, spreadsheets, and other forms of documents is that they mix structure with formatting. This makes it difficult to manage content and design, because the two are intermingled.

As an example, in HTML, there is a <u> tag used for underlining text. Very often, it is used for emphasis, but it also might be used to mark a book title. It would be very difficult to write an application that searched through such a document for book titles.

In XML, the book titles could be placed in <book_title> tags and the emphasized text could be place in <em> tags. The XML document does not specify how the content of either tag should be displayed. Rather, the formatting is left up to an external stylesheet. Even though the book titles and emphasized text might appear the same, it would be relatively straight forward to write an application that finds all the book titles. It would simply look for text in <book_title> tags. It also becomes much easier to reformat a document; for example, to change all emphasized text to be italicized rather than underlined, but leave book titles underlined.

XML Promotes Data Sharing

Very often, applications that hold data in different structures must share data with one another. It can be very difficult for a developer to map the different data structures to each other. XML can serve as a go between. Each application's data structure is mapped to an agreed-upon XML structure. Then all the applications share data in this XML format. Each application only has to know two structures, its own and the XML structure, to be able to share data with many other applications.

XML is Free

XML doesn't cost anything to use. It can be written with a simple text editor or one of the many freely available XML authoring tools, such as XML Notepad. In addition, many web development tools, such as Dreamweaver and Visual Studio .NET have built-in XML support. There are also many free XML parsers, such as Microsoft's MSXML (downloadable from microsoft.com) and Xerces (downloadable at apache.org).

XML is Human-Readable

XML documents are (or can be) read by people. Perhaps this doesn't sound so exciting, but compare it to data stored in a database. It is not easy to browse through a database and read different segments of it as you would a text file. Take a look at the XML document below.

When talking about XML, here are some terms that would be helpful:

- XML: eXtensible Markup Language, a standard created by the W3Group for marking up data.

- DTD: Document Type Definition, a set of rules defining relationships within a document; DTDs can be "internal" (within a document) or "external" (links to another document).

- XML Parser: Software that reads XML documents and interprets or "parse" the code according to the XML standard. A parser is needed to perform actions on XML, such as comparing an XML document to a DTD.

XML documents, at a minimum, are made of two parts: the prolog and the content. The prolog or head of the document usually contains the administrative metadata about the rest of document. It will have information such as what version of XML is used, the character set standard used, and the DTD, either through a link to an external file or internally. Content is usually divided into two parts that of the structural markup and content contained in the markup, which is usually plain text.

XML Documents

An XML document is made up of the following parts.

- An optional prolog.
- A document element, usually containing nested elements.
- Optional comments or processing instructions.
-

The Prolog - The prolog of an XML document can contain the following items.

- An XML declaration
- Processing instructions
- Comments
- A Document Type Declaration

Let's take a look at a simple prologue for an XML document:

<?xml version="1.0" encoding="iso-8859-1"?>

<?xml declares to a processor that this is where the XML document begins. version="1.0" declares which recommended version of XML the document should be evaluated in.encoding="iso-8859-1" identifies the standardized character set that is being used to write the markup and content of the XML.

The XML Declaration

The XML declaration, if it appears at all, must appear on the very first line of the document with no preceding white space. It looks like this.

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

This declares that the document is an XML document. The version attribute is required, but the encoding and standalone attributes are not. If the XML document uses any markup declarations that set defaults for attributes or declare entities then standalone must be set to no.

Processing Instructions

Processing instructions are used to pass parameters to an application. These parameters tell the application how to process the XML document. For example, the following processing instruction tells the application that it should transform the XML document using the XSL style sheet beatles.xsl.

<?xml-stylesheet href="beatles.xsl" type="text/xsl"?>

As shown above, processing instructions begin with and <? end with ?>.

Comments

Comments can appear throughout an XML document. Like in HTML, they begin with <!-- and end with -->.

<!--This is a comment-->A Document Type Declaration

The Document Type Declaration (or DOCTYPE Declaration) has three roles.

- It specifies the name of the document element.
- It may point to an external Document Type Definition (DTD).
- It may contain an internal DTD.

<!DOCTYPE beatles>If a DOCTYPE Declaration points to an external DTD, it must either specify that the DTD is on the same system as the XML document itself or that it is in some public location. To do so, it uses the keywords SYSTEM and PUBLIC. It then points to the location of the DTD using a relative Uniform Resource Indicator (URI) or an absolute URI. Here are a couple of examples.

Syntax

<!--DTD is on the same system as the XML document-->
<!DOCTYPE beatles SYSTEM "dtds/beatles.dtd">Syntax
<!--DTD is publicly available-->
<!DOCTYPE beatles PUBLIC "-//Webucator//DTD Beatles 1.0//EN" "http://www.webucator.com/beatles/DTD/beatles.dtd"> As shown in the second declaration above, public identifiers are divided into three parts:

An organization (e.g, Webucator)
A name for the DTD (e.g, Beatles 1.0)
A language (e.g, EN for English)

Elements

Every XML document must have at least one element, called the document element. The document element usually contains other elements, which contain other elements, and so on. Elements are denoted with tags.

Elements have a few particular rules:

1. Element names can be any mixture of characters, with a few exceptions. However, element names are case sensitive, unlike HTML. For instance, <elementname> is different from <ELEMENTNAME>, which is different from <ElementName>.

Note: The characters that are excluded from element names in XML are &, <, ", and >, which are used by XML to indicate markup. The character: should be avoided as it has been used for special extensions in XML. If you want to use these restricted characters as part of the content within elements but do not want to create new elements, then you would need to use the following entities to have them displayed in XML:

XML Entity Names for Restricted Characters	
Use	**For**
&amp;	&
&lt;	<
&gt;	>
&quot;	"

2. Elements containing content must have closing and opening tags.

<elementName> (opening) </elementName> (closing) Note that the closing tag is the exact same as the opening tag, but with a backslash in front of it.

The content within elements can be either elements or character data. If an element has additional elements within it, then it is considered a parent element; those contained within it are called child elements. For example,

<elementName>This is a sample of <anotherElement> simple XML</anotherElement>coding</elementName>.

So in this example, <elementName> is the parent element. <anotherElement> is the child of elementName, because it is nested within elementName.

Elements can have attributes attached to them in the following format:
<elementName attributeName="attributeValue" >

For example ***dde.xml***

```
<?xml version="1.0"?>
<note>
   <to>DDE</to>
   <from>Student</from>
```

```
  <heading>Greetings</heading>
   <body> Happy New Year </body>
        </note>
```

Empty Elements

Not all elements contain other elements or text. For example, in XHTML, there is an img element that is used to display an image. It does not contain any text or elements within it, so it is called an empty element. In XML, empty elements must be closed, but they do not require a separate close tag. Instead, they can be closed with a forward slash at the end of the open tag as shown below.

<img src="images/paul.jpg"/>The above code is identical in funciton to the code below.

<img src="images/paul.jpg"></img>Attributes

XML elements can be further defined with attributes, which appear inside of the element's open tag as shown below.

Syntax

```
<name title="Sir">
 <firstname>Paul</firstname>
 <lastname>McCartney</lastname>
</name>CDATA
```

Sometimes it is necessary to include sections in an XML document that should not be parsed by the XML parser. These sections might contain content that will confuse the XML parser, perhaps because it contains content that appears to be XML, but is not meant to be interpreted as XML. Such content must be nested in CDATA sections. The syntax for CDATA sections is shown below.

Syntax

```
<![CDATA[
 This section will not get parsed
 by the XML parser.
]]>White Space
```

In XML data, there are only four white space characters.

Tab
Line-feed
Carriage-return

Single space

There are several important rules to remember with regards to white space in XML.

White space within the content of an element is significant; that is, the XML processor will pass these characters to the application or user agent. White space in attributes is normalized; that is, neighboring white spaces are condensed to a single space. White space in between elements is ignored.

xml:space Attribute

The xml:space attribute is a special attribute in XML. It can only take one of two values: default and preserve. This attribute instructs the application how to treat white space within the content of the element. Note that the application is not required to respect this instruction.

XML Syntax Rules

XML has relatively straightforward, but very strict, syntax rules. A document that follows these syntax rules is said to be well-formed.

- There must be one and only one document element.
- Every open tag must be closed.
- If an element is empty, it still must be closed.
    - Poorly-formed: <tag>
    - Well-formed: <tag></tag>
    - Also well-formed: <tag />
- Elements must be properly nested.
    - Poorly-formed: <a><b></a></b>
    - Well-formed: <a><b></b></a>
- Tag and attribute names are case sensitive.
- Attribute values must be enclosed in single or double quotes.

3.2.1 Document Type Definition (DTD)

A Document Type Definition (DTD) is a type of schema. The purpose of DTDs is to provide a framework for validating XML documents. By defining a structure that XML documents must conform to, DTDs allow different organizations to create shareable data files

The building blocks of XML documents

XML documents (and HTML documents) are made up by the following building blocks:

- Elements
- Tag
- Attributes
- Entities
- PCDATA and
- CDATA

Elements - Elements are the main building blocks of both XML and HTML documents.

Examples of HTML elements are "body" and "table". Examples of XML elements could be "note" and "message". Elements can contain text, other elements, or be empty. Examples of empty HTML elements are "hr", "br" and "img".

Declaring an Element

In the DTD, XML elements are declared with an element declaration. An element declaration has the following syntax:

<!ELEMENT element-name (element-content)>

Empty elements

Empty elements are declared with the keyword EMPTY inside the parentheses:

<!ELEMENT element-name (EMPTY)>

example:<!ELEMENT img (EMPTY)>

Elements with data

Elements with data are declared with the data type inside parentheses:

<!ELEMENT element-name (#CDATA)>
or
<!ELEMENT element-name (#PCDATA)>
or
<!ELEMENT element-name (ANY)>example:<!ELEMENT note (#PCDATA)>

#CDATA means the element contains character data that is not supposed to be parsed by a parser.#PCDATA means that the element contains data that IS going to be parsed by a parser.The keyword ANY declares an element with any content. If a #PCDATA section contains elements, these elements must also be declared.

Elements with children (sequences)

Elements with one or more children are defined with the name of the children elements inside the parentheses:

```
<!ELEMENT element-name (child-element-name)>
        Or
<!ELEMENT element-name (child-element-name,child-element-
name,.....)>
```

example: <!ELEMENT note (to,from,heading,body)>

When children are declared in a sequence separated by commas, the children must appear in the same sequence in the document. In a full declaration, the children must also be declared, and the children can also have children. The full declaration of the note document will be:

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to      (#CDATA)>
<!ELEMENT from    (#CDATA)>
<!ELEMENT heading (#CDATA)>
<!ELEMENT body    (#CDATA)>
```

Wrapping

If the DTD is to be included in your XML source file, it should be wrapped in a DOCTYPE definition with the following syntax:

```
<!DOCTYPE root-element [element-declarations]>example: <?xml
version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note (to,from,heading,body)>
  <!ELEMENT to      (#CDATA)>
  <!ELEMENT from    (#CDATA)>

 <!ELEMENT heading (#CDATA)>
  <!ELEMENT body    (#CDATA)>
]>

<note>
  <to>Tove</to>
  <from>Jani</from>
```

NOTES

```
  <heading>Reminder</heading>
  <body>Don't forget me this weekend</body>
</note>
```

Tags - Tags are used to markup elements.

A starting tag like <element_name> mark up the beginning of an element, and an ending tag like </element_name> mark up the end of an element.

Examples:

A body element: <body>body text in between</body>.
A message element: <message>some message in between</message>

Attributes - Attributes provide extra information about elements.

Attributes are placed inside the start tag of an element. Attributes come in name/value pairs. The following "img" element has an additional information about a source file:

<img src="computer.gif" />

The name of the element is "img". The name of the attribute is "src". The value of the attribute is "computer.gif". Since the element itself is empty it is closed by a " /".

PCDATA - PCDATA means parsed character data.

Think of character data as the text found between the start tag and the end tag of an XML element.

PCDATA is text that will be parsed by a parser. Tags inside the text will be treated as markup and entities will be expanded.

CDATA - CDATA also means character data.

CDATA is text that will NOT be parsed by a parser. Tags inside the text will NOT be treated as markup and entities will not be expanded.

Entities

Entities as variables used to define common text. Entity references are references to entities.

Most of you will known the HTML entity reference: "&nbsp;" that is used to insert an extra space in an HTML document. Entities are expanded when a document is parsed by an XML parser.

The following entities are predefined in XML:

Entity References	**Character**
&lt	<
&gt	>
&amp	&
&quot	"
&apos	'

Creating DTDs

DTDs are simple text files that can be created with any basic text editor. Although they look a little cryptic at first, they are not terribly complicated once you get used to them.

A DTD outlines what elements can be in an XML document and the attributes and subelements that they can take.

A DTD can be declared inline in your XML document, or as an external reference.

Internal DTD

```
<?xml version="1.0"?>
<!DOCTYPE note [
  <!ELEMENT note    (to,from,heading,body)>
  <!ELEMENT to      (#PCDATA)>
  <!ELEMENT from    (#PCDATA)>
  <!ELEMENT heading (#PCDATA)>
  <!ELEMENT body    (#PCDATA)>
>

<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

External DTD

This is the same XML document with an external DTD: (Open it in IE5)

```
<?xml version="1.0"?>
<!DOCTYPE note SYSTEM "note.dtd">
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend!</body>
</note>
```

This is a copy of the file **"note.dtd"** containing the Document Type Definition:

```
<?xml version="1.0"?>
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```

Limitations of DTD

DTDs do not have built-in datatypes.
DTDs do not support user-derived datatypes.
DTDs allow only limited control over cardinality (the number of occurrences of an element within its parent).
DTDs do not support Namespaces or any simple way of reusing or importing other schemas.

3.2.2 XML Schema

XML Schema is an XML-based language used to create XML-based languages and data models. An XML schema defines element and attribute names for a class of XML documents. The schema also specifies the structure that those documents must adhere to and the type of content that each element can hold.

XML documents that attempt to adhere to an XML schema are said to be instances of that schema. If they correctly adhere to the schema, then they are valid instances. This is not the same as being well formed. A well-formed XML document follows all the syntax rules of XML, but it does necessarily adhere to any particular schema. So, an XML document can be well formed without being valid, but it cannot be valid unless it is well formed.

An XML schema describes the structure of an XML instance document by defining what each element must or may contain. An element is limited by its type. For example, an element of complex type can contain child elements and attributes, whereas a simple-type element can only contain text.

The <schema> Element

The <schema> element is the root element of every XML Schema:

```
<?xml version="1.0"?>

<xs:schema>
...
...
</xs:schema>
```

The <schema> element may contain some attributes. A schema declaration often looks something like this:

```
<?xml version="1.0"?>

<xs:schema xmlns:xs="http://www.dde,org/2001/XMLSchema"
targetNamespace="http://www.w3schools.com"
xmlns="http://www.w3schools.com"
elementFormDefault="qualified">
        ...
        ...
</xs:schema>
```

Defining a Simple Element

The syntax for a simple element is:

```
<xs:element name="aaa" type="bbb"/>
```

where aaa is the name of the element and bbb is the data type of the element.
XML Schema has a lot of built-in data types. The most common types are:

```
xs:string
xs:decimal
xs:integer
xs:boolean
xs:date
xs:time
```

Example:

Few of XML elements:

```
<name>Rahul</name>
<age>15</age>
<currentdate>2007-05-15</currentdate>
```

The corresponding simple element definitions:

```
<xs:element name="name" type="xs:string"/>
<xs:element name="age" type="xs:integer"/>
<xs:element name="currentdate" type="xs:date"/>
```

Default Values for Simple Elements

Simple elements may have a specified default value OR a fixed specified value .A default value is automatically assigned to the element when no other value is specified for example to set the "orange" default value .

```
<xs:element name="fruit" type="xs:string" default="orange"/>
```

Fixed Values for Simple Elements

A fixed value is also automatically assigned to the element, and it cannot further specify another value. In the following example the fixed value is "apple":

```
<xs:element name="fruit" type="xs:string" fixed="apple"/>
```

XSD Complex Elements:

A complex element contains other elements or attributes.

Complex Element

It is an XML element that contains other elements and/or attributes. They are of four types:

- empty elements
- elements that contain only other elements
- elements that contain only text
- elements that contain both other elements and text
-

Note: Each of these elements may contain attributes as well!

Examples of Complex Elements

A complex empty XML element, "employee"

```
<employee eid="1234"/>
```

A complex XML element, "employee", which contains only other elements:

```
<employee>
<firstname>Amit</firstname>
<lastname>Gupta</lastname>
</employee>
```

A complex XML element, "employee", which contains only text:

```
<employee type="category">Programmer</employee>
```

A complex XML element, "event", which contains both elements and text:

```
<event>
It occured on <date lang="norwegian">15.05.07</date> ....
</event>
```

Defining a Complex Element:

Look at this complex XML element, "employee", which contains only other elements:

```
<employee>
<firstname>Amit</firstname>
<lastname>Gupta</lastname>
</employee>
```

We can define a complex element in an XML Schema in two different ways:

1. "employee" element can be declared directly by naming the element, like this:

```
<xs:element name="employee">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="firstname" type="xs:string"/>
      <xs:element name="lastname" type="xs:string"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

In the above described method only "employee" element can use the specified complex type. Note that the child elements, "firstname" and "lastname", are surrounded by the <sequence> indicator. This means that the child elements must appear in the same order as they are declared.

2. "employee" element can have a type attribute refering to the name of the complex type to use:

```
<xs:element name="employee" type="personinfo"/>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>

 <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Using the method described above, several elements can refer to the same complex type, like this:

```
<xs:element name="employee" type="personinfo"/>
<xs:element name="employer" type="personinfo"/>
<xs:element name="teammember" type="personinfo"/>
<xs:complexType name="personinfo">
  <xs:sequence>
    <xs:element name="firstname" type="xs:string"/>
    <xs:element name="lastname" type="xs:string"/>
  </xs:sequence>
</xs:complexType>
```

Attributes

Simple elements do not contain attributes. If an element has attributes, then it is of a complex type element. But the attribute itself is always declared as a simple type.

The syntax for defining an attribute is:

```
<xs:attribute name="aaa" type="bbb"/>
```

where aaa is the name of the attribute and bbb specifies the data type of the attribute.

XML Schema has a lot of built-in data types. The most common types are:

xs:string
xs:decimal
xs:integer
xs:boolean
xs:date
xs:time

Example:

Here is an XML element with an attribute:

<name lang="EN">Rahul</name>

And here is the corresponding attribute definition:

<xs:attribute name="lang" type="xs:string"/>

Default Values for Attributes

Attributes may have a specified default value OR a specified fixed value A default value is automatically assigned to the attribute when no other value is specified for example..the default value is "EN":

<xs:attribute name="lang" type="xs:string" default="EN"/>

Fixed Values for Attributes

A fixed value is automatically assigned to the attribute, and it cannot further specify another value for example..the fixed value is "EN":

<xs:attribute name="lang" type="xs:string" fixed="EN"/>

Optional and Required Attributes
Attributes are optional by default. To specify that the attribute is required, use the "use" attribute:

<xs:attribute name="lang" type="xs:string" use="required"/>

Schema types

Elements can be of simple type or complex type.

- Simple type elements can only contain text.
- They can not have child elements or attributes.
- All the built-in types are simple types (e.g, xs:string).

NOTES

- Schema authors can derive simple types by restricting another simple type.

For example, an email type could be derived by limiting a string to a specific pattern. Simple types can be atomic (e.g, strings and integers) or non-atomic (e.g, lists).

- Complex-type elements can contain child elements and attributes as well as text.
- By default, complex-type elements have complex content, meaning that they have child elements.
- Complex-type elements can be limited to having simple content, meaning they only contain text. They are different from simple type elements in that they have attributes.
- Complex types can be limited to having no content, meaning they are empty, but they have may have attributes.
- Complex types may have mixed content - a combination of text and child elements.

A Simple XML Schema

Let's take a look at a simple XML schema, which is made up of one complex type element with two child simple type elements.

Code Sample: SchemaBasics/Demos/Author.xsd

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.dde.org/2001/XMLSchema">
  <xs:element name="Author">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="FirstName" type="xs:string" />
        <xs:element name="LastName" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Validating an XML Instance Document

In the last section, you saw an example of a simple XML schema, which defined the structure of an Author element. The code sample below shows a valid XML instance of this XML schema.

Code Sample: SchemaBasics/Demos/MarkTwain.xml

```
<?xml version="1.0"?>
<Author xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
 xsi:noNamespaceSchemaLocation="Author.xsd">
   <FirstName>Mark</FirstName>
   <LastName>Twain</LastName>
</Author>
```

3.2.3 Document Object Model (DOM)

Document Object Model: DOM is a platform- and language-neutral interface that provides a standard model of how the objects in an XML object are put together and a standard interface for accessing and manipulating these objects and their inter-relationships.

The DOM is an interface that exposes an XML document as a tree structure comprised of nodes. The DOM allows you to programmatically navigate the tree and add, change and delete any of its elements. The DOM programming interface standards are defined by the World Wide Web Consortium (W3C). The W3C site provides a comprehensive reference of the XML DOM.

3.2.4 Presenting XML

XML is about defining data. With XML you can define documents that are understood by computers. But to make these documents understandable to humans, you need to show them. You can present XML by the following ways,

Presenting XML document with JavaScript

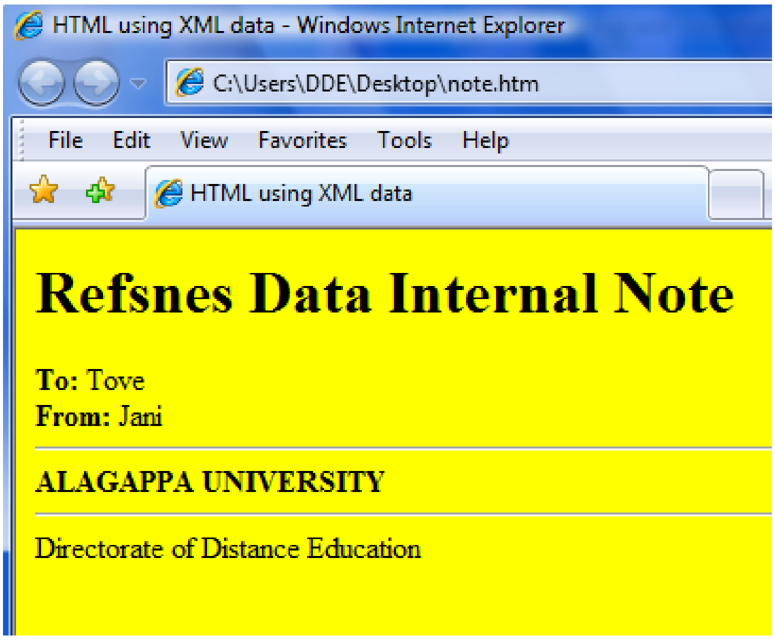
note.xml

```
<?xml version="1.0" encoding="ISO8859-1" ?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>ALAGAPPA UNIVERSITY</heading>
  <body>Directorate of Distance Education</body>
</note>
```

note.html

```
<html>
<head>
<script language="JavaScript" for="window" event="onload">
var xmlDoc = new ActiveXObject("Microsoft.XMLDOM")
xmlDoc.async="false"
xmlDoc.load("note.xml")
nodes = xmlDoc.documentElement.childNodes
to.innerText = nodes.item(0).text
from.innerText = nodes.item(1).text
header.innerText = nodes.item(2).text
body.innerText = nodes.item(3).text
</script>
<title>HTML using XML data</title>
</head>
<body bgcolor="yellow">
<h1>Refsnes Data Internal Note</h1>
<b>To: </b><span id="to"></span>
<br>
<b>From: </b><span id="from"></span>
<hr>
<b><span id="header"></span></b>
<hr>
<span id="body"></span>
</body>
</html>
```

Output :


HTML using XML data - Windows Internet Explorer
C:\Users\DDE\Desktop\note.htm
File Edit View Favorites Tools Help
HTML using XML data

Refsnes Data Internal Note

To: Tove
From: Jani

ALAGAPPA UNIVERSITY

Directorate of Distance Education

Presenting XML with CSS

Cascading Style sheets (CSS offer possibilities to show XML.It works just like adding styles to HTML elements.

cd_catelog.css *file*

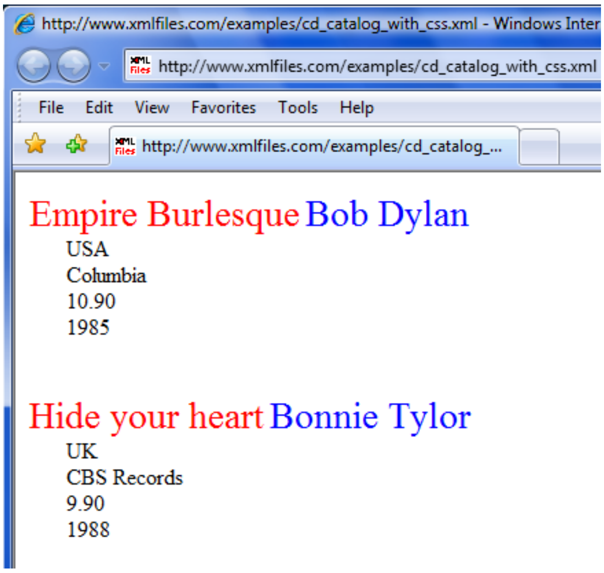
```
CATALOG
{
background-color: #ffffff;
width: 100%;
}
CD
{
display: block;
margin-bottom: 30pt;
margin-left: 0;
}
TITLE
{
color: #FF0000;
font-size: 20pt;
}
ARTIST
{
color: #0000FF;
font-size: 20pt;
}
COUNTRY,PRICE,YEAR,COMPANY
{
Display: block;
color: #000000;
margin-left: 20pt;
}
```

cd_catelog.xml

```
<?xml version="1.0" encoding="ISO8859-1" ?>
<?xml-stylesheet type="text/css" href="cd_catalog.css"?>
<CATALOG>
  <CD>
    <TITLE>Empire Burlesque</TITLE>
    <ARTIST>Bob Dylan</ARTIST>
    <COUNTRY>USA</COUNTRY>
    <COMPANY>Columbia</COMPANY>
    <PRICE>10.90</PRICE>
    <YEAR>1985</YEAR>
```

```
</CD>
<CD>
  <TITLE>Hide your heart</TITLE>
  <ARTIST>Bonnie Tylor</ARTIST>
  <COUNTRY>UK</COUNTRY>
  <COMPANY>CBS Records</COMPANY>
  <PRICE>9.90</PRICE>
  <YEAR>1988</YEAR>
</CD>
<CD>
  <TITLE>Greatest Hits</TITLE>
  <ARTIST>Dolly Parton</ARTIST>
  <COUNTRY>USA</COUNTRY>
  <COMPANY>RCA</COMPANY>
  <PRICE>9.90</PRICE>
  <YEAR>1982</YEAR>
</CD>  </CATALOG>
```

Output


http://www.xmlfiles.com/examples/cd_catalog_with_css.xml - Windows Inter
http://www.xmlfiles.com/examples/cd_catalog_with_css.xml
File
Edit
View
Favorites
Tools
Help
http://www.xmlfiles.com/examples/cd_catalog_...

```
Empire Burlesque Bob Dylan
    USA
    Columbia
    10.90
    1985

Hide your heart Bonnie Tylor
    UK
    CBS Records
    9.90
    1988
```

3.3 Using XML Processor / Parser

An XML parser is a necessary piece of the XML puzzle. The XML processor takes the XML document and DTD and processes the information so that it may then be used by applications requesting the information.

The parser is a software module that reads the XML document to find out the structure and content of the XML document. The structure and content can be derived by the processor because XML documents contain self-explanatory data. To manipulate an XML document, XML parser is needed. The parser loads the document into the computer's memory. Once the document is loaded, its data can be manipulated using the appropriate parser.

3.3.1 DOM: Document Object Model

The XML Document Object Model (XML DOM) defines a standard way to access and manipulate XML documents using any programming language (and a parser for that language).

The DOM presents an XML document as a tree-structure (a node tree), with the elements, attributes, and text defined as nodes. DOM provides access to the information stored in your XML document as a hierarchical object model.

The DOM converts an XML document into a collection of objects in a object model in a tree structure (which can be manipulated in any way). The textual information in XML document gets turned into a bunch of tree nodes and an user can easily traverse through any part of the object tree, any time. This makes easier to modify the data, to remove it, or even to insert a new one. This mechanism is also known as the random access protocol.

DOM is very useful when the document is small. DOM reads the entire XML structure and holds the object tree in memory, so it is much more CPU and memory intensive. The DOM is most suited for interactive applications because the entire object model is present in memory, where it can be accessed and manipulated by the user.

3.3.2 SAX Simple API for XML

This API was an innovation, made on the XML-DEV mailing list through a product collaboration, rather than being a product of the W3C.

SAX (Simple API for XML) like DOM gives access to the information stored in XML documents using any programming language (and a parser for that language).

This standard API works in serial access mode to parse XML documents. This is a very fast-to-execute mechanism employed to read and write XML data comparing to its competitors. SAX tells the application, what is in the document by notifying through a stream of parsing events. Application then processes those events to act on data.

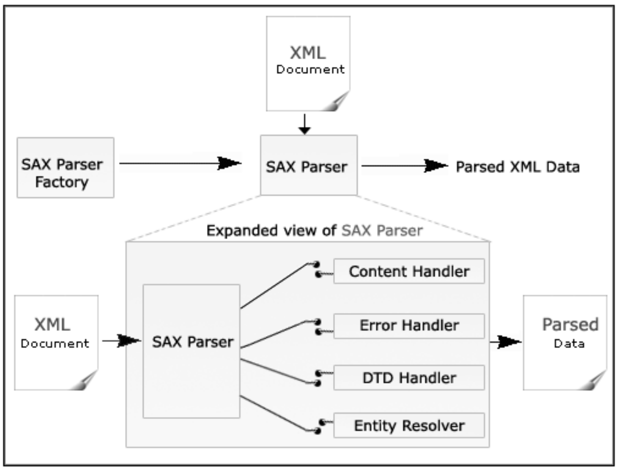
NOTES

SAX is also called as an event-driven protocol, because it implements the technique to register the handler to invoke the callback methods whenever an event is generated. Event is generated when the parser encounters a new XML tag or encounters an error, or wants to tell anything else. SAX is memory-efficient to a great extend.

SAX is very useful when the document is large.

Using SAX Parser

At the very first, create an instance of the SAXParserFactory class which generates an instance of the parser. This parser wraps a SAXReader object. When the parser's parse() method is invoked, the reader invokes one of the several callback methods (implemented in the application). These callback methods are defined by the interfaces ContentHandler, ErrorHandler, DTDHandler, and EntityResolver.



Xml Parser

Brief description of the key SAX APIs:

SAXParserFactory

SAXParserFactory object creates an instance of the parser determined by the system property, using the class javax.xml.parsers.SAXParserFactory.

SAXParser

The SAXParser interface defines several kinds of parse() methods. Generally, XML data source and a DefaultHandler object is passed to the parser. This parser processes the XML file and invokes the appropriate method on the handler object.

SAXReader

The SAXParser wraps a SAXReader (may use SAXParser's getXMLReader() and configure it). It is the SAXReader which carries on the conversation with the SAX event handlers you define.

DefaultHandler

Not shown in the diagram, a DefaultHandler implements the ContentHandler, ErrorHandler, DTDHandler, and EntityResolver interfaces (with null methods).You override only the ones you're interested in.

ContentHandler

Methods like startDocument, endDocument, startElement, and endElement are invoked when an XML tag is recognized. This interface also defines methods characters and processingInstruction, which are invoked when the parser encounters the text in an XML element or an inline processing instruction, respectively.

ErrorHandler

Methods error, fatalError, and warning are invoked in response to various parsing errors. The default error handler throws an exception for fatal errors and ignores other errors (including validation errors). To ensure the correct handling, you'll need to supply your own error handler to the parser.

DTDHandler

Defines methods you will rarely call. Used while processing a DTD to recognize and act on declarations for an unparsed entity.

EntityResolver

The resolveEntity method is invoked when the parser needs to identify the data referenced by a URI. .

DOM reads the entire XML structure and holds the object tree in memory, so it is much more CPU and memory intensive. For that reason, the SAX API is preferred for server-side applications and data filters that do not require any memory intensive representation of the data.

3.4 Check Your Progress

1. Write True or False for the following:

(a) In XML data contains only four white space characters.
(b) XML documents, at a minimum, are made of three parts.
(c) ErrorHandler deals fatalError.

3.5 Answers to Check Your Progress Questions

1. (a) True (b) False (c) True

3.6 Summary

XML is designed to transport and store data. It is a markup language much like HTML but unlike HTML, XML was designed to carry data, not to display data XML tags are not predefined. The XML Document Object Model (XML DOM) defines a standard way to access and manipulate XML documents using any programming language (and a parser for that language).

3.7 Keywords

XML stands for eXtensible Markup Language.
DOM presents an XML document as a tree-structure (a node tree), with the elements, attributes, and text defined as nodes

3.8 Self-Assessment Questions and Exercises

1. What is XML. What are its characteristics?
2. Explain about DTD.
3. Brief description about key SAX APIs.

3.9 Further Readings

1. Heather Williamson, XML: The Complete Reference, Tata Mgraw Hill (2001)
2. Chris Bates, Web Programming, Building Internet Applications, 2nd Edition, Dreamtech
3. Patrick Naughton and Herbert Schildt, The complete Reference Java 2, 5th Edition, Tata McGraw Hill.
4. Jason Hunter Java Servlet Programming, O'Reilly
5. Hans Bergsten, Java Server Pages, O'Reilly
6. Ramesh Bangia, Web Technology, Firewall media, 2006.

NOTES

BLOCK – II
JAVA BEANS

UNIT 4
INTRODUCTION TO JAVA BEANS

Structure

4.0 Introduction

This Unit gives an overview of the software component models, features of a software component and an overview of the JavaBeans Technology. A software component model is a specification for how to develop reusable software components and how these component objects can communicate with each other. Java promises a truly open cross platform execution environment for permitting the creation of applications once with deployment everywhere. However, the Java component model, JavaBeans, will provide these benefits too real people, the non-programmers of the world, finally fulfilling the promise of software component technology.

4.1 Objectives

At the end of this unit you will be able to know,

- Introduction to Software Component Models
- Introduction to JavaBeans
- Working with Beans Development Kit(BDK)
- How to build simple Bean
- Bean Introspection
- About using Bound Properties
- Bean customization
- Java Beans API

4.2 Software Component Model – Java Beans

A software component model is a specification for how to develop reusable software components and how these component objects can communicate with each other. We can now focus on how component models work in a general sense. A Java Bean has been defined as ‘a reusable software component that can be manipulated visually in a builder tool.”. Simply, a Java Bean is a software Component.

4.3 Advantages of Javabeans

- Portable : Written in java with no platform-native code
- Lightweight: It should be possible to implement a component as small as a push button or as large as a complete spreadsheet or word processor.
- Simple to create: it should be a simple job to create a java component with our implementing countless methods.
- Hostable in other component models : It should be possible to use JavaBeans as a first-class ActiveX, OpenDoc, or other component
- Able to access remote data: A Java component may use any of the standard distributed objects (Remote Method Invocation RMI) or distributed data mechanisms to access remote data.

4.4 BDK (Bean Development Kit)

JavaBeans are reusable software components written in Java. These components may be built into an application using an appropriate building environment. The BDK is delivered separately from the JDK. You can download the BDK freely from the JavaBeans web site. This site contains instructions for installing the BDK on your system. Here is a general description of the BDK files and directories:

1. README.html contains an entry point to the BDK documentation

2. LICENSE.html contains the BDK license agreement

3. GNUmakefile and Makefile are Unix and Windows makefiles (.gmk and .mk suffixes) for building the demos and the BeanBox, and for running the BeanBox

4. beans/apis contains

    a. a java directory containing JavaBeans source files
    b. a sun directory containing property editor source files

5. beans/beanbox contains

    a. makefiles for building the BeanBox
    b. scripts for running the BeanBox
    c. a classes directory containing the BeanBox class files
    d. a lib directory containing a BeanBox support jar file used by MakeApplet's produced code sun and sunw directories containing BeanBox source (.java) files
    e. a tmp directory containing automatically generated event adapter source and class files, .ser files, and applet files automatically generated by MakeApplet

6. beans/demos contains

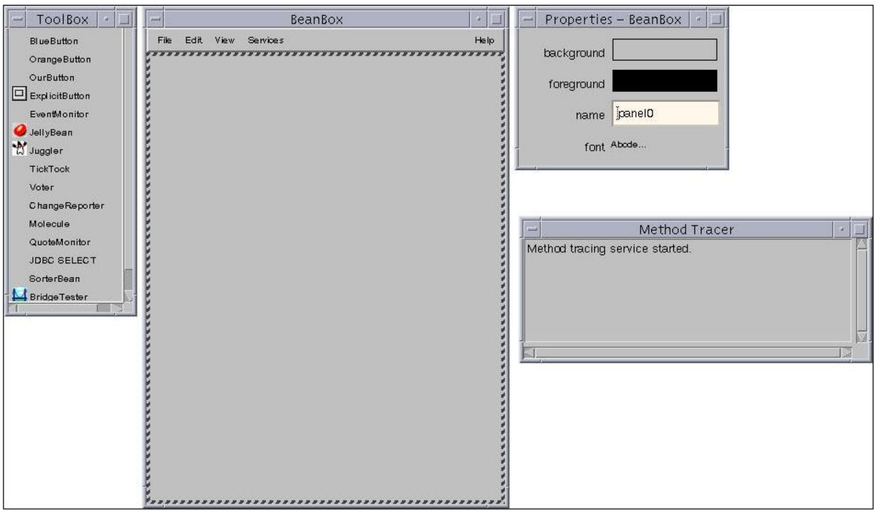
7. makefiles for building the demo Beans
    a. an HTML directory containing an applet wrapper demonstration that must be run in appletviewer, HotJava, or JDK1.1-compliant browsers
    b. a sunw directory containing
    c. a wrapper directory containing a Bean applet wrapper
    d. a demos directory containing demo source file
8. beans/doc contains

    a. demos documentation
    b. a javadoc directory containing JavaBeans and JavaBeans-related class and interface documentation
    c. miscellaneous documentation
9. beans/jars contain jar files for demo Beans

NOTES

Starting the BDK

1. Type the cd c:\bdk\beanbox to change to the correct bean directory after downloading BDK Package.
2. Type run to execute batch file that starts the BDK. Three windows show in the following figure will appear.



background

foreground

name panel0

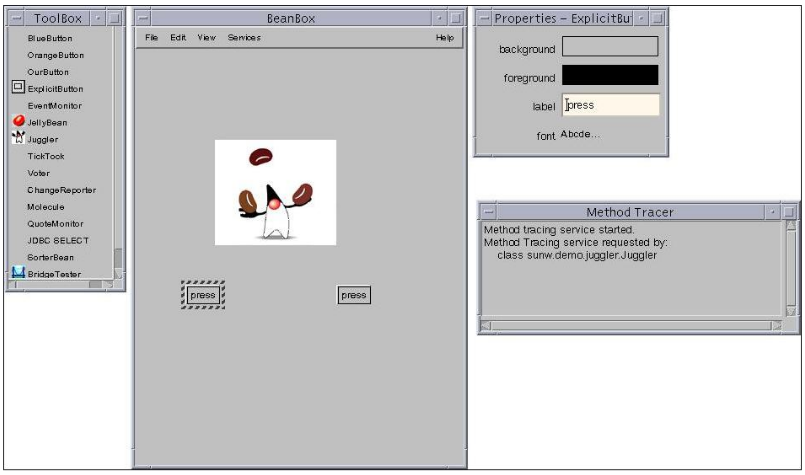
font Abcde...

Method tracing service started.

Adding beans to the canvas

To add beans to the BeanBox canvas, click on the bean's name or icon in the ToolBox window, then click on the location in the BeanBox canvas where you want the new bean to appear. The bean that appears on the BeanBox canvas is the current selected bean (shown by a hatched border around the bean) and its properties appear in the Properties sheet.

The properties in the Properties sheet can be edited by typing new values or by special editors (eg for colours). For example, in the diagram below, we have added two ExplicitButton beans and a Juggler bean to the canvas. Click on the left hand bean labeled Press. This bean becomes the current selected bean.



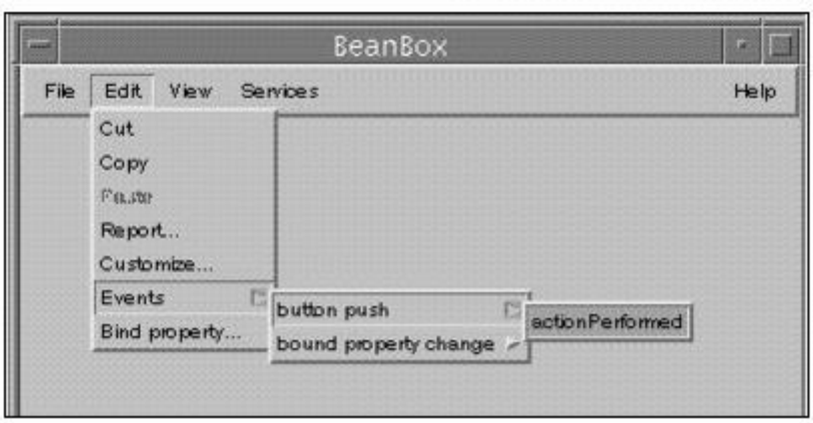
background
foreground
label press
font Abcde...

Method tracing service started.
Method Tracing service requested by:
class sunw.demo.juggler.Juggler

In the Properties sheet, change the name in the label from Press to Stop. The name on the selected button changes to Stop when you press the Return key

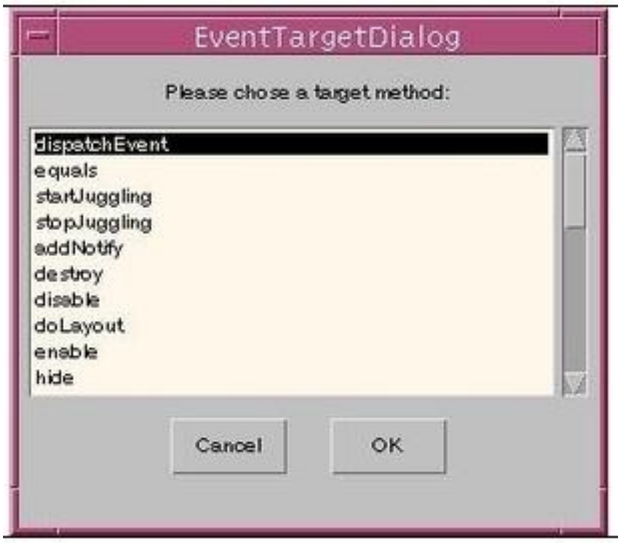
Writing up events

We can connect events from a selected bean to another bean. Let us wire up the Stop button to stop the juggler. Select the Stop button and choose Edit->Events->button push->actionPerformed.



Now as you move the mouse, a red line attached to the Stop button moves with it. Move the mouse to the Juggler bean and click the mouse button. A new EventTargetDialog window which lists available events appears.

NOTES



Please chose a target method:

Scroll down and choose stopJuggling. Click on OK. The actionPerformed event from the Stop button will now cause the stopJuggling method of the Juggler bean to be called. In order to cause this to happen, the BeanBox generates adapter code, compiles it and loads it automatically. So clicking on the Stop button now stops the Juggler.

You can similarly set up a Start button to restart the Juggler. Choose the other Press button on the canvas, rename it Start, choose Edit->Events->button push->actionPerformed and this time select startJuggling from the EventTargetDialog window.

4.4.1 Writing a Simple Bean

In this section you will learn more about creating Beans and the BeanBox. The easiest way to learn to build a Bean is to start with a basic Bean, then add a property to it. The simplest Bean must implement the java.io.Serializable interface. In fact, any Java class that implements the Serializable interface is a minimal JavaBean by default. A very simple Bean, such as one that merely displays a red rectangle, extends the AWT component java.awt.Canvas. The Bean itself needs only to define a constructor that sets the size of the rectangle and sets the background color to red.

Let's begin with a simple Bean that defines a rectangle of a certain size and background color (red) in its constructor. Within this red rectangle, the Bean displays a smaller rectangle in green. In addition to the constructor, the Bean includes a simple property called color that it initially sets to green. Notice that the color property is declared private.

The Bean includes two public methods, one to get the value of the color property and the other to set the value of color. Because the color property is private, it cannot be accessed directly; it can only be accessed via these getter and setter methods. Let's take a look at the code for a SimpleBean JavaBean:

```
import java.awt.*;
import java.io.Serializable;
public class SimpleBean extends Canvas
                implements Serializable {
  private Color color = Color.green;
  //getter method
  public Color getColor() {
          return color;
  }
  //setter method
  public void setColor(Color newColor) {
          color = newColor;
          repaint();
  }
  //override paint method
  public void paint (Graphics g)
{
          g.setColor(color);
          g.fillRect(20,5,20,30);
  }
  //Constructor: sets inherited properties
  public SimpleBean()
{
          setSize(60,40);
          setBackground(Color.red);
  }
}
```

The Serializable Interface

First, a Bean must implement the Serializable interface. Objects that support this interface can save and restore their state from disk. Beans that have been customized (usually by editing their properties in builder tools) must be able to save and restore themselves on request. To implement the Serializable interface, import the java.io.Serializable package and declare that your Bean implements Serializable in the class definition. Because this Bean is also visible, it extends the AWT component java.awt.Canvas.

NOTES

The Color Property

The color property is a private instance variable that we initialized to green:

```
private Color color = Color.green;
```

Get and Set Methods

There must be get and set methods to access each private variable. Note that the names of the getter and setter methods follow a particular format. The method names begin with either "get" or "set" and are followed by the name of the property, with the initial letter of the property name capitalized. (It is important that this format be followed so that the Bean introspection tools work.)

Thus, get and set methods have the following format:

```
public <returntype> get<Propertyname>
public void set<Propertyname> (parameter)
```

For the SimpleBean color property, we define the following pair of methods:

```
public Color getColor() { ... }
public void setColor(Color c) { ... }
```

Testing and Editing your Bean

This Bean is almost ready to be dropped into the BeanBox, where it will appear on the palette of ToolBox components.

Dropping into the BeanBox

First, you compile the Java source code. Next, create an executable JAR file. Here are the steps to do this:

Compile the source code for the Bean and generate the SimpleBean.class:

```
javac SimpleBean.java
```

Create a manifest file in a text editor.

The manifest file specifies the name of the class file and indicates that it is a JavaBean. The manifest file becomes part of the JAR file. You can name the manifest file manifest.tmp. It contains the following two lines:

```
Name: SimpleBean.class
Java-Bean: True
```

Create the executable JAR file.

Use the form of the jar command to include the manifest file along with the SimpleBean.class file (Type the command on one line):

```
jar cfm SimpleBean.jar manifest.tmp  SimpleBean.class
```

Load the JAR file into the BeanBox.

From the BeanBox File pull-down menu, select LoadJar.

This brings up a file browser.

Locate the SimpleBean.jar file and choose it. Notice that SimpleBean appears at the bottom of the list of Beans in the ToolBox.

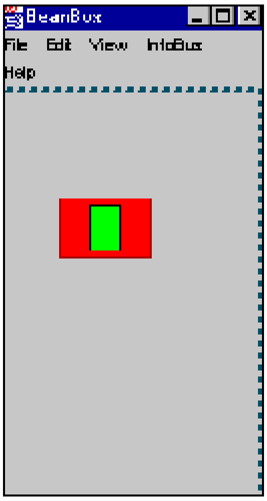
To drop the new Bean into the BeanBox, click on SimpleBean in the ToolBox.

The cursor changes to a cross hair.

Move the cursor to any spot within the BeanBox, then click.

SimpleBean appears as a painted red rectangle with a hatched border enclosing a smaller green rectangle. The hatched border indicates that SimpleBean is selected; its properties appear in the Properties sheet.

Output :



4.5 Introspection

Introspection is the automatic process that analyzes the design pattern of a bean in order to reveal bean's properties, events, and methods. This process is used to control publishing and discovery of bean operations and properties.

Java object repository sites existing on the Internet are continuously growing in numbers to answer the demand for centralized deployment of applets, classes, and source code in general. Any developer who has spent time hunting through these sites for licensable Java code to incorporate into a program has undoubtedly struggled with issues of how to quickly and cleanly integrate code from one particular source into an application.The way of implementing introspection provides great advantages, including:

Portability

Everything is java works on the principle "write once run anywhere", therefore you can write components once, reuse them everywhere. No extra specification files required to be maintained independently from your component code. You do not require any dealing with platform-specific issues.

Reuse

To provide the potential of reusing your component simply follow the JavaBeans design conventions, implement the appropriate interfaces and extend the appropriate classes to make possible to your expectations.

4.6 The BeanInfo (in the API reference documentation) interface

The JavaBeans API architecture supplies a set of classes and interfaces to provide introspection.

The BeanInfo Interface of the java.beans package defines a set of methods that allow bean implementors to provide explicit information about their beans. By specifying BeanInfo for a bean component, a developer can hide methods, specify an icon for the toolbox, provide descriptive names for properties, define which properties are bound properties, and much more.

The getBeanInfo(beanName) (in the API reference documentation) of the Introspector (in the API reference documentation) class can be used by builder tools and other automated environments to provide detailed information about a bean. The getBeanInfo method relies on the naming conventions for the bean's properties, events, and methods. A call to getBeanInfo results in the introspection process analyzing the bean’s classes and superclasses.

BeanInfo defines methods that return descriptors for each property, method, or eventthat is to be exposed in a builder tool. Here’s the prototypes for these methods:

```
PropertyDescriptor[] getPropertyDescriptors();
MethodDescriptor[] getMethodDescriptors();
EventSetDescriptor[] getEventSetDescriptors();
```

Each of these methods returns an array of descriptors for each item that is to be exposed in a builder tool.

The Introspector class provides descriptor classes with information about properties, events, and methods of a bean. Methods of this class locate any descriptor information that has been explicitly supplied by the developer through BeanInfo classes. Then the Introspector class applies the naming conventions to determine what properties the bean has, the events to which it can listen, and those which it can send.

The following example represents code to perform introspection:

```
import java.beans.BeanInfo;
import java.beans.Introspector;
import java.beans.IntrospectionException;
import java.beans.PropertyDescriptor;
```

```
public class SimpleBean
{
   private final String name = "SimpleBean";
   private int size;

   public String getName()
   {
      return this.name;
   }

   public int getSize()
   {
      return this.size;
   }

   public void setSize( int size )
   {
      this.size = size;
   }

   public static void main( String[] args ) throws
IntrospectionException
   {
      BeanInfo info = Introspector.getBeanInfo( SimpleBean.class );
      for ( PropertyDescriptor pd : info.getPropertyDescriptors() )
         System.out.println( pd.getName() );
   }
}
```

This example creates a non-visual bean and displays the following properties derived from the BeanInfo object:

```
class
name
size
```

4.7 Using Bound Properties

A bound property gives a JavaBean the capability of notifying another JavaBean when there is a change to the value of that property. The JavaBean that is notified can then react to the change. For example, you might have a button in one JavaBean that, when pressed, causes another Bean to display some information.

Because other Beans are notified when there is a change to a Bean's bound property, there has to be a means for Beans to communicate. Beans communicate using events. A change to a bound property causes an event to be fired. These events are referred to as property change events.

When you set up a bound property, you also set up other Beans as interested listeners to changes in the bound property. Then, whenever the value of the bound property of the first JavaBean changes, notification is sent to those Beans that have registered as interested listeners.

Let's look at the Bean MyButton.java first, which uses bound properties. MyButton includes code to notify other interested Beans when its properties change. Once we're done with MyButton, we'll set up a listener Bean that responds to changes in MyButton's properties.

Setting Up Bound Properties

These are the basic steps for setting up bound properties in a Bean:

Import the java.beans package to gain access to certain convenience classes defined by the package. MyButton imports the package as follows:

```
import java.beans.*;
```

Instantiate the java.beans.PropertyChangeSupport class.

```
private PropertyChangeSupport changes = new
PropertyChangeSupport(this);
```

MyButton creates a new object, called changes, which instantiates the PropertyChangeSupport class. The changes variable holds a collection of listener objects that will be notified about property changes to the bound property.

Implement the methods defined by the PropertyChangeSupport class.

The PropertyChangeSupport class includes methods to add and remove listener objects, specifically PropertyChangeListener objects. MyButton includes the following code to implement the add and remove listener methods.

Note that the parameter l refers to the property change listener Bean that registers or removes its interest as a property change listener.

```
public void addPropertyChangeListener(
        PropertyChangeListener l)
{
```

NOTES

```
        changes.addPropertyChangeListener(l);
}
public void removePropertyChangeListener(
            PropertyChangeListener l)
{
   changes.removePropertyChangeListener(l);
}
```

Modify the Bean's setter methods for the bound properties.

For those properties that you want to be bound properties, modify the Bean's setter methods to include code to fire a property change event when the property value changes.

```
        public void setFont(Font f)
         {
                Font old = getFont();
                super.setFont(f);
                sizeToFit();
                changes.firePropertyChange( "font", old, f);
        }
```

Setting Up a Property Listener

Now we set up a listener Bean that receives and perhaps responds to property change events in MyButton.

Implement the PropertyChangeListener interface. There is one method in this interface, the propertyChange method:

```
public abstract void propertyChange(PropertyChangeEvent evt)
```

Beans that fire property change events call the propertyChange method to notify property change listeners of changed properties.

In the listener Bean's implementation of the propertyChange method define the actions you want to take as a result of the property change.

You make the connection between the source Bean—the Bean that fires the property change event—and the listener Bean that reacts to the changed property in the BeanBox (or in another builder tool). You can also manually make this connection by writing a special adapter class.

For example, the listener Bean MyChangeReporter includes a method reportChange whose parameter is a PropertyChangeEvent object. The method extracts the property name and new value from the passed object, formats its text window, and displays the information.

```
public void reportChange(PropertyChangeEvent evt)
{
 String text = evt.getPropertyName()
  + " := " + evt.getNewValue();
 int width = getSize().width - 10;
 Font f = getFont();
  if (f != null)
{
  // Trim the text to fit.
    FontMetrics fm = getFontMetrics(f);
     while (fm.stringWidth(text) > width)
 {
            text = text.substring(0, text.length()-1);
  }
}
  setText(text);
}
```

You can also make the connection between the source and listener Beans in the code itself. To do this, write an adapter class to catch the property change event. You set up the adapter class to call the correct method within the listener object. Set up the listener Bean to call the listener registration method on the source Bean. For example, our listener invokes the following method on MyButton:

```
MyButton.addPropertyChangeListener(PropertyChangeListener);
```

Connecting the Beans in the BeanBox

The BeanBox recognizes when a Bean correctly defines a bound property—that it can broadcast property change events—and the BeanBox includes a propertyChange interface item for that Bean. The propertyChange interface item appears in the Events menu for the selected Bean that defines a bound property. Select Events in the Bean Edit menu. Once you select propertyChange, connect the source Bean to the listener Bean. Extending the rubber band line from the source Bean to the listener Bean. An EventTargetDialog appears. Select the appropriate listener method (in our example, this is the reportChange method).

4.8 Check Your Progress

Write True or False for the following:

1. Is Java Bean and EJB are same?
2. There are two basic kinds of session bean: stateless and stateful?

4.9 Answers to Check Your Progress Questions

1. No 2. True

4.10 Summary

In this unit we discussed about software component model. We discussed about how to write a simple bean and its properties. We discussed about Introspection and customization.

4.11 Key Words

Bean Class: This class has to implement the bean's business methods in the remote interface apart from some other callback methods. An entity bean must implement javax.ejb.EntityBean and a session bean must implement javax.ejb.SessionBean. Both EntityBean and Session Bean extend javax.ejb.EnterpriseBean.

Primary Key: This is a very simple class that provides a reference into the database. This class has to implement java.io.Serializable. Only entity beans need a primary key.

4.12 Self-Assessment Questions and Exercises

1. What are Java Beans?
2. Difference between java bean and bean?
3. What is introspection API in java beans?
4. What is the purpose of introspection?
5. What are the properties of java beans?
6. What are externizable interface?
7. How to control serialization in java beans?
8. What are introspection properties in java beans?
9. What is the serializable class in java beans?

10. How to create bound property in bean application?

4.13 Further Readings

1. Tom Valesky, Enterprise Java Beans, Pearson Education inc, 2008
2. Robert Englander, Developing Java Beans, O'Reilly Media

UNIT 5

CONSTRAINED PROPERTIES and EJB

Structure

5.0 Introduction

Constrained properties are bound properties that go one step further. With a constrained property, an outside object--either a listener Bean or the source Bean itself--can veto a property change. Customization provides a means for modifying the appearance and behavior of a bean within an application builder so it meets your specific needs. The mechanism that makes persistence possible is called serialization. Object serialization means converting an object into a data stream and writing it to storage.

Enterprise JavaBeans (EJB) is a comprehensive technology that provides the infrastructure for building enterprise-level server-side distributed Java components.

5.1 Objectives

After going through the unit, you will be able to know about;

- Various Constrained properties
- Bean Customization
- Bean Persistence
- Controlling serialization
- Understanding Serialization Interface
- Externalization Interface
- Java Beans API
- Introduction to EJB's

5.2 Constrained Properties

. The JavaBeans API provides an event mechanism for handling constrained properties that is similar to the one used for bound properties.

To implement a constrained property, you must have:

A source Bean that defines a constrained property.

Listener objects that implement the VetoableChangeListener interface.

A PropertyChangeEvent object that contains the property name, its old value, and its new value. (Note that this is the same object used by bound properties.)
Setting Up Constrained Properties

A source Bean sets up constrained properties as follows:

Implements a mechanism to allow listener objects who have implemented the VetoableChangeListener interface to register and unregister their interest in receiving property change notification. Rather than receive notification that a property has already changed, these listeners receive notification that a change to a property has been proposed.

The Bean accomplishes this using the VetoableChangeSupport utility class, which it either inherits or instantiates. The utility class includes methods to add and remove VetoableChangeListener objects to a listener list, plus a method that fires property change events and also catches and responds appropriately to veto responses.

fireVetoableChange(<property name>,

```
     <old value>, <new value>);
      addVetoableChangeListener(listener);
    removeVetoableChangeListener(listener);
```

Fires proposed property change events to interested listeners. Because these changes may be vetoed, the source Bean fires the event before the change takes place. Source Beans fire the PropertyChangeEvent by calling the listener's vetoableChange method.

Provides a means for listener objects to keep the original property value should any one of the listeners veto the change. To do this, the source Bean issues the vetoableChange call to all listeners a second time, and passes the listeners the original property value.

Example

The JellyBean JavaBean implements a constrained property, which it calls priceInCents. The Bean begins by instantiating two new objects: a VetoableChangeSupport object vetos to maintain the list of vetoable listeners and a PropertyChangeSupport object changes to hold the list of property change listeners. It does this as follows:

```
private VetoableChangeSupport vetos =
        new VetoableChangeSupport(this);
private PropertyChangeSupport changes =
         new PropertyChangeSupport(this);
```

Then, JellyBean implements the VetoableChangeSupport interface methods to register and unregister constrained property change listeners:

```
public void addVetoableChangeListener(
            VetoableChangeListener l) {
          vetos.addVetoableChangeListener(l);
}
public void removeVetoableChangeListener(
             VetoableChangeListener l) {
    vetos.removeVetoableChangeListener(l);
}
```

The above add and remove listener methods apply to all constrained properties of the Bean. It's possible to specify that the listener be attuned only to a specific constrained property within the Bean. If so, these two methods can include an additional String parameter containing the property name. The property name appears first in the parameter list, followed by the listener object. For example, to add a listener Bean for a specific property:

```
public void addVetoableChangeListener(
                String propertyName,
                      VetoableChangeListener
                              listener);
```

You can also register and unregister vetoable change listeners on a per property basis by specifying the particular property name, as follows:

```
void add<PropertyName>Listener(
          VetoableChangeListener p);
void remove<PropertyName>Listener(
          VetoableChangeListener p);
```

JellyBean also defines a method to get the current price and another method to set a new price. In the method to set the new price, JellyBean invokes the method fireVetoableChange to notify registered listeners of the property change.

```
public void setPriceInCents(int newPriceInCents)
      throws PropertyVetoException {
        vetos.fireVetoableChange("priceInCents",
        new Integer(oldPriceInCents),
        new Integer(newPriceInCents));
        //if vetoed (PropertyVetoException thrown)
        // don't catch
        // exception here so routine exits.
        //if not vetoed, make the property
        // change
        ourPriceInCents = newPriceInCents;
        changes.firePropertyChange
(
                "priceInCents",
        new Integer(oldPriceInCents),
        new Integer(newPriceInCents));
}
```

Setting Up Constrained Property Listeners

Constrained property listener objects implement the VetoableChangeListener interface, which includes the vetoableChange method. When the source Bean fires a constrained property change event, it calls vetoableChange method on each registered listener. The listener uses this method to receive the change event and to accept or reject proposed changes. If they reject a proposed change, they throw a PropertyVetoException.

```
public void vetoableChange(
                    PropertyChangeEvent x)
                 hrows PropertyVetoException {
          if (vetoAll) {
          throw new PropertyVetoException( "NO!", x);
          }
}
```

Constrained Properties in the BeanBox

The BeanBox handles constrained properties very much like bound properties.

- Drop the Bean that defines the constrained property into the BeanBox. Our example uses the JellyBean.
- Drop in a Bean that listens for changes to that property and that can potentially veto such a change. In our example, we use the Voter Bean, which vetoes changes to that property.
- Choose the vetoableChange event for the JellyBean. Select the JellyBean and, from the Edit menu, select the Events > vetoableChange > vetoableChange menu item
- Connect the event to the listener Bean and select its vetoableChange method.
- Connect the rubber band to the Voter Bean instance. When the EventTargetDialog panel appears, select the vetoableChange method.
- The BeanBox generates the event hookup adapter.
- Test the constrained property.

Select the JellyBean and, in the Property sheet, try changing its priceInCents property. In the terminal window from which you started the BeanBox, you see a message indicating that an exception was thrown and the change is not allowed.

5.3 Bean Customization

Customization provides a means for modifying the appearance and behavior of a bean within an application builder so it meets your specific needs. There are several levels of customization available for a bean developer to allow other developers to get maximum benefit from a bean’s potential functionality.

A bean's appearance and behavior can be customized at design time within beans-compliant builder tools. There are two ways to customize a bean:

By using a property editor. Each bean property has its own property editor. The NetBeans GUI Builder usually displays a bean's property editors in the Properties window. The property editor that is associated with a particular property type edits that property type.

By using customizers.

Customizers give you complete GUI control over bean customization. Customizers are used where property editors are not practical or applicable. Unlike a property editor, which is associated with a property, a customizer is associated with a bean.

When you use a bean Customizer, you have complete control over how to configure or edit a bean. A Customizer is an application that specifically targets a bean's customization. Sometimes properties are insufficient for representing a bean's configurable attributes. Customizers are used where sophisticated instructions would be needed to change a bean, and where property editors are too primitive to achieve bean customization.

All customizers must:

- Extend java.awt.Component or one of its subclasses.
- Implement the java.beans.Customizer interface This means implementing methods to register PropertyChangeListener objects, and firing property change events at those listeners when a change to the target bean has occurred.
- Implement a default constructor.
- Associate the customizer with its target class via BeanInfo.getBeanDescriptor.

5.4 Bean Persistence

A bean has the property of persistence when its properties, fields, and state information are saved to and retrieved from storage. Component models provide a mechanism for persistence that enables the state of components to be stored in a non-volatile place for later retrieval.

The mechanism that makes persistence possible is called serialization. Object serialization means converting an object into a data stream and writing it to storage. Any applet, application, or tool that uses that bean can then "reconstitute" it by deserialization. The object is then restored to its original state.

For example, a Java application can serialize a Frame window on a Microsoft Windows machine, the serialized file can be sent with e-mail to a Solaris machine, and then a Java application can restore the Frame window to the exact state which existed on the Microsoft Windows machine.

Any applet, application, or tool that uses that bean can then "reconstitute" it by deserialization.

All beans must persist. To persist, your beans must support serialization by implementing either the java.io.Serializable(in the API reference documentation) interface, or the java.io.Externalizable(in the API reference documentation) interface. These interfaces offer you the choices of automatic serialization and customized serialization. If any class in a class's inheritance hierarchy implements Serializable or Externalizable, then that class is serializable.

5.4.1 Controlling Serialization

You can control the level of serialization that your Beans undergo:

Automatic: implement Serializable. Everything gets serialized.Selectively exclude fields you do not want serialized by marking with the transient (or static) modifier
Complete control: implement Externalizable, and its two methods.

5.4.2 The Serializable Interface

Default Serialization: **The Serializable Interface**

The Serializable interface provides automatic serialization by using the Java Object Serialization tools. Serializable declares no methods; it acts as a marker, telling the Object Serialization tools that your Bean class is serializable. Marking your class with Serializable means you are telling the JVM that you have made sure your class will work with default serialization. Here are some important points about
working with the Serializable interface:

- Classes that implement Serializable must have a no−argument constructor. This constructor will be called when an object is "reconstituted" from a .ser file.
- There is no need to implement Serializable in your class if if it is already implemented in a superclass.
- All fields but static and transient are serialized. Use the transient modifier to specify fields you do not want serialized, and to specify classes that are not serializable.

The BeanBox writes serialized Beans to a file with a .ser extension. The OurButton demo Bean uses default serialization to make its properties persist. OurButton only added Serializable to its class definition to make use of default

Serialization:

public class OurButton extends Component implements Serializable,...

If you drop an OurButton instance into the BeanBox, the properties sheet displays OurButton's properties. To ascertain that serialization is working

1. Change some OurButton properties. For example change the font size and colors.
2. Serialize the changed OurButton instance by selecting the File|SerializeComponent... BeanBox menu item. A file browser will pop up.
3. Put the .ser file in a JAR file with a suitable manifest (need to explain how to do this).
4. Clear the BeanBox form by selecting the File|Clear menu item.
5. Reload the serialized instance by selecting the File|LoadJar menu item.

The OurButton instance will appear in the BeanBox with your property changes intact. By implementing Serializable in your class, simple, primitive properties and fields can be serialized. For more complex class members, different techniques must be used.

Selective Serialization Using the transient Keyword

To exclude fields from serialization in a Serializable object from serialization, mark the fields with the transient modifier.

```
transient int Status;
```

Default serialization will not serialize transient and static fields.

Selective Serialization: writeObject and readObject()

If your serializable class contains either of the following two methods (the signatures must be exact), then the default serialization will not take place.

private void writeObject(java.io.ObjectOutputStream out) throws IOException;

private void readObject(java.io.ObjectInputStream in) throws IOException, ClassNotFoundException;

You can control how more complex objects are serialized, by writing your own implementations of the writeObject() and readObject() methods. Implement writeObject when you need to exercise greater control over what gets serialized, when you need to serialize objects that default serialization cannot handle, or when you need to add data to the serialization stream that is not an object data member.

Implement readObject() to reconstruct the data stream you wrote with writeObject().

Example: The Molecule Demo Bean

The Molecule demo keeps a version number in a static field. Since static fields are not serialized by default, writeObject() and readObject() are implemented to serialize this field. Here is the writeObject() and readObject() implementations in Molecule.java:

```
private void writeObject(java.io.ObjectOutputStream s)
throws java.io.IOException
{
s.writeInt(ourVersion);
s.writeObject(moleculeName);
}
private void readObject(java.io.ObjectInputStream s)
throws java.lang.ClassNotFoundException,java.io.IOException
{
// Compensate for missing constructor.
reset();
if (s.readInt() != ourVersion)
 {
throw new IOException("Molecule.readObject: version mismatch");
}
moleculeName = (String) s.readObject();
}
```

These implementations limit the fields serialized to our Version and moleculeName.Any other data in the class will not be serialized. It is best to use the ObjectInputStream’s defaultWriteObject() and defaultReadObject before doing your own specific stream writing. For example:

```
private void writeObject(java.io.ObjectOutputStream s)
throws java.io.IOException
{
//First write out defaults
s.defaultWriteObject();
//...
}
private void readObject(java.io.ObjectInputStream s)
throws java.lang.ClassNotFoundException,
java.io.IOException {
//First read in defaults
s.defaultReadObject();
//...
}
```

5.4.3 The Externalizable Interface

Use the Externalizable interface to serialize (write) your Beans in a specific file format. Externalizable gives you complete flexibility in specifying the serialized format. You need to implement two methods: readExternal() and writeExternal(). Externalizable classes must also have a no−argument constructor.

Example: The BlueButton and OrangeButton Demo Beans

When you run the BeanBox, you will see two Beans named BlueButton and OrangeButton in the ToolBox. These two Beans are actually serialized instances of the ExternalizableButton class.

ExternalizableButton implements the Externalizable interface. This means it does all its own serialization, by implementing Externalizable.readExternal() and Externalizable.writeExternal(). The BlueButtonWriter program is used by the buttons makefile to create an ExternalizableButton instance, change its background property to blue, and write the Bean out to the file BlueButton.ser.

OrangeButton is created the same way, using OrangeButtonWriter. The button makefile then puts these .ser files in buttons.jar, where the ToolBox can find and reconstitute them.

5.5 The JavaBeans API

The JavaBeans API includes several interfaces and classes in the java.beans package. In addition, it employs interfaces and classes from other Java technology API areas including:

The Java event model: java.util.EventObject, java.awt.event
*Object serialization: java.io.Serializable, java.io.Object**
Reflection: java.lang.reflect

JDK 1.1 introduced the subject-observer or source-target event model. JDK 1.1 provides base-level support for this event model outside the AWT package, specifically, in the java.util package. The relevant interface, class, and exception are java.util.EventListener, java.util.EventObject, and java.util.TooManyListenersException.

With this approach, programmers can use event-driven communication strategies above and beyond either GUI or JavaBeans contexts, for example, thermostats communicating with heating and cooling unit controllers. Ideally, however, nongraphical event-driven scenarios should take advantage of the JavaBeans framework, because the JavaBeans component API is now well known and supported by many IDEs.

In most cases, however, the JavaBeans API is used for designing graphical components. The source-target event model is ideally suited for graphical events such as mouse pointer motion, mouse button clicks, key activity, and high-level component operations such as pressing a command button and selecting items from a drop-down list. In fact, all AWT components employ the JavaBeans framework for asynchronous event communication, as well as for configuration support; that is, all AWT components are fully compliant JavaBeans components. While the AWT components include support for the original JDK 1.0 event model, components created to use the JavaBeans framework cannot use the older model.

Object serialization is another core Java API area that is indispensable for basic JavaBeans functionality. When programmers assemble, configure, and connect Beans using an IDE, the JavaBeans components must be "live," dynamically created objects. The IDE must be able to save the worksheet's state at the end of the day and restore it at the start of a subsequent session. That is, a Bean's state must persist via external storage.

JavaBeans components, like all other user-defined data types in the

Java environment, support persistence by implementing the Serializable interface. Serializable is a tagging interface; that is, it marks an object as suitable for serialization by the Java runtime environment.

Before attempting to write an object to disk (or send it over the network) as a serialized byte stream, the Java interpreter verifies that the object implements Serializable. Take for example, a worksheet such as the one with connected, operational Progress and ProgressTarget Beans shown in the previous example. For an IDE to save the state of the worksheet, it would use selected java.io.Object* classes to serialize the subsequently deserialize the worksheet contents (along with other IDE- and project-related information).

Reflection is the third indispensable API (java.lang.reflect) for the JavaBeans architecture. With reflection, it's straightforward to examine any object dynamically, to determine (and potentially invoke) its methods.

5.6 Introduction to Enterprise JavaBeans (EJB)

The EJB technology provides a distributed component architecture that integrates several enterprise-level requirements such as distribution, transactions, security, messaging, persistence, and connectivity to mainframes and Enterprise Resource Planning (ERP) systems.

Thus, an Enterprise Java Bean is a remote object with semantics specified for creation, invocation and deletion. The EJB container is assigned the system-level tasks mentioned above. What a web container does for Java servlets and JSPs in a web server, the EJB container is for EJBs.

EJB Architecture

Any distributed component technology should have the following requirements:

1. There should be a mechanism to create the client-side and server-side proxy objects. A client-side proxy represents the server-side object on the client-side. As far as the client is concerned, the client-side proxy is equivalent to the server-side object. On the other hand, the purpose of the server-side proxy is to provide the basic infrastructure to receive client requests and delegate these request to the actual implementation object

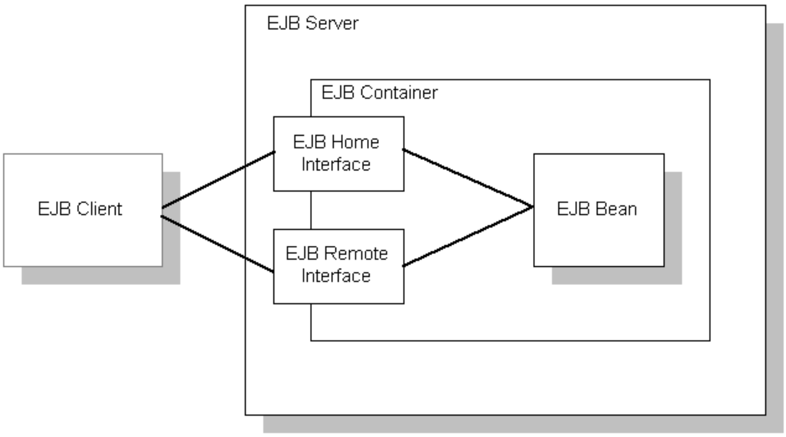
2. We need to obtain a reference to client-side proxy object. In

order to communicate with the server-side object, the client needs to obtain a reference to the proxy.

3. There should be a way to inform the distributed component system that a specific component is no longer in use by the client.

In order to meet these requirements, the EJB architecture specifies two kinds of interfaces for each bean. They are home interface and remote interface. These interfaces specify the bean contract to the clients. However, a bean developer need not provide implementation for these interfaces. The home interface will contain methods to be used for creating remote objects. The remote interface should include business methods that a bean is able to serve to clients. One can consider using the home interface to specify a remote object capable of creating objects conforming to the remote interface. That is, a home interface is analogous to a factory of remote objects. These are regular Java interfaces extending the javax.ejb.EJBHome and javax.ejb.EJBObject interfaces respectively.

As discussed below, the EJB architecture specifies three types of beans - session beans, entity beans, and message-driven beans. A bean developer has to specify the home and remote interfaces and also he has to implement one of these bean interfaces depending upon the type of the bean. For instance, for session beans, he has to implement the javax.ejb.SessionBean interface. The EJB architecture expects him to implement the methods specified in the bean interface and the methods specified in the home and remote interfaces. During the deployment time, he should specify the home and remote interfaces and bean implementation class to define a bean. The EJB container relies on specific method names and uses delegation for invoking methods on bean instances.



EJB Architecture

Thus, regarding the first requirement, the EJB container generates the proxy objects for all beans. For the second one, the EJB container for each bean implement a proxy object to the home interface and publishes in the JNDI implementation of the J2EE platform. One can use JNDI to look for this and obtain a reference. As this object implements the home interface only, he can use one of the creation methods of the home object to get a proxy to the remote interface of the bean. When one invokes a creation method on the home proxy object, the container makes sure that a bean instance is created on the EJB container runtime and its proxy is returned to the client. Once the client gets hold of the proxy for the remote interface, it can directly access the services of the bean.

Finally, once the client decides to stop accessing the services of the bean, it can inform the EJB container by calling a remote method on the bean. This signals the EJB container to disassociate the bean instance from the proxy and that bean instance is ready to service any other client

Types of Enterprise JavaBeans

Enterprise JavaBeans server-side components come in two fundamentally different types:

- entity beans and
- session beans.

Basically, entity beans model business concepts that can be expressed as nouns. For example, an entity bean might represent a customer, a piece of equipment, an item in inventory. Thus entity beans model real-world objects. These objects are usually persistent records in some kind of database.

Session beans are for managing processes or tasks. A session bean is mainly for coordinating particular kinds of activities. That is, session beans are plain remote objects meant for abstracting business logic. The activity that a session bean represents is fundamentally transient. A session bean does not represent anything in a database, but it can access the database.

Thus an entity bean has persistent state whereas a session bean models interactions but does not have persistent state.

Session Beans

Session beans are transaction-aware. In a distributed component environment, managing transactions across several components mandates distributed transaction processing. The EJB architecture allows the container to manage transactions declaratively. This mechanism lets a bean developer to specify transactions across bean methods. Session beans are client-specific. That is, session bean instances on the server side are specific to the client that created them on the client side. This eliminates the need for the developer to deal with multiple threading and concurrency.

Session beans can be either stateful or stateless.

Stateful session beans maintain conversational state when used by a client. Conversational state is not written to a database but can store some state in private variables during one method call and a subsequent method call can rely on this state. Maintaining a conversational state allows a client to carry on a conversation with a bean. As each method on the bean is invoked, the state of the session bean may change and that change can affect subsequent method calls.

Stateless session beans do not maintain any conversational state. Each method is completely independent and uses only data passed in its parameters. One can specify whether a bean is stateful or not in the bean's deployment descriptor.

Entity Beans

Unlike session beans, entity beans have a client-independent identity. This is because an entity bean encapsulates persistent data. The EJB architecture lets a developer to register a primary key class to encapsulate the minimal set of attributes required to represent the identity of an entity bean. Clients can use these primary key objects to accomplish the database operations, such as create, locate, or delete entity beans. Since entity beans represent persistent state, entity beans can be shared across different clients. Similar to session beans, entity beans are also transactional, except for the fact that bean instances are not allowed to programmatically control transactions.

There are two types of entity beans and they are distinguished by how they manage persistence.

Container-managed

Beans have their persistence automatically managed by the EJB container. This is a more sophisticated approach and here the bean developer does not implement the persistence logic. The developer relies on the deployment descriptor to specify attributes whose persistence should be managed by the container. The container knows how a bean instance's fields map to the database and automatically takes care of inserting, updating, and deleting the data associated with entities in the database.

Bean Managed

Beans using bean-managed persistence do all this work explicitly: the bean developer has to write the code to manipulate the database. The EJB container tells the bean instance when it is safe to insert, update, and delete its data from the database, but it provides no other help. The bean instance has to do the persistence work itself.

Writing Stateful Session Bean

A Session Bean is composed of the following parts:

1. Remote Interface: The Remote Interface is the client view of the bean.
2. Home Interface: The Home Interface contains all the methods for the bean life cycle (creation, suppression) used by the client application.
3. Bean Class: The bean implementation class implements the business methods.
4. Deployment Descriptor: The deployment descriptor contains the bean properties that can be edited at assembly or deployment time.

Steps involved in developing the Stateful Session Bean can be summarized in following steps:

- Define Home Interface
- Define Remote Interface
- Develop EJB class
- Write deployment descriptors
- Package, deploy and test the application
- Define Home Interface

The Session bean's home interface defines one or more create(...) methods and each create method must be named create and must match one of the ejbCreate methods defined in the enterprise Bean class. The return type of a create method is the Bean's remote interface type.

In case of Stateless Session Bean there is one create method with no arguments. A remote home interface extends the javax.ejb.EJBHome interface, while a local home interface extends the javax.ejb.EJBLocalHome interface.

Here is the source code of our home interface:

```
/*
 * TestSessionBeanHome.java
 */
package examples;

/**
 * Home interface for TestSessionBean.
 */

public interface TestSessionBeanHome
   extends javax.ejb.EJBHome
{
   public examples.TestSessionBean create()
      throws javax.ejb.CreateException,java.rmi.RemoteException;

}
```

Define Remote Interface

As mentioned earlier, Remote Interface is the client view of the bean and the functions defined in the remote interface is visible to the client. There should be an implementation in the Bean class for the all the functions defined in the remote interface.

Here is the source code of our Remote Interface:

```
/*
* TestSessionBean.java
*
*/
package examples;

/**
* Remote interface for TestSessionBean.
*/

public interface TestSessionBean
   extends javax.ejb.EJBObject
{
   /**
    * The method that returns Hello Message
    */
   public java.lang.String SayHello(  )
      throws java.rmi.RemoteException;
```

}

Define Enterprise Bean Class

In the EJB class we write all the business methods along with required that is necessary to implement. In the EJB class methods are defined public. The Session Bean interface methods that the EJB provider must must be defined in the Session bean class are:

public void setSessionContext(SessionContext ic);

This method is used by the container to pass a reference to the SessionContext to the bean instance.

public void ejbRemove();

This method is invoked by the container when the instance is in the process of being removed by the container.

public void ejbPassivate();

This method is invoked by the container when it wants to passivate the instance.

public voidejbActivate();

This method is invoked by the container when the instance has just been reactivated.

A stateful session Bean with container-managed transaction demarcation can optionally also implements the javax.ejb.SessionSynchronization interface.

The Session Synchronization interface methods that must be developed are:

public void afterBegin();

This method notifies a session Bean instance that a new transaction has started.

public void afterCompletion(boolean committed);

This method notifies a session Bean instance that a transaction commit protocol has completed and tells the instance whether the transaction has been committed or rolled back.

public void beforeCompletion();

This method notifies a session Bean instance that a transaction is about to be committed. Here is the source code of our Session Bean class:

```
/*
* SessionBean.java
*
*/
package examples;

import java.rmi.RemoteException;
import javax.ejb.CreateException;
import javax.ejb.EJBException;
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;
public class MyTestSessionBean implements SessionBean
{
  public void ejbCreate() throws CreateException
{
  }
  public void setSessionContext( SessionContext aContext ) throws
EJBException {
  }
  public void ejbActivate() throws EJBException
{
  }
  public void ejbPassivate() throws EJBException
{
  }
  public void ejbRemove() throws EJBException
 {
  }
  /**
  *  The method that returns Hello Message
  *
  */
  public String SayHello()
{
    String msg="Hello! I am Session Bean";
    System.out.println(msg);
    return msg;
  }
}
```

5.7 Check Your Progress

Write True or False for the following:

1. EJB2.0 components are based on distributed objects?

5.8 Answers to check Your Progress

1. True

5.9 Summary

We discussed about introduction to Enterprise java beans, enterprise java beans architecture and the types of Enterprise java beans. The JavaBeans API includes several interfaces and classes in the java.beans package. Enterprise JavaBeans (EJB) is a comprehensive technology that provides the infrastructure for building enterprise-level server-side distributed Java components.

5.10 Glossary

EJB Container: The environment that surrounds the beans on the EJB server is often referred to as the container. The container acts as an intermediary between the bean class and the EJB server. The container manages the EJB objects and EJB homes for a particular type of bean and helps these constructs to manage bean resources and apply the primary services relevant to distributed systems to bean instances at run time. An EJB server can have more than one container and each container in turn can accommodate more than one enterprise bean.

Remote Interface: This interface for an enterprise bean defines the enterprise bean's business methods that clients for this bean can access. The remote interface extends javax.ejb.EJBObject, which in turn extends java.rmi.Remote.

Home interface: This interface defines the bean's life cycle methods such as creation of new beans, removal of beans, and locating beans. The home interface extends javax.ejb.EJBHome, which in turn extends java.rmi.Remote.

Deployment Descriptors: Much of the information about how beans are managed at runtime is not supplied in the interfaces and classes mentioned above. There are some common primary services related with distributed systems apart from some specific services such as security, transactions, naming that are being handled automatically by EJB server. But still EJB server needs to know beforehand how to apply the primary services to each bean class at runtime. Deployment descriptors exactly do this all important task.

JAR Files: Jar files are ZIP files that are used specifically for packaging Java classes that are ready to be used in some type of application. A Jar file containing one or more enterprise beans includes the bean classes, remote interfaces, home interfaces, and primary keys for each bean. It also contains one deployment descriptor.

Deployment is the process of reading the bean's JAR file, changing or adding properties to the deployment descriptor, mapping the bean to the database, defining access control in the security domain, and generating vendor-specific classes needed to support the bean in the EJB environment. Every EJB server product comes with its own deployment tools containing a graphical user interface and a set of command-line programs.

5.11 Self-Assessment Questions and Exercises

1. What is Enterprise JavaBean?
2. What is the relationship between Enterprise JavaBeans and JavaBeans?
3. What is an entity?
4. What is EJB object?
5. How to implement a bound property in your bean application?
6. What is entity reference?
7. What is entity bean?
8. What is bean customization?
9. Why are component architectures useful?
10. Why a component architecture for the Java platform?
11. Difference between java Beans and Enterprise Java Beans?
12. What is EJB server?

5.12 Further Readings

1. Andrew Lee Rubinger, and Bill Burge, Enterprise Java Beans 3.1. 6th edition,O'Reilly Sep 2010.
2. Ed Roman, Mastering Enterprise JavaBeans and the Java 2 platform, Enterprise edition,Wiley Computer Publishing.

NOTES

BLOCK – III
SERVLETS

UNIT 6
Web Servers and Servlets

Structure

6.0 Introduction

In client- server computing, each application had its own client program and it worked as a user interface and need to be installed on each user's personal computer. Most web applications use HTML/XHTML that are mostly supported by all the browsers and web pages are displayed to the client as static documents. A web page can merely displays static content and it also lets the user navigate through the content, but a web application provides a more interactive experience.

Any computer running Servlets or JSP needs to have a container. A container is nothing but a piece of software responsible for loading, executing and unloading the Servlets and JSP. While servlets can be used to extend the functionality of any Java- enabled server. They are mostly used to extend web servers, and are efficient replacement for CGI scripts.

Java Servlet is a generic server extension that means a java class can be loaded dynamically to expand the functionality of a server. Servlets are used with web servers and run inside a Java Virtual

Machine (JVM) on the server so these are safe and portable. Unlike applets they do not require support for java in the web browser. Unlike CGI, servlets don't use multiple processes to handle separate request. Servets can be handled by separate threads within the same process. Servlets are also portable and platform independent.

6.1 Objectives

After going through this lesson, you would be able to:

- Know about the web servers and its types
- How to install and use TomCat webserver
- Understand the Servlet API
- Know about cookies and session Tracking
- Understand Handling of security issues in servlets

6.2 Web Server

A web server is also known as an HTTP Server. It responds to request from a web browser by returning Html images, applets or other data. It is also responsible for enforcing security policies, storing frequently requested files in cache memory logging request and much more. Example web server is Tomcat web server.

6.3 Tomcat Web Server

Tomcat is an open source web server developed by Apache Group. Apache Tomcat is the servlet container that is used in the official Reference Implementation for the Java Servlet and JavaServer Pages technologies.

Here we are illustrating the installation process only for Windows. Steps involved in installation and configuration process for Tomcat 6.0.10 are illustrated below:

Step 1:

Installation of JDK: Don't forget to install JDK on your system (if not installed) because any tomcat requires the Java 1.5 (Java 5) and Java 1.6 (Java 6) and then set the class path (environment variable) of JDK.

Step 2:

Setting the class path variable for JDK: Two methods are there to set the classpath.

1. Set the class path using the following command.

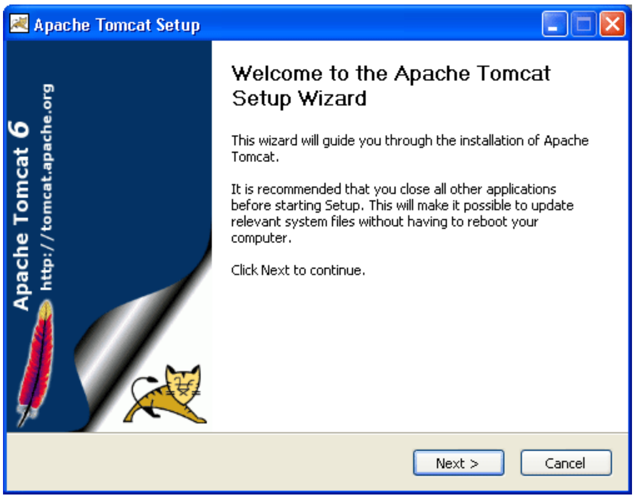
```
set PATH="C:\Program Files\Java\jdk1.5.0_08\bin";%PATH%
```

2. The other way of setting the class path variable is:

First right click on the My Computer->properties->advance->Environment Variables->path. Set bin directory path of JDK in the path variable.

Step 3:

Now it's time to shift on to the installation process of Tomcat 6.0.10. It takes various steps for installing and configuring the Tomcat 6.0.For Windows, Tomcat comes in two forms : .zip file and the Windows installer (.exe file). Here we are exploring the installation process by using the .exe file. The directory C:\apache-tomcat-6.0.10 is the common installation directory as it is pre-specified C:\ as the top-level directory. First unpack the zipped file and simply execute the .exe file.



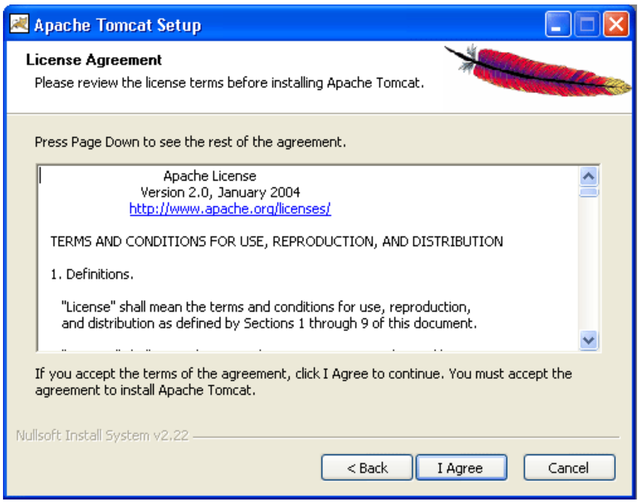
Welcome to the Apache Tomcat Setup Wizard

This wizard will guide you through the installation of Apache Tomcat.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

The above shown screen shot is the first one shown in the installation process. Just click on the Next button to proceed the installation process.



License Agreement

Please review the license terms before installing Apache Tomcat.

Press Page Down to see the rest of the agreement.

Apache License
Version 2.0, January 2004
http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

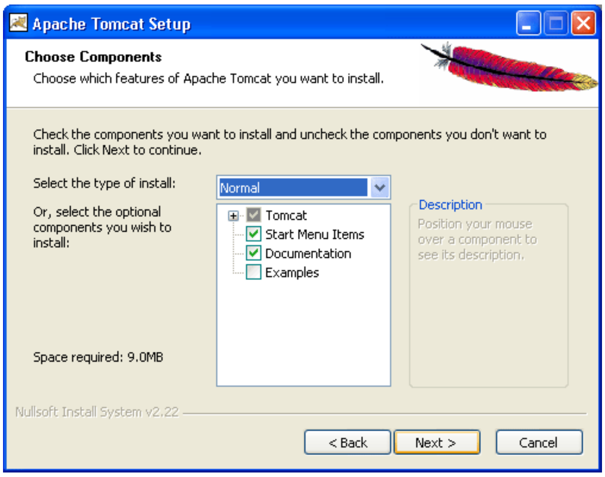
1. Definitions.

"License" shall mean the terms and conditions for use, reproduction, and distribution as defined by Sections 1 through 9 of this document.

If you accept the terms of the agreement, click I Agree to continue. You must accept the agreement to install Apache Tomcat.

Nullsoft Install System v2.22

click "I Agree" button to continue the installation process.



Choose Components

Choose which features of Apache Tomcat you want to install.

Check the components you want to install and uncheck the components you don't want to install. Click Next to continue.

Select the type of install: Normal

Or, select the optional components you wish to install:

- ☑ Tomcat
- ☑ Start Menu Items
- ☑ Documentation
- □ Examples

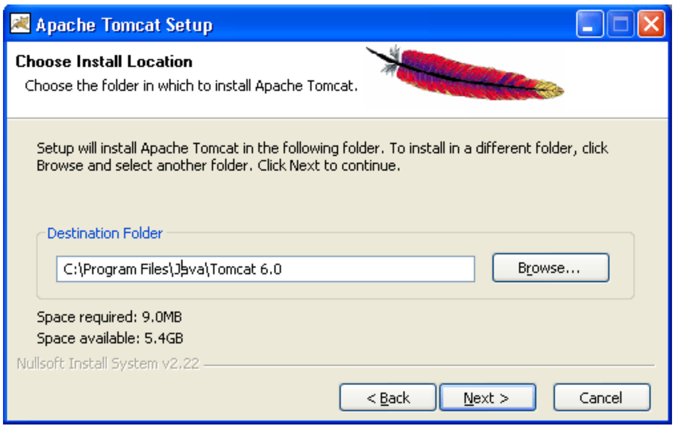
Description

Position your mouse over a component to see its description.

Space required: 9.0MB

Nullsoft Install System v2.22

Click next to go with the default components choosen.



Choose Install Location

Choose the folder in which to install Apache Tomcat.

Setup will install Apache Tomcat in the following folder. To install in a different folder, click Browse and select another folder. Click Next to continue.

Destination Folder

C:\Program Files\Java\Tomcat 6.0

Space required: 9.0MB
Space available: 5.4GB

Nullsoft Install System v2.22

Choose the location for the Tomcat files as per your convenience. You can also choose the default location.

Now choose the port number on which you want to run the tomcat server. Tomcat uses the port number 8080 as its default value. But Most of the people change the port number to 80 because in this case the user is not required to specify the port number at request time. But we are using here the default port number as 8080. Choose the user name and password as per your convenience. We can change the port number even the installation process is over. For that, go to the specified location as " Tomcat 6.0 \conf \server.xml ". Within the server.xml file choose "Connector" tag and change the port number.

e.g While using the port number 8080, give the following request in the address bar as:

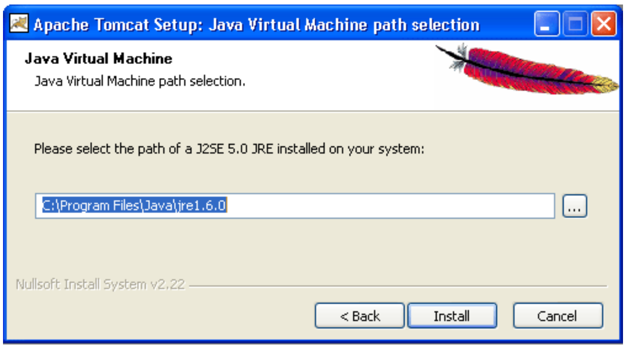
Default Port: http//localhost:8080/index.jsp

In case of port number number 80 just type the string illustrated below in the address bar:

New Port: http://localhost/index.jsp

Note that we do no need to specify any port number in the URL.

Now click on the Next button to proceed the installation process.



Apache Tomcat Setup: Java Virtual Machine path selection

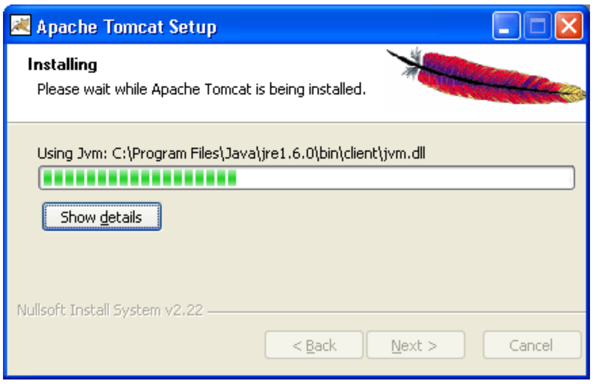
Java Virtual Machine

Java Virtual Machine path selection.

Please select the path of a J2SE 5.0 JRE installed on your system:

C:\Program Files\Java\jre1.6.0

The installation process shows the above screen as the next window. This window asks for the location of the installed Java Virtual Machine. Browse the location of the JRE folder and click on the Install button. This will install the Apache tomcat at the specified location.



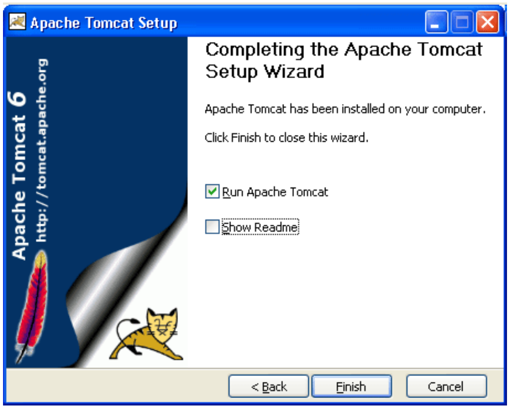
Apache Tomcat Setup

Installing

Please wait while Apache Tomcat is being installed.

Using Jvm: C:\Program Files\Java\jre1.6.0\bin\client\jvm.dll

To get the information about installer click on the "Show details" button.



Completing the Apache Tomcat Setup Wizard

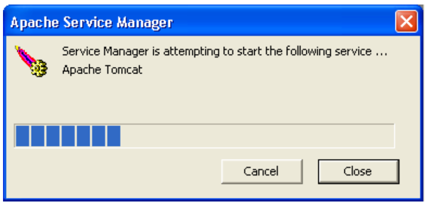
Apache Tomcat has been installed on your computer.

Click Finish to close this wizard.

☑ Run Apache Tomcat

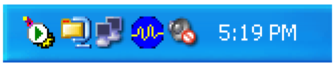
□ Show Readme

After completion of installation process it will display the window like the above one.

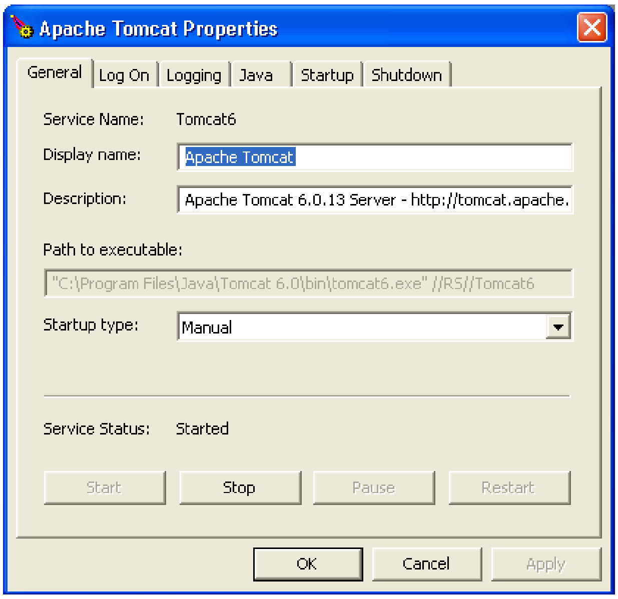


Service Manager is attempting to start the following service ...
Apache Tomcat

On clicking at Finish button, a window like the above one will display a message printed on the window given below.



After successfully installing, a shortcut icon to start the tomcat server appears in the icon tray of the task bar as shown above. Double clicking the icon displays the window of Apache Manager for Tomcat. It will show the "Startup type" as manual since we have changed the destination folder for tomcat during the installation process. Now we can configure the other options like "Display name" and "Description" .We can also start, stop and restart the service from here.



Service Name: Tomcat6

Display name: Apache Tomcat

Description: Apache Tomcat 6.0.13 Server - http://tomcat.apache.

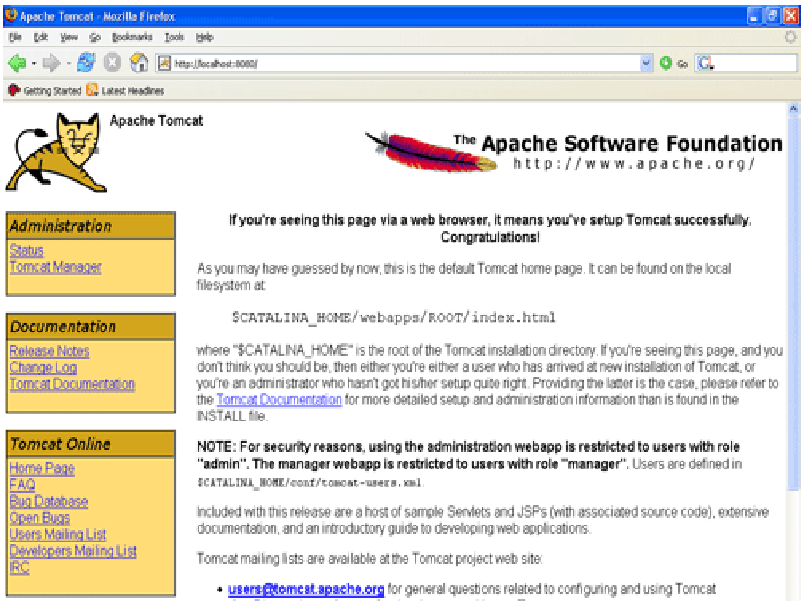
Path to executable:
"C:\Program Files\Java\Tomcat 6.0\bin\tomcat6.exe" //RS//Tomcat6

Startup type: Manual

Service Status: Started

NOTES

If installation process completes successfully then a window as shown below will appear.



If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.html
```

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then either you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the administration webapp is restricted to users with role "admin". The manager webapp is restricted to users with role "manager". Users are defined in `$CATALINA_HOME/conf/tomcat-users.xml`.

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation, and an introductory guide to developing web applications.

Tomcat mailing lists are available at the Tomcat project web site:

- users@tomcat.apache.org for general questions related to configuring and using Tomcat

Now , set the environment variable for tomcat :

Step 4:

Setting the JAVA_HOME Variable: Purpose of setting the environment variable JAVA_HOME is to specify the location of the java run time environment needed to support the Tomcat else Tomcat server does not run. This variable contains the path of JDK installation directory. Note that it should not contain the path up to bin.

set JAVA_HOME=C:\Program Files\Java\jdk1.5.0_08

Here, we have taken the URI path according to our installation convention

For Windows XP, Go through the following steps:

Start menu->Control Panel->System->Advanced tab->Environment Variables->New->set the Variable Name as JAVA_HOME and Variable Value as C:\Program Files\Java\jdk1.6.0 and then click on the entire three ok buttons one by one. It will set the JDK path.

For Windows 2000 and NT, follow these steps:

Start->Settings->Control Panel->System->Environment Variable->New->set the Variable Name as JAVA_HOME and Variable Value as C:\Program Files\Java\jdk1.6.0 and then click on all the three ok button one by one. It will set the JDK path

Now , Start the Tomcat Server : Start the tomcat server from the bin folder of Tomcat 6.0 directory by double clicking the " tomcat6.exe " file. You can also create a shortcut of this .exe file at your desktop.

Stop the Tomcat Server: Stop the server by pressing the "Ctrl + c" keys.

6.4 Introduction to Servlets

Servlet is a software component written in Java that dynamically extends the functionality of a server. The servlet's work is done behind the scenes on the server and only the results of the servlets processing are returned to the client usually in the form of HTML. Servlets provide a frame work for executing applications that implement the response – request paradigm.

Advantages

The following are the advantages of servlets;

- ✓ Capable of running in process
- ✓ Compiled (bytecode)
- ✓ Crash resistant
- ✓ Cross platform
- ✓ Cross server
- ✓ Durable
- ✓ Dynamically loaded across network
- ✓ Extensible (third party component)
- ✓ Multithreaded
- ✓ Protocol independent (HTTP, FTP, SMTP)
- ✓ Secure(memory access, security manager)

Characters of servlets

The following are the characteristics of servlets;

i) Dynamically built and returns HTML file based on the request.

ii) Processes user input passed by the html form and returns response.

iii) Facilitate communication between groups of people.

iv) Provide user authentication and other security mechanisms.

v) Interact with server resources such as databases, other applications and network files.

vi) Process inputs from many clients – eg: multiplayer games.
vii) Allow the server to communicate with the client applet.
viii) Automatically attach web page design elements.
ix) Forward request from one server to another.
x) Partition a single logical service across servers.
xi) Virtually any other way to enhance or extend a server.

6.5 Servlet Life Cycle

The life cycle of a servlet can be categorized into four parts:

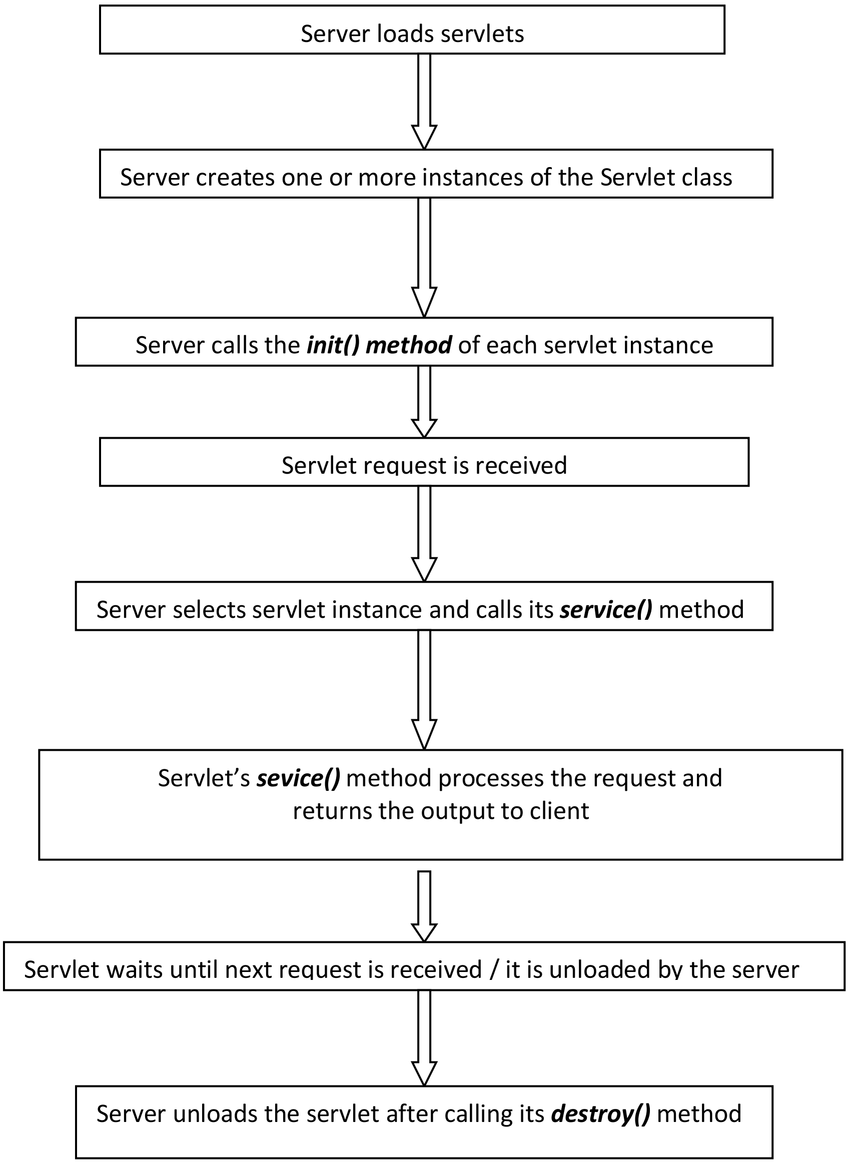
Loading and Instantiation:

The servlet container loads the servlet during startup or when the first request is made. The loading of the servlet depends on the attribute <load-on-startup> of web.xml file. If the attribute <load-on-startup> has a positive value then the servlet is load with loading of the container otherwise it load when the first request comes for service. After loading of the servlet, the container creates the instances of the servlet.

Initialization:

After creating the instances, the servlet container calls the init() method and passes the servlet initialization parameters to the init() method. The init() must be called by the servlet container before the servlet can service any request. The initialization parameters persist untill the servlet is destroyed. The init() method is called only once throughout the life cycle of the servlet.

The servlet will be available for service if it is loaded successfully otherwise the servlet container unloads the servlet.


Server loads servlets
Server creates one or more instances of the Servlet class
Server calls the ***init() method*** of each servlet instance
Servlet request is received
Server selects servlet instance and calls its ***service()*** method
Servlet's ***sevice()*** method processes the request and returns the output to client
Servlet waits until next request is received / it is unloaded by the server
Server unloads the servlet after calling its ***destroy()*** method

Servlet Life cycle

Servicing the Request:

After successfully completing the initialization process, the servlet will be available for service. Servlet creates seperate threads for each request. The sevlet container calls the service() method for servicing any request. The service() method determines the kind of request and calls the appropriate method (doGet() or doPost()) for handling the request and sends response to the client using the methods of the response object.

Destroying the Servlet:

If the servlet is no longer needed for servicing any request, the servlet container calls the destroy() method . Like the init() method this method is also called only once throughout the life cycle of the servlet. Calling the destroy() method indicates to the servlet container not to sent

NOTES

the any request for service and the servlet releases all the resources associated with it. Java Virtual Machine claims for the memory associated with the resources for garbage collection.

6.6 JSDK (Java Servlet Development Kit)

Servlets are server-side programs. Servlets are loaded and executed by a web server. The Java Servlet Development Kit (JSDK) is the reference implementation of the Java servlet API. It can be used for developing, running, testing, and debugging of Java Servlets.

6.7 Check Your Progress

1. Write True or False for the following :

(a) Tomcat is an open source web server
(b) The life cycle of a servlet can be categorized into five parts

6.8 Answers to Check Your Progress Questions

1. (a) True (b) False

6.9 Summary

Servlet is a software component written in Java that dynamically extends the functionality of a server. The life cycle of a servlet can be categorized into four parts. The Java Servlet Development Kit (JSDK) is the reference implementation of the Java servlet API.

6.10 Key Words

Web Server - A web server is also known as an HTTP Server. It responds to request from a web browser by returning Html images, applets or other data.

Servlet - It is a Java class in Java EE that conforms to the Java Servlet API, a protocol by which a Java class may respond to HTTP requests.

6.11 Self-Assessment Questions and Exercises

1) What is a Servlet?
2) When init() and destroy() method will be called?
3) What is life cycle of servlet?
4) How to make a context thread safe?

6.12 Further Readings

1. Jason Hunter Java Servlet Programming, second Edition , O'Reilley
2. Karl Moss, Java Servlets, Second Edition,McGraw Hill 1999.

UNIT 7
The Servlet API

Structure

7.0 Introduction

Servlets use classes and interfaces from two packages javax.servlet and javax.servlet.http. The top level package in javax instead of the familiar java, to indicate that ther servlet API is a standard extension.

7.1 Objectives

After going through the unit you will be able to;

- List the advantages of Servlet
- List the characteristics of Servlet
- Understand the life cycle of the servlet
- Work with Java servlet development kit

7.2 Servlet API (Application Program Interface)

Java Servlet API contains two core packages:

1. *javax.servlet*
2. *javax.servlet.http*

Servlets implement the javax.servlet.Servlet interface. The javax.servlet package contains the generic interfaces and classes that are implemented and extended by all servlets. While the javax.servlet.http package contains the classes that are used when developing HTTP -specific servlets.

The HttpServlet is extended from GenericServlet base class and it implements the Servlet interface. HttpServlet class provides a framework for handling the HTTP requests.

7.2.1 Java.servlet Package

Java.servlet Package consists of following interfaces and classes,

Interface

RequestDispather

Defines a request dispatcher object that receives request from the client and sends them to any resource (such as a servlet, CGI script, HTML file, or JSP file) available on the server.

ServletConfig

Defines an object that a servlet engine generates to pass configuration information to a servlet when such servlet is initialized.

ServletContext

A servlet engine generated object that gives servlets information about their environment.

ServletRequest

Defines a servlet engine generated object that enables a servlet to get information about a client request.

ServletResponse

Interface for sending MIME data from the servlet's service method to the client.

SingleThread

Model Defines a "single" thread model for servlet execution.

Class

The servlet API provides the following classes

GenericServlet class

The GenericServlet class implements the Servlet interface and, forconvenience, the ServletConfig interface. A Generic servlet contains the following five methods:

init()

The init() method is called only once by the servlet container throughout the life of a servlet. By this init() method the servlet get to know that it has been placed into service.

public void init(ServletConfig config) throws ServletException

The servlet cannot be put into the service if The init() method does not return within a fix time set by the web server. It throws a ServletException Parameters - The init() method takes a ServletConfig object that contains the initialization parameters and servlet's configuration and throws a ServletException if an exception has occurred.

service()

Once the servlet starts getting the requests, the service() method is called by the servlet container to respond. The servlet services the client's request with the help of two objects. These two objects javax.servlet.ServletRequest and javax.servlet.ServletResponse are passed by the servlet container. The status code of the response always should be set for a servlet that throws or sends an error.

public void service(ServletRequest req, ServletResponse res) throws ServletException, IOException

Parameters - The service() method takes the ServletRequest object that contains the client's request and the object ServletResponse contains the servlet's response. The service() method throws ServletException and IOExceptions exception.

getServletConfig()

public ServletConfig getServletConfig()

This method contains parameters for initialization and startup of the servlet and returns a ServletConfig object. This object is then passed to the init method. When this interface is implemented then it stores the ServletConfig object in order to return it. It is done by the generic class which implements this inetrface. It returns the ServletConfig object

getServletInfo()

public String getServletInfo()

The information about the servlet is returned by this method like version, author etc. This method returns a string which should be in the form of plain text and not any kind of markup. It returns a string that contains the information about the servlet

destroy()

This method is called when we need to close the servlet. That is before removing a servlet instance from service, the servlet container calls the destroy() method.

public void destroy()

Once the servlet container calls the destroy() method, no service methods will be then called . That is after the exit of all the threads running in the servlet, the destroy() method is called. Hence, the servlet gets a chance to clean up all the resources like memory, threads etc which are being held.

ServletInputStream class

An input stream for reading servlet requests, it provides an efficient readLine() method.

ServletOutputStream class

An output stream for writing servlet responses.

7.2.2 Reading using Parameters

In the servlet which will work as a controller here picks the value from the html page by using the method getParameter(). The output will be displayed to you by the object of the PrintWriter class.

This is a very simple example in which we are going to display the name on the browser which we have entered from the Html page.

To get the desired result firstly we have to make one html form which will have only one field named as name in which we will enter the name. And we will also have one submit button, on pressing the submit button the request will go to the server and the result will be displayed to us. The code of the program is given below:

Login.html

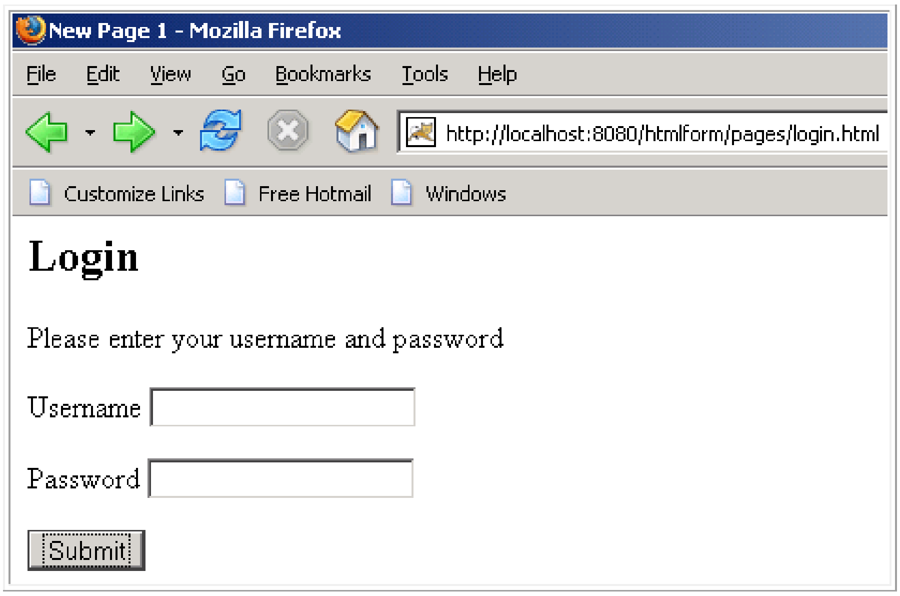
```
<html>
<head>
<title>New Page 1</title>
</head>
<body>
<h2>Login</h2>
<p>Please enter your username and password</p>
<form method="GET" action="/htmlform/LoginServlet">
  <p> Username  <input type="text" name="username" size="20"></p>
  <p> Password  <input type="text" name="password" size="20"></p>
  <p><input type="submit" value="Submit" name="B1"></p>
</form>
<p>&nbsp;</p>
</body>
</html>
```

LoginServlet.java

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class LoginServlet extends HttpServlet{
public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException {
    response.setContentType("text/html");
    PrintWriter out = response.getWriter();
    String name = request.getParameter("username");
    String pass = request.getParameter("password");
    out.println("<html>");
    out.println("<body>");
    out.println("Thanks Mr." + "  " + name + "  " + "for visiting Alagappa
University<br>" );
    out.println("Now you can see your password : " + "  " + pass + "<br>");
    out.println("</body></html>");
  }
}
```

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--<!DOCTYPE web-app
 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd"> -->

<web-app>
 <servlet>
  <servlet-name>Hello</servlet-name>
  <servlet-class>LoginServlet</servlet-class>
 </servlet>
 <servlet-mapping>
 <servlet-name>Hello</servlet-name>
 <url-pattern>/LoginServlet</url-pattern>
 </servlet-mapping>
</web-app>
```

The Output of the above program is:



New Page 1 - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/htmlform/pages/login.html
Customize Links Free Hotmail Windows

Login

Please enter your username and password

Username

Password

Submit

7.2.3 Reading Initialization Parameters for servlet

The servlet container supports the ability to store startup and configuration information for a servlet. After the container instantiates the servlet, it makes this information available to the servlet instance. This example demonstrates a servlet that retrieves some initialization parameters in its init() method:

```
// This method is called by the servlet container just before this servlet
// is put into service.
public void init() throws ServletException
 {
   getServletContext().log("getinit init");
   // Get the value of an initialization parameter
   String value = getServletConfig().getInitParameter("param1");
   // Get all available intialization parameters
   java.util.Enumeration enum =
getServletConfig().getInitParameterNames();
   for (; enum.hasMoreElements(); )
{
      // Get the name of the init parameter
      String name = (String)enum.nextElement();
      // Get the value of the init parameter
      value = getServletConfig().getInitParameter(name);
   }
   // The int parameters can also be retrieved using the servlet context
   value = getServletContext().getInitParameter("param1");
}
```

The initialization parameters for the servlet are specified in the deployment descriptor (***i.e., web.xml file***). Here is an example of a deployment descriptor that specifies two initialization parameters:

```
<web-app>
   <servlet>
      <servlet-name>Alagappa University</servlet-name>
      <servlet-class>com.mycompany.MyServlet</servlet-class>
      <init-param>
         <param-name> param1 </param-name>
         <param-value> value1 </param-value>
      </init-param>
      <init-param>
         <param-name> param2 </param-name>
         <param-value> value2 </param-value>
      </init-param>
      ...
   </servlet>
   ...
   ...

</web-app>
```

7.3 Check Your Progress

1. Write True or False for the following :

(a) Java Servlet API contains two core packages.
(b) The javax.servlet package contains the generic interfaces only.

7.4 Answers to Check Your Progress Questions

1. (a) True (b) False.

7.5 Summary

It can be used for developing, running, testing, and debugging of Java Servlets. Java.servlet Package consists of interfaces and classes.

7.6 Keywords

Session

A session is a persistent network connection between two host.(Client and Server) When the connection is closed the session gets over.

7.7 Self-Assessment Questions and Exercises

1) What is the difference between applet and servlet?
2) What is servlet API?
3) How to you Initialization Parameters for servlet?
4) Define javax.servlet.http.

7.8 Further Readings

1. James Goodwill,Developing Java Servlets,Second Edition
2. Callaway, Inside Servlets,Pearson Education India.

NOTES

UNIT 8

The javax.servlet Package

Structure

8.0 Introduction

The javax.servlet package contains classes to support generic,protocol independent servlets. These classes are extended by the classes in the javax.servlet.http package to add HTTP specific functionality. Most servlets implement it by extending one the two special classes : javax.servlet.GenericServlet or javax.servlet.HttpServlet.

8.1 Objectives

At the end of this unit you will be able to know;

- The javax.servlet HTTP package
- Handling Http Request & Responses
- How to use Cookies
- About Session Tracking
- Various Security Issues

8.2 javax.servlet.http Package

The javax.servlet.http package contains a number of classes and interfaces that describe and define the contracts between a servlet class running under the HTTP protocol and the runtime environment provided for an instance of such a class by a conforming servlet container.

The class HttpServlet implements the Servlet interface and provides a base developer will extend to implement servlets for implementing web applications employing the HTTP protocol. In addition to generic Servlet interface methods, the class HttpServlet implements interfaces providing HTTP functionality. The basic Servlet interface defines a service method for handling client requests. This method is called for each request that the servlet container routes to an instance of a servlet.

INTERFACES

HttpServletRequest - An HTTP servlet request.

HttpServletResponse - An HTTP servlet response.

HttpSession - The HttpSession interface is implemented by services to provide an association between an HTTP client and HTTP server.

HttpSessionBindingListener - Objects implement this interface so that they can be notified when they are being bound or unbound from a HttpSession.

HttpSessionContext- Deprecated.The HttpSessionContext class has been deprecated for security reasons.

CLASSES

Cookie This class represents a "Cookie", as used for session management with HTTP and HTTPS protocols.

HttpServlet An abstract class that simplifies writing HTTP servlets.

HttpSessionBindingEvent This event is communicated to a HttpSessionBindingListener whenever the listener is bound to or unbound from a HttpSession value.

HttpUtils A collection of static utility methods useful to HTTP servlets.

NOTES

8.3 Handling Http Request & Responses

Http request and response

Whenever the user sends the request to the server then server generates two obects, first is HttpServletRequest object and the second one is HttpServletResponse object. HttpServletRequest object represents the client's request and the HttpServletResponse represents the servlet's response.

Inside the doGet(() method our servlet has first used the setContentType() method of the response object which sets the content type of the response to text/html. It is the standard MIME content type for the Html pages. After that it has used the method getWriter() of the response object to retrieve a PrintWriter object. To display the output on the browser we use the println() method of the PrintWriter class.

The code the program is given below:

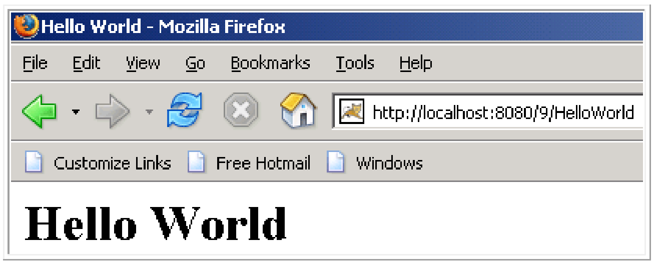
```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class HelloWorld extends HttpServlet{
 public void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException,IOException
 {
   response.setContentType("text/html");
   PrintWriter pw = response.getWriter();
   pw.println("<html>");
   pw.println("<head><title>Hello World</title></title>");
   pw.println("<body>");
   pw.println("<h1>Hello World</h1>");
   pw.println("</body></html>");
  }
}
```

web.xml file for this program:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!--<!DOCTYPE web-app
 PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"
 "http://java.sun.com/dtd/web-app_2_3.dtd"> -->
```

```
<web-app>
 <servlet>
  <servlet-name>Hello</servlet-name>
  <servlet-class>HelloWorld</servlet-class>
 </servlet>
 <servlet-mapping>
 <servlet-name>Hello</servlet-name>
 <url-pattern>/HelloWorld</url-pattern>
 </servlet-mapping>
</web-app>
```

The output of the program is given below:


Hello World - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/9/HelloWorld
Customize Links
Free Hotmail
Windows

Hello World

8.4 Using Cookies

A cookie is a http mechanism that is used for maintaining specific user settings and managing state. A cookie is a simple mechanism which is used to store and retrieve user specified info on the web.

When a httpserver receives an request the server may choose to return some state information that is stored by a cookie enabled client. This state information includes URL range whenever an Client receives http request it checks the URL of the request against the URL range of all stored cookies.

HTTP Header Syntax for Cookies:

```
HTTP//1.0 200 ok
Server netscape_enterprise/7.0
Content type: Text.html
Content length: 87
Set_cookies: customer id=1234;Domain =acmesite.com
```

Syntax:

```
Set_cookies:<name>=<value>;expression=<date>
Domain=<Domain_name>;path=<path>;secure
```

NOTES

For Example,

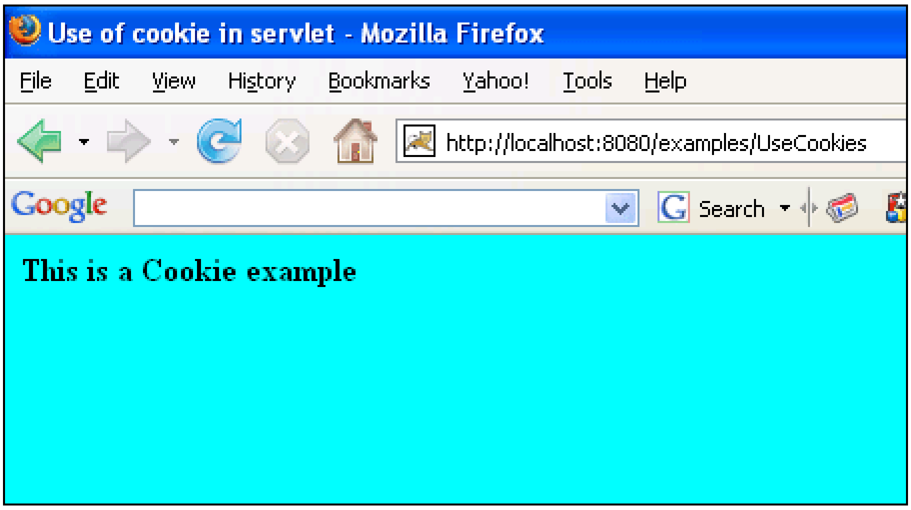
A servlet uses the getCookies() method of HTTPServletRequest to retrieve cookies as request. The addCookie() method of HTTPServletResponse sends a new cookie to the browser. You can set the age of cookie by setMaxAge() method. Here is the code which defines cookie and shows how to set the maximum age of cookie.

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

public class UseCookies extends HttpServlet
 {
   public void doGet ( HttpServletRequest request,
   HttpServletResponse response )throws ServletException, IOException
{
  PrintWriter out;
  response.setContentType("text/html");
  out = response.getWriter();
  Cookie cookie = new Cookie("CName","Cookie Value");
  cookie.setMaxAge(100);
  response.addCookie(cookie);
  out.println("<HTML><HEAD><TITLE>");
  out.println(" Use of cookie in servlet");
  out.println("</TITLE></HEAD><BODY BGCOLOR='cyan'>");
  out.println(" <b>This is a Cookie example</b>");
  out.println("</BODY></HTML>");
  out.close();
   }
}
```

In the above example, a servlet class UseCookies defines the cookie class. Here the age of cookie has been set as setMaxAge(100). If its value is set to 0, the cookie will delete immediately. After the time provided been expired, cookie will automatically deleted.

Output :

NOTES


Use of cookie in servlet - Mozilla Firefox
File Edit View History Bookmarks Yahoo! Tools Help
http://localhost:8080/examples/UseCookies
Google
Search

This is a Cookie example

8.5 Session Tracking

A session is a persistent network connection between two host (Client and Server). When the connection is closed the session gets over. As we know that the Http is a stateless protocol, means that it can't persist the information. It always treats each request as a new request. In Http client makes a connection to the server, sends the request., gets the response, and closes the connection.

In session management client first make a request for any servlet or any page, the container receives the request and generate a unique session ID and gives it back to the client along with the response. This ID gets stores on the client machine. Thereafter when the client request again sends a request to the server then it also sends the session Id with the request. There the container sees the Id and sends back the request. Session Tracking can be done in three ways:

Hidden Form Fields:

This is one of the way to support the session tracking. As we know by the name, that in this fields are added to an HTML form which are not displayed in the client's request. The hidden form field are sent back to the server when the form is submitted. In hidden form fields the html entry will be like this : <input type ="hidden" name = "name" value="">. This means that when you submit the form, the specified name and value will be get included in get or post method. In this session ID information would be embedded within the form as a hidden field and submitted with the Http POST command.

For example,

NOTES

```
<html>
<head>
<title> What is your name</title></head>
<body><form action = “Sample servlets” method = post>
<p>“Please enter your first name” <BR>
<Input type = “Text” name = “name” size = 25><br>
<input type = “Hidden” name = “bcolor” value = “blue”><br>
<input type = “submit” value = “submit”>
</form>
</body>
</html>
```

URL Rewriting:

This is another way to support the session tracking. URL Rewriting can be used in place where we don't want to use cookies. It is used to maintain the session. Whenever the browser sends a request then it is always interpreted as a new request because http protocol is a stateless protocol as it is not persistent. Whenever we want that out request object to stay alive till we decide to end the request object then, there we use the concept of session tracking. In session tracking firstly a session object is created when the first request goes to the server. Then server creates a token which will be used to maintain the session. The token is transmitted to the client by the response object and gets stored on the client machine. By default the server creates a cookie and the cookie get stored on the client machine. For example,

```
<html>
<head>
<title> menu</title></head>
<body>
<p> please select choices below<p>
<a href = “/catalog.html?sessionID = 122893”>Catalog</A><BR>
<A href = “/info.html?sessionID = 122893”>Corporate
Info</A></BR>
<A href = “/press.html?sessionID = 122893”>Press
release</A></BR>
<A href = “/employ.html?sessionID =
122893”>Employment</A></BR>
<A href = “/info.html?sessionID = 122893”>Corporate
Info</A></BR>
<A href = “/search.html?sessionID = 122893”>Search the
Site</A></BR>
</body>
</html>
```

Disadvantage:

If a client submits a html form rather than clicking on the hyperlink the state information within rewritten URL is not passed to the Server.

Cookies:

When cookie based session management is used, a token is generated which contains user's information, is sent to the browser by the server. The cookie is sent back to the server when the user sends a new request. By this cookie, the server is able to identify the user. In this way the session is maintained. Cookie is nothing but a name- value pair, which is stored on the client machine. By default the cookie is implemented in most of the browsers. If we want then we can also disable the cookie. For security reasons, cookie based session management uses two types of cookies.

In this example we are going to make a program in which we will find the session id which was generated by the container.

HttpSession session = request.getSession(); Inside the service method we ask for the session and everything gets automatically, like the creation of the HttpSession object. There is no need to generate the unique session id. There is no need to make a new Cookie object. Everything happens automatically behind the scenes.

As soon as call the method getSession() of the request object a new object of the session gets created by the container and a unique session id generated to maintain the session. This session id is transmitted back to the response object so that whenever the client makes any request then it should also attach the session id with the request object so that the container can identify the session.

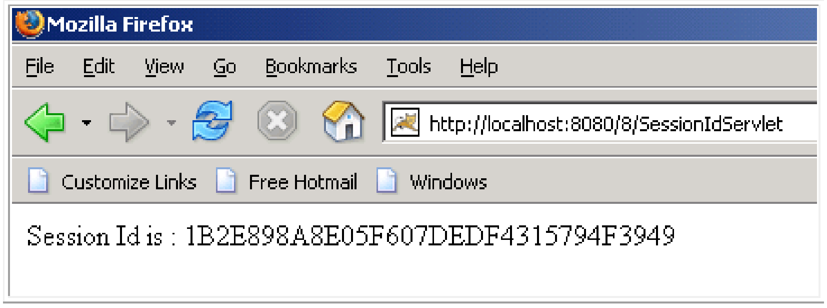
The code of the program is given below:

```
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;
public class SessionIdServlet extends HttpServlet{
  protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException{
    response.setContentType("text/html");
    PrintWriter pw = response.getWriter();
    HttpSession session = request.getSession();
    String id = session.getId();
  pw.println("Session Id is : " + id);
 }}
```

web.xml file for this program:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app  PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application 2.3//EN"  "http://java.sun.com/dtd/web-app_2_3.dtd">
<web-app> <servlet>
 <servlet-name>Zulfiqar</servlet-name>
 <servlet-class>SessionIdServlet</servlet-class>
 </servlet>
 <servlet-mapping>
 <servlet-name>Zulfiqar</servlet-name>
 <url-pattern>/SessionIdServlet</url-pattern>
 </servlet-mapping></web-app>
```

The output of the program is given below**:**


Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost:8080/8/SessionIdServlet
Customize Links
Free Hotmail
Windows

```
Session Id is : 1B2E898A8E05F607DEDF4315794F3949
```

8.6 Security Issues

As with Java applets, Java servlets have security issues to worry about, too.

The Servlet Sandbox

A servlet can originate from several sources. A webmaster may have written it; a user may have written it; it may have been bought as part of a third-party package or downloaded from another web site. Based on the source of the servlet, a certain level of trust should be associated with that servlet. Some web servers provide a means to associate different levels of trust with different servlets. This concept is similar to how web browsers control applets, and is known as "sandboxing".

A servlet sandbox is an area where servlets are given restricted authority on the server. They may not have access to the file system or network, or they may have been granted a more trusted status. It is up to the web server administrator to decide which servlets are granted this status. Note that a fully trusted servlet has full access to the server's file system and networking capabilities. It could even perform a System.exit(), stopping the web server...

Access Control Lists (ACLs)

Many web servers allow you to restrict access to certain web pages and servlets via access control lists (ACLs). An ACL is a list of users who are allowed to perform a specific function in the server. The list specifies:

- What kind of access is allowed
- What object the access applies to
- Which users are granted access

Each web server has its own means of specifying an ACL, but in general, a list of users is registered on the server, and those user names are used in an ACL. Some servers also allow you to add users to logical groups, so you can grant access to a group of users without specifying all of them explicitly in the ACL.

ACLs are extremely important, as some servlets can present or modify sensitive data and should be tightly controlled, while others only present public knowledge and do not need to be controlled.

Threading Issues

A web server can call a servlet's service() method for several requests at once. This brings up the issue of thread safety in servlets.

But first consider what you do not need to worry about: a servlet's init() method. The init() method will only be called once for the duration of the time that a servlet is loaded. The web server calls init() when loading, and will not call it again unless the servlet has been unloaded and reloaded. In addition, the service() method or destroy() method will not be called until the init() method has completed its processing.

Things get more interesting when you consider the service() method. The service() method can be called by the web server for multiple clients at the same time. (With the JSDK 2.0, you can tag a servlet with the SingleThreadModel interface. This results in each call to service() being handled serially. Shared resources, such as files and databases, can still have concurrency issues to handle.)

If your service() method uses outside resources, such as instance data from the servlet object, files, or databases, you need to carefully examine what might happen if multiple calls are made to service() at the same time. For example, suppose you had defined a counter in your servlet class that keeps track of how many service() method invocations are currently running:

```
 private int counter = 0;
```

Next, suppose that your service() method contained the following code:

```
int myNumber = counter + 1;  // line 1
counter = myNumber;          // line 2

 // rest of the code in the service() method

 counter = counter - 1;
```

What would happen if two service() methods were running at the same time, and both executed line 1 before either executed line 2? Both would have the same value for myNumber, and the counter would not be properly updated.

For this situation, the answer might be to synchronize the access to the counter variable:

```
synchronized(this) {
   myNumber = counter + 1;
   counter = myNumber;
 }
 // rest of code in the service() method
 synchronized(this) {
   counter = counter - 1 ;
 }
```

This ensures that the counter access code is executed only one thread at a time.

8.7 Check Your Progress

1. Write True or False for the following :
   (a) HttpUtils is an collection of static utility methods.
   (b) A session is a persistent network connection.
   (c) An ACL is a list of Browers.

8.8 Answers to Check Your Progress Questions

1. (a) True (b) True (c) False.

8.9 Summary

In this you learnt about cookie, session tracking and security issues of javax servlet.

8.10 Glossary

Cookie - It is a http mechanism that is used for maintaining specific user settings and managing state.

8.11 Self-Assessment Questions and Exercises

1) How can you set and delete cookie?
2) What is session tracking?
3) What is the difference between session and Cookie?

8.12 Further Readings

1. James Goodwill, Developing Java Servlets, Second Edition
2. Callaway, Inside Servlets, Pearson Education India.

NOTES

BLOCK IV

JAVA SERVER PAGES (JSP)

UNIT 9

Introduction to Jave Server Pages (JSP)

Structure

9.0 Introduction

Java Server Pages or JSP for short is Sun's solution for developing dynamic web sites. JSP provide excellent server side scripting support for creating database driven web applications. JSP enable the developers to directly insert java code into jsp file, this makes the development process very simple and its maintenance also becomes very easy.

JSP pages are efficient, it loads into the web server’s memory on receiving the request very first time and the subsequent calls are served within a very short period of time.

9.1 Objectives

After going through the unit you will be able to;

- Understand the problems with servlet
- Know the anatomy of servlet page
- Explain about JSP Processing
- Know JSP Application Design with MVC
- Understand how to Setting Up JSP Environment
- Installing the Java Software Development Kit
- Know about Tomcat Server & Testing Tomcat

9.2 Problem with Servlet

Both Servlet & JSP follows MVC (Model View Controller) pattern using Model 2 architecture ie servlet act the controller, JSP act as the view & java technology classes act as the model.

Servlet often contain both business logic & presentation logic. The main advantage of JSP over servlets is that JSP separates the presentation code (HTML code) from the Java Code. servlet-based approach still has a few problems:

- Thorough Java programming knowledge is needed to develop and maintain all aspects of the application, since the processing code and the HTML elements are lumped together.

- Changing the look and feel of the application, or adding support for a new type of client (such as a WML client), requires the servlet code to be updated and recompiled.

- It's hard to take advantage of web-page development tools when designing the application interface. If such tools are used to develop the web page layout, the generated HTML must then be manually embedded into the servlet code, a process which is time consuming, error prone, and extremely boring.

- It is tedious to use the out.println() statements in the servlet for each line of generated HTML code.

- Also it is difficult to change/debug a servlet as the HTML code is integrated with the java code.

NOTES

By implementing JSP you can make the HTML code generation a separate process and the Java Code generation a separate process. Instead of embedding HTML in the code, you place all static HTML in a JSP page, just as in a regular web page, and add a few JSP elements to generate the dynamic parts of the page. The request processing can remain the domain of the servlet, and the business logic can be handled by JavaBeans and EJB components.

9.3 The Anatomy of a JSP Page

A JSP page is a mixture of standard HTML tags, web page content, and some dynamic content that is specified using JSP constructs. Everything except the JSP constructs is called Template Text.

Everything in the page that isn't a JSP element is called template text. Template text can be any text: HTML, WML, XML, or even plain text. Since HTML is by far the most common web-page language in use today, most of the descriptions and examples in this book use HTML, but keep in mind that JSP has no dependency on HTML; it can be used with any markup language. Template text is always passed straight through to the browser. When a JSP page request is processed, the template text and dynamic content generated by the JSP elements are merged, and the result is sent as the response to the browser.

9.4 JSP Processing

The following steps explain how the web server creates the web page using JSP:

- As with a normal page, your browser sends an HTTP request to the web server.

- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with .jsp instead of .html.

- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code that implements the corresponding dynamic behavior of the page.

- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.

- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format, which the servlet engine passes to the web server inside an HTTP response.

- The web server forwards the HTTP response to your browser in terms of static HTML content.

- Finally web browser handles the dynamically generated HTML page inside the HTTP response exactly as if it were a static page

Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with other scripting languages (such as PHP) and therefore faster.

So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet

The key to understanding the low-level functionality of JSP is to understand the simple life cycle they follow.

A JSP life cycle can be defined as the entire process from its creation till the destruction which is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

The following are the paths followed by a JSP

Compilation

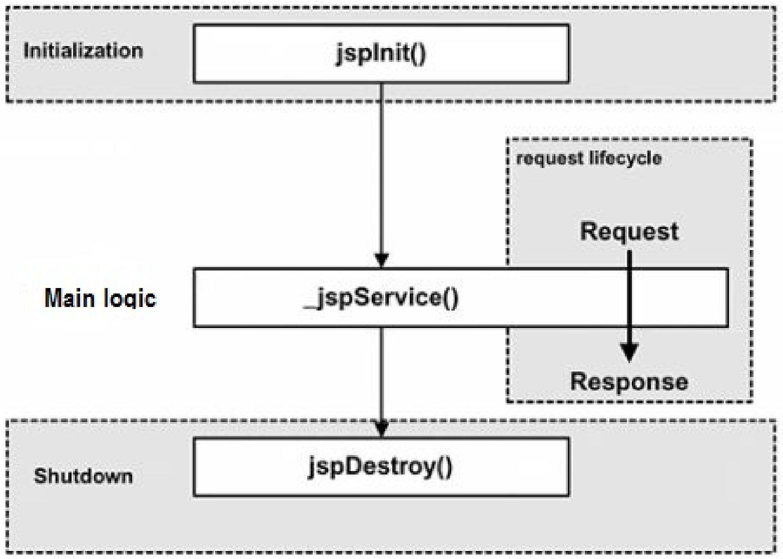
Initialization

Execution

Cleanup

NOTES

The three major phases of JSP life cycle are very similar to Servlet Life Cycle and they are as follows:



(1) JSP Compilation:

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps:

Parsing the JSP.

Turning the JSP into a servlet.

Compiling the servlet.

(2) JSP Initialization:

When a container loads a JSP it invokes the jspInit() method before servicing any requests. If you need to perform JSP-specific initialization, override the jspInit() method:

```
public void jspInit()
{
  // Initialization code...
}
```

Typically initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

(3) JSP Execution:

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the _jspService() method in the JSP.

The _jspService() method takes an HttpServletRequest and an HttpServletResponse as its parameters as follows:

```
void _jspService(HttpServletRequest request,
              HttpServletResponse response)
{
  // Service handling code...
}
```

The _jspService() method of a JSP is invoked once per a request and is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods ie. GET, POST, DELETE etc.

(4) JSP Cleanup:

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The jspDestroy() method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

The jspDestroy() method has the following form:

```
        public void jspDestroy()
        {
          // Your cleanup code goes here.
        }
```

9.5 JSP Application Design with MVC

The main aim of the MVC architecture is to separate the business logic and application data from the presentation data to the user.

Here are the reasons why we should use the MVC design pattern.

1. They are resuable : When the problems recurs, there is no need to invent a new solution, we just have to follow the pattern and adapt it as necessary.
2. They are expressive: By using the MVC design pattern our application becomes more expressive.

1). Model:

The model object knows about all the data that need to be displayed. It is model who is aware about all the operations that can be applied to transform that object. It only represents the data of an application. The model represents enterprise data and the business rules that govern access to and updates of this data. Model is not aware about the presentation data and how that data will be displayed to the browser.

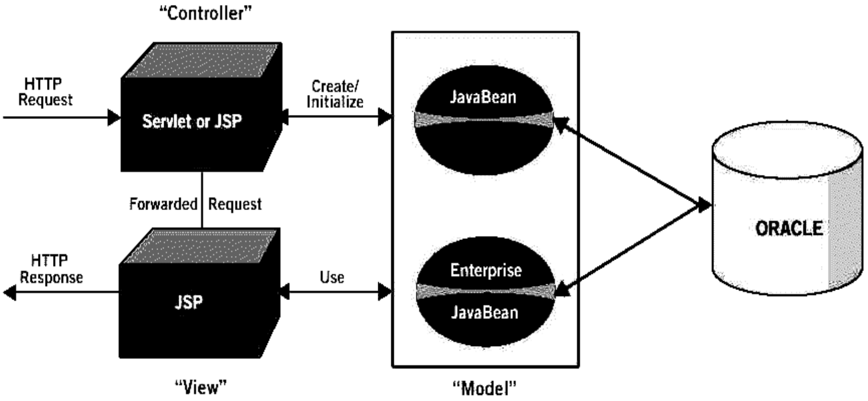
2). View :

The view represents the presentation of the application. The view object refers to the model. It uses the query methods of the model to obtain the contents and renders it. The view is not dependent on the application logic. It remains same if there is any modification in the business logic. In other words, we can say that it is the responsibility of the of the view's to maintain the consistency in its presentation when the model changes.

3). Controller:

Whenever the user sends a request for something then it always go through the controller. The controller is responsible for intercepting the requests from view and passes it to the model for the appropriate action. After the action has been taken on the data, the controller is responsible for directing the appropriate view to the user. In GUIs, the views and the controllers often work very closely together.

NOTES



9.6 Setting Up and JSP Environment

A development environment is where you would develop your JSP programs, test them and finally run them. This chapter will guide you to setup your JSP development environment which involves following steps:

9.6.1 Installing Java Software Development Kit

This step involves downloading an implementation of the Java Software Development Kit (SDK) and setting up PATH environment variable appropriately.

You can downloaded SDK from Oracle's Java site: Java SE Downloads.

Once you download your Java implementation, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac, typically java_install_dir/bin and java_install_dir respectively.

If you are running Windows and installed the SDK in C:\jdk1.5.0_20, you would put the following line in your C:\autoexec.bat file.

```
set PATH=C:\jdk1.5.0_20\bin;%PATH%
set JAVA_HOME=C:\jdk1.5.0_20
```

Alternatively, on Windows NT/2000/XP, you could also right-click on My Computer, select Properties, then Advanced, then Environment Variables. Then, you would update the PATH value and press the OK button.

NOTES

On Unix (Solaris, Linux, etc.), if the SDK is installed in /usr/local/jdk1.5.0_20 and you use the C shell, you would put the following into your .cshrc file.

```
setenv PATH /usr/local/jdk1.5.0_20/bin:$PATH
setenv JAVA_HOME /usr/local/jdk1.5.0_20
```

Alternatively, if you use an Integrated Development Environment (IDE) like Borland JBuilder, Eclipse, IntelliJ IDEA, or Sun ONE Studio, compile and run a simple program to confirm that the IDE knows where you installed Java.

9.6.2 Tomcat Server and testing Tomcat

Apache Tomcat is an open source software implementation of the JavaServer Pages and Servlet technologies and can act as a standalone server for testing JSP and Servlets and can be integrated with the Apache Web Server. Here are the steps to setup Tomcat on your machine:

Download latest version of Tomcat from http://tomcat.apache.org/.

Once you downloaded the installation, unpack the binary distribution into a convenient location. For example in C:\apache-tomcat-5.5.29 on windows, or /usr/local/apache-tomcat-5.5.29 on Linux/Unix and create CATALINA_HOME environment variable pointing to these locations.

Tomcat can be started by executing the following commands on windows machine:

```
%CATALINA_HOME%\bin\startup.bat
```

or

```
C:\apache-tomcat-5.5.29\bin\startup.bat
```

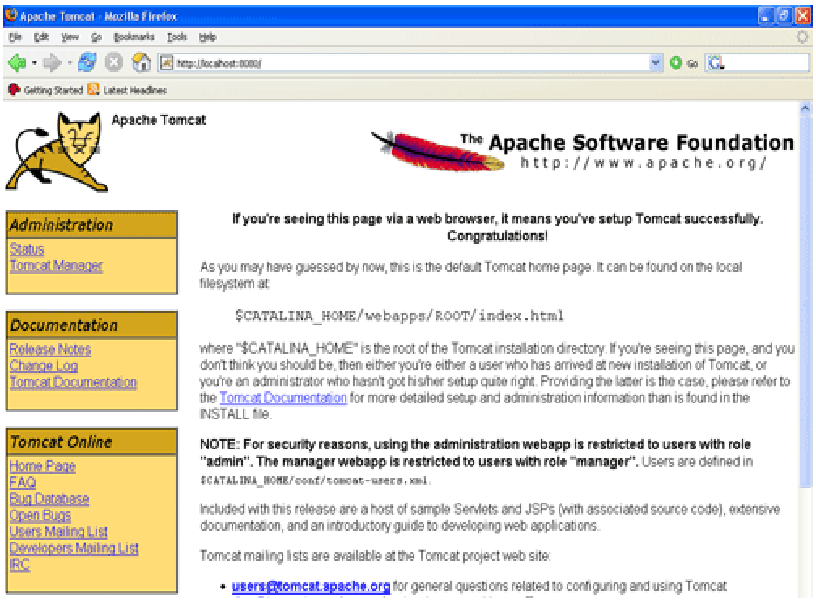
Tomcat can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine:

```
$CATALINA_HOME/bin/startup.sh
```

or

/usr/local/apache-tomcat-5.5.29/bin/startup.sh

After a successful startup, the default web applications included with Tomcat will be available by visiting http://localhost:8080/. If everything is fine then it should display following result:



If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

```
$CATALINA_HOME/webapps/ROOT/index.html
```

where "$CATALINA_HOME" is the root of the Tomcat installation directory. If you're seeing this page, and you don't think you should be, then either you're either a user who has arrived at new installation of Tomcat, or you're an administrator who hasn't got his/her setup quite right. Providing the latter is the case, please refer to the Tomcat Documentation for more detailed setup and administration information than is found in the INSTALL file.

NOTE: For security reasons, using the administration webapp is restricted to users with role "admin". The manager webapp is restricted to users with role "manager". Users are defined in `$CATALINA_HOME/conf/tomcat-users.xml`.

Included with this release are a host of sample Servlets and JSPs (with associated source code), extensive documentation, and an introductory guide to developing web applications.

Tomcat mailing lists are available at the Tomcat project web site:

- users@tomcat.apache.org for general questions related to configuring and using Tomcat

Further information about configuring and running Tomcat can be found in the documentation included here, as well as on the Tomcat web site: http://tomcat.apache.org

Tomcat can be stopped by executing the following commands on windows machine:

%CATALINA_HOME%\bin\shutdown
or
C:\apache-tomcat-5.5.29\bin\shutdown

Tomcat can be stopped by executing the following commands on Unix (Solaris, Linux, etc.) machine:

$CATALINA_HOME/bin/shutdown.sh
or
/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh

Setting up CLASSPATH

Since servlets are not part of the Java Platform, Standard Edition, you must identify the servlet classes to the compiler.

If you are running Windows, you need to put the following lines in your C:\autoexec.bat file.

set CATALINA=C:\apache-tomcat-5.5.29
set CLASSPATH=%CATALINA%\common\lib\jsp-api.jar;%CLASSPATH%

Alternatively, on Windows NT/2000/XP, you could also right-click on My Computer, select Properties, then Advanced, then Environment Variables. Then, you would update the CLASSPATH value and press the OK button.

On Unix (Solaris, Linux, etc.), if you are using the C shell, you would put the following lines into your .cshrc file.

setenv CATALINA=/usr/local/apache-tomcat-5.5.29
setenv CLASSPATH $CATALINA/common/lib/jsp-api.jar:$CLASSPATH

NOTE:

Assuming that your development directory is C:\JSPDev (Windows) or /usr/JSPDev (Unix) then you would need to add these directories as well in CLASSPATH in similar way as you have added above.

9.7 Check Your Progress

1. Write True or False for the following :
   (a) JSP page which ends with .html.
   (b) SDK downloaded from Oracle's Java site.
   (c) The compilation process of JSP involves three steps.

9.8 Answers to Check Your Progress Questions

1. (a) False (b) True (c) True.

9.9 Summary

In this unit your have learnt about the problem with servlet, anatomy of a JSP page, JSP Processing and JSP Application Design with MVC. The main aim of the MVC architecture is to separate the business logic and application data from the presentation data to the user. We have also learnt how to set up Tomcat and JSP Environment in easy steps. We have also learnt JSP Application Development like, Generating Dynamic Content and Scripting Elements.

9.10 Key Words

Expression

An expression tag contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

Cookie

A cookie is a name/value pair that the server passes to the browser in a response header.

9.11 Self-Assessment Questions and Exercises

Q: What is an Expression?

A: An expression tag contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file. Because the value of an expression is converted to a String, you can use an expression within text in a JSP file. Like

```
<%= someexpression %>
<%= (new java.util.Date()).toLocaleString() %>
```

You cannot use a semicolon to end an expression

Q: What is a Declaration?

A: A declaration declares one or more variables or methods for use later in the JSP source file. A declaration must contain at least one complete declarative statement. You can declare any number of variables or methods within one declaration tag, as long as they are separated by semicolons. The declaration must be valid in the scripting language used in the JSP file.

<%! somedeclarations %>
<%! int i = 0; %>
<%! int a, b, c; %>

9.12 Further Readings

- Hans Bergsten, Second Edition,Java Server Pages, O’Reilly Media, August 2002.
- Phil Hanna, The Complete reference JSP 2.0, Tata McGraw Hill,Delhi

UNIT 10 JSP Application Development

Structure

10. 0 Introduction

If you have had the opportunity to build web applications using technologies such as Common Gateway Interface (CGI) and servlets, you are accustomed to the idea of writing a program to generate the whole page (the static and the dynamic part) using that same program. If you are looking for a solution in which you can separate the two parts, look no further. JavaServer Pages (JSP) is here.

10. 1 Objectives

After going through the unit, you will be able to;

- Write a JSP Application Development Programs
- Generating Dynamic Content
- Using Scripting Elements
- List and understand the types of Implicit JSP Objects
- Know about Conditional Processing
- Using an Expression to Set an Attribute
- Declaring Variables and Methods

NOTES

10.2 JSP Application Development

JSP pages allow you to separate front-end presentation from business logic (middle and back-end tiers). It is a great Rapid Application Development (RAD) approach to Web applications.

10.2.1 Generating Dynamic Content

JSP pages are a third generation solution that can be combined easily with some second generation solutions, creating dynamic content, and making it easier and faster to build web-based applications that work with a variety of other technologies: web servers, web browsers, application servers and other development tools.

Creating JSP Page

When working with JSP pages, you just need a regular text editor such as Notepad on Windows or Emacs on Unix. There are a number of tools that may make it easier for you, such as syntax-aware editors that color-code JSP and HTML elements. Some Interactive Development Environments (IDE) even includes a small web container that allows you to easily execute and debug the pages during development.

The primary method for generating dynamic content on a JSP page is through the use of JSP expressions. A JSP expression is essentially a Java expression that is automatically converted to a String and written to the output stream. Here are some examples of JSP expressions:

The value of Pi is: *<%= Math.PI %>*

A random number between 1 and 100 is:

*<%= (int)(Math.random()*100)+1 %>*

The value of the "p" request paramter is:

<%= request.getParameter("p") %>

Today is: <%= new java.text.SimpleDateFormat("EEEE").format(new java.util.Date()) %>

Dynamic content can also be generated from a scriptlet by using out to write to the output stream. Here are some examples:

```
 <%
   if (Math.random() > .5)
{
     out.println("<b>You win!</b>");
   } else
{
     out.println("Sorry, you lose. Try again.");
   }
%>
```

10.2.2 Using Scripting Elements

JSP Scripting allows you to insert Java code into the servlet which is generated from JSP page. These are the forms of scripting elements such as: comment, expression, scriptlet, and declaration and expression language.

JSP Comment

JSP comments are used to explain the complicated logic code or to mark some region inside a JSP page for later changes. Comments in JSP are declared inside a JSP page as follows:

<%-- This is a JSP comment -- %> <%-- This is a JSP comment can span in multiple lines --%>JSP comment is stripped off from the page in the response to the web browser.

Expression

The most simple and basic of JSP scripting is expression. Expression is used to insert values directly to the output. The syntax of the expression is as follows.

(Be noted that there is no space between <% and =)

<?= expression ?>

For example, if you want to print out the current date and time you can use the expression as follows:

<%= new java.util.Date()%>

The XML syntax of the JSP expression is as follows:

NOTES

<jsp:expression>   Java Expression </jsp:expression>

Scriptlet

Scriptlet is similar to the Expression without the equal sign "=". You can insert any plain Java code inside the scriptlet. Because of mixing between Java code and HTML is difficult to maintain so scriptlet is not recommended to use anymore. Here is the syntax of the scriptlet:

```
<% // any java source code here %>
```

In this example, we print the greeting based on the current time of the day using scriptlet.

```
<%@page contentType="text/html" pageEncoding="UTF-8"%>

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01
Transitional//EN"     "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<title>JSP syntax</title>
 </head>
<body>

<%
// using scriptlet
java.util.Calendar now = new java.util.GregorianCalendar();
String tod = "";
if (now.get(now.HOUR_OF_DAY) < 12)
        {               tod = "Morning!";         }
        else if
         (now.get(now.HOUR_OF_DAY) < 18)
        {               tod = "Afternoon!";              }
        else
        {               tod = "Evening!";              }
%>
Good <%=tod%>
</body>
</html>
```

The XML syntax of JSP scriptlet is as follows:

```
<jsp:scriptlet>    // Java code of scriptlet </jsp:scriptlet>
```

Using Arrays

One can easily use scriptlets to loop over arrays. In this example, the user is presented with choice boxes. When s/he presses the submit button, the choices are displayed.

```
 <HTML>
  <BODY BGCOLOR="WHITE">
   <FORM ACTION="choices.jsp">
     <INPUT    type="checkbox"    name="music"    value="Classical">
Classical<BR>
     <INPUT type="checkbox" name="music" value="Rock"> Rock<BR>
     <INPUT type="checkbox" name="music" value="Jazz"> Jazz<BR>
     <INPUT    type="checkbox"    name="music"    value="Blues">
Blues<BR>
     <INPUT  type="checkbox"  name="music"  value="DC-GoGo">  DC
GoGo<BR>
     <INPUT type="submit" value="Submit">
   </FORM>
   <%
      String[] selected = request.getParameterValues("music");

      if (selected != null && selected.length != 0) {
   %>
         You like the following kinds of music:
         <UL>
          <%
            for (int i = 0; i < selected.length; i++) {
              out.println("<LI>" + selected[i]);
            }
          %>
         <UL>
    <% } %>
  </BODY>
 </HTML>
```

Declaration

If you want to define methods or fields you can use JSP declaration. The JSP declaration is surrounded by the sign <%! and %>. For example, if you want to declare a variable x, you can use JSP declaration as follows:

*<%! int x = 10; %>*The final semicolon is very important.

The difference between a variable using declaration and a variable is declared using Scriptlet is that a variable declare using declaration tag is accessible by all the methods while a variable declared using Scriptlet is only accessible to the method _jspservice of the generated servlet from JSP page.

We can also declare a method using declaration tag, such as:

```
 <%!      public boolean isInRange(int x,int min,int max)
        {
        return x >= min && x <= max;
         }
%>
```

10.2.3 Implicit JSP Objects

Implicit objects in jsp are the objects that are created by the container automatically and the container makes them available to the developers, the developers do not need to create them explicitly. Since these objects are created automatically by the container and are accessed using standard variables; hence, they are called implicit objects. The implicit objects are parsed by the container and inserted into the generated servlet code. They are available only within the jspService method and not in any declaration. Implicit objects are used for different purposes. Our own methods (user defined methods) can't access them as they are local to the service method and are created at the conversion time of a jsp into a servlet. But we can pass them to our own method if we wish to use them locally in those functions.

JSP container provides a list of instantiated objects for you to access different kind of data in a web application. Those objects are called implicit object because it is automatically available to access. These are some main implicit objects in JSP which you use most:

- request object
- response object
- session object
- out object
- pageContext object
- application object
- config object
- page object
- excpetion object

The request object

Each time a client requests a JSP page, the JSP engine creates a new object to represents that request called request object. The request object is an instance of class javax.servlet.http.HttpServletRequest. The request object contains all information about the current HTTP request and the clients. Be noted that request object only available in a scope of the current request. It is re-created each time new request is made. By using methods of the request object, you can access almost information such as HTTP header, query string, cookies...

The response object

JSP also creates the response object which is an instance of class javax.servlet.http.HttpServletResponse. The response object represents the response to the client. By using this object, you can add new cookies, change MIME content type of the page. In addition, the response object also contains sufficient information on the HTTP to be able to return HTTP status codes or make the page redirect to the other page.

The session object

The session object is used to track information of a particular client between multiple requests. the session object is available in the server so it can helps you to overcome the stateless of HTTP protocol. You can use session object to store a arbitrary information between client requests. The session object is an instance of class javax.servlet.http.HttpSession.

The out object

The output stream is exposed to the JSP through the out object. the out object is an instance of class javax.servlet.jsp.JspWriter. The out object may refer to an output stream or a filtered stream... You can use the out object methods to send the data into the output stream such as println method. Then JSP take care the rest.

The pageContext object

The pageContext object represent the entire JSP page. You can use the pageContext object to get page attributes of a page. the pageContext object is an instance of class javax.servlet.jsp.pagecontext.

The application object

The application object is a representation of JSP page through its life cycle. The application object is created when a JSP page is initialized and removed when the JSP page is removed by jspDestroy() method or JSP page is recompiled. As its name imply, the information of the application object is accessible to any object used within the JSP page.

The config object

The config object allows you to access the initialization parameters for the Servlet and JSP engine. The config object is an instance of the class javax.servlet.ServletConfig.

The page object

The page object is an instance of a JSP page. By using the page object, you can call any method of the page's servlet.

The exception object

The exception object contains the exception which is thrown from the previous JSP page. You can use the exception object to generate friendly error message based on erorr condition to the end-user.

10.2.4 Conditional Processing

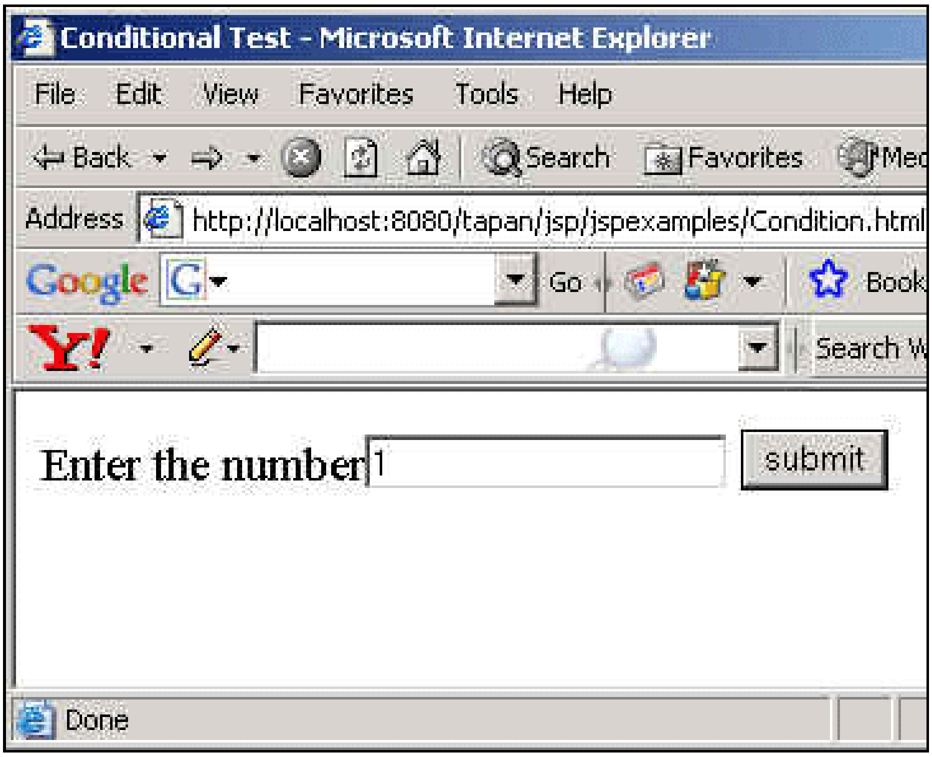
We make use of the condition to check if the entered value is correct. It correct then what will be the output, and in case if it is wrong then what output will be displayed to the browser.

In this program we are using the switch statement to check the conditions. In this program we are making the using of html form. When the values are entered in the html form then these values are checked by the controller, then on checking the values the output will be displayed to the user. The code of the program is given below:

```
<html>
<head>
<title>Conditional Test</title>
</head>
<body>
<form method = "get" action = "Conditional.jsp">
 Enter the number<input type  = "text" name = "number" value = "">
<input type = "submit" name = "submit" value = "submit">
</form>
</body>
```

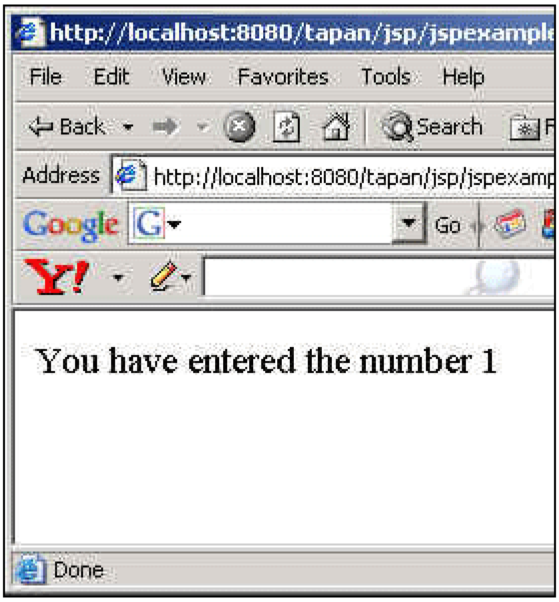
```
</html>
<% switch (Integer.parseInt(request.getParameter("number")))
{
      case 0: %>
         You have entered the number 0
   <%    break;
      case 1: %>
         You have entered the number 1
   <%    break;
      default: %>
         You can enter either o or 1
   <%
}
%>
```

Output :


Conditional Test - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back
Search
Favorites
Med
Address
http://localhost:8080/tapan/jsp/jspexamples/Condition.html
Google
Go
Book
Y!
Search W
Done

Enter the number 1 submit

NOTES


http://localhost:8080/tapan/jsp/jspexampl
File Edit View Favorites Tools Help
Back
Search
Address http://localhost:8080/tapan/jsp/jspexamp
Google
Go
Done

You have entered the number 1

10.2.5 Display values Using an Expression to Set an Attribute

In this section, We will simply output "Hello JSP" in a web browser using Jsp expression. JSP expressions insert values directly into the output. The syntax to include a JSP expressions in the JSP file is:

<%= expression %>.

In JSP page, we can use both, static HTML with dynamically-generated HTML. In this program, we produce output "hello Jsp" using this tag :

<%= "Hello World!" %>

All the other content except this is simple HTML code.

```
<html>
        <head><title>Hello World JSP Page.</title></head>
        <body>
                <font size="10"><%="Hello JSP!" %></font>
        </body>
</html>
```

10.2.6 Declaring Variables and Methods

To add a declaration, you must use the <%! and %> sequences to enclose your declarations, as shown below.

```
<%@ page import="java.util.*" %>
<HTML>
<BODY>
<%!
   Date theDate = new Date();
   Date getDate()
   {
      System.out.println( "In getDate() method" );
      return theDate;
   }
%>
Hello!  The time is now <%= getDate() %>
</BODY>
</HTML>
```

The example has been created a little contrived, to show variable and method declarations.

Example 2 :

In this example we are going to show you how we can declare a method and how we can use it. In this example we are making a method named as addNum(int i, int b) which will take two numbers as its parameters and return integer value. We have declared this method inside a declaration directive i.e. <%! -----------%> this is a declaration tag. This tag is used mainly for declaration the variables and methods.

In this method we are adding to numbers. To print the content of the method we are using scriptlet tag inside which we are going to use the out implicit object .<% ------- %> This tag is known as Scriptlets . The main purpose of using this tag is to embed a java code in the jsp page.

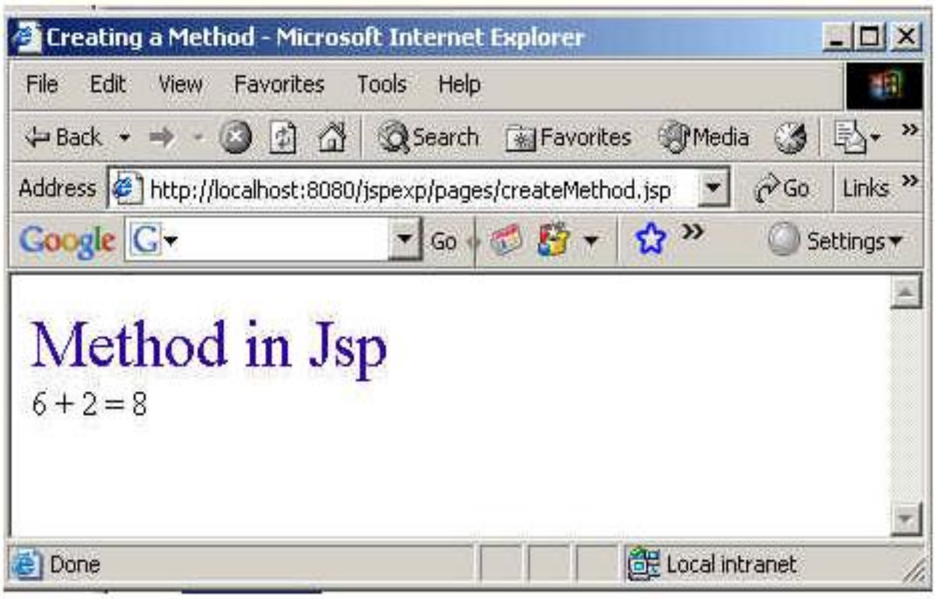
The code:

```
<HTML>
  <HEAD>
    <TITLE>Creating a Method</TITLE>
  </HEAD>
  <BODY>
    <font  size="6" color ="#330099"> Method in Jsp </font><br>
    <%!
    int addNum(int n, int m)
    {
      return n + m;
```

NOTES

```
    }
    %>
    <%
        out.println("6 + 2 = " + addNum(6, 2));
    %>
  </BODY>
</HTML>
```

Output



Creating a Method - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites Media
Address http://localhost:8080/jspexp/pages/createMethod.jsp Go Links
Google Go Settings
Done
Local intranet

Method in Jsp

6 + 2 = 8

10.3 Check Your Progress

1. Write True or False for the following :

(a) The JSP declaration is surrounded by the sign <%! and %>.

(b) JSP Scripting doesn’t allows you to insert Java code into the servlet.

10.4 Answers to Check Your Progress Questions

1. (a) True (b) False.

10.5 Summary

We have learnt about Implicit JSP Objects and its types. We have also learnt how to Declaring Variables and Methods. JSP Scripting allows you to insert Java code into the servlet which is generated from JSP page. These are the forms of scripting elements such as: comment, expression, scriptlet, and declaration and expression language. JSP container provides a list of instantiated objects for you to access different kind of data in a web application.

10.6 Key Words

JSP- JSP pages allow you to separate front-end presentation from business logic (middle and back-end tiers). It is a great Rapid Application Development (RAD) approach to Web applications.

10.7 Self-Assessment Questions and Exercises

Q: What are implicit objects? List them?

A: Certain objects that are available for the use in JSP documents without being declared first. These objects are parsed by the JSP engine and inserted into the generated servlet. The implicit objects re listed below

- request
- response
- pageContext
- session
- application
- out
- config
- page
- exception

10.8 Further Readings

- Hans Bergsten, Second Edition,Java Server Pages, O'Reilly Media, August 2002.
- Phil Hanna, The Complete reference JSP 2.0, Tata McGraw Hill,Delhi.

UNIT 11
Error Handling and Debugging

Structure

11. 0 Introduction

You usually won't be able to find all the errors in your Web application no matter how much debugging you do. Eventually, an error is going to crop up somewhere. In a servlet, you can put a try-catch block around your code to make sure the servlet doesn't throw an exception but goes to an error-handling page instead. For a Java Server Page, you might be able to insert a try-catch block, but there's a better solution. You can specify an error page for each JSP.

11. 1 Objectives

After going through the unit, you will be able to;

- Know Error Handling and Debugging
- Sharing Data between JSP pages, Requests, and Users
- Passing Control and Data between Pages
- Sharing Session and Application Data
- Understand Memory Usage Considerations

11.2 Error Handling and Debugging

Because Java Server Pages eventually become servlets, you can debug a JSP by following the same procedure you use to debug servlets. The only catch is that you must find out the name of the servlet first.

When Tomcat translates a JSP into a servlet, it puts the generated servlet code into the work directory underneath the main Tomcat directory. This is where you need to look for a JSP's servlet. Although the filenames and classnames of the servlets are roughly the same as the pathname for the JSP, you might find that tomcat represents some characters using a 4-digit hexadecimal code.

For example, the URL

http://localhost:8080/jspbook/ch02/examples/HelloWorld.jsp is stored by Tomcat version 3.1 in a directory named localhost_8080%2Fjspbook.

The servlet is in a file named

_0002fch_00030_00032_0002fexamples_0002fHelloWorld_
0002ejspHelloWorld_jsp_0.java.

Although this might be a lot of typing, at least you can debug the JSP. Also keep in mind that when you debug a JSP, you're actually debugging a servlet. The generated servlet code might not match your JSP code exactly. You might find a development environment that allows you to debug JSP files directly. As servlets and Java Server Pages become more popular, IDE developers will provide more support for them in the various development environments.

Error Handling

You usually won't be able to find all the errors in your Web application no matter how much debugging you do. Eventually, an error is going to crop up somewhere. In a servlet, you can put a try-catch block around your code to make sure the servlet doesn't throw an exception, but goes to an error-handling page instead.

Specifying an Error Page for a JSP

For a Java Server Page, you might be able to insert a try-catch block, but there's a better solution. You can specify an error page for each JSP. Whenever the page throws an exception, the JSP engine automatically invokes the error page. The error page can then log the exception (which is passed via the exception built-in variable) and display some sort of response to the user. To set up an error page, use the <%@ page errorPage="xxx" %> directive. For example, code 1 shows a Java Server Page that intentionally throws an exception.

Code 1 for ThrowError.jsp

```
<%@ page errorPage="ShowError.jsp" %>
<html>
<body>
<h1> Hello World! </h1>
</body>
</html>
<%
// Throw an exception to invoke the error page
// Do a stupid expression to keep the compiler
// from generating "Statement Not Reached" errors
   int x = 1;
   if (x == 1)
   {
      throw new RuntimeException("Oh no!!!");
   }
%>
```

Code 2 shows the error page that was specified in code 1.

Notice that the error handling page includes the directive <%@ page isErrorPage="true" %>. This directive causes the JSP compiler to generate the exception instance variable.

Code 2 for ShowError.jsp

```
<%@ page isErrorPage="true" %>
<html>
<body bgcolor="#ffffff">
<h1>Error</h1>
Sorry, an error occurred.
<p>
Here is the stack trace:
<p>
<pre>
<% exception.printStackTrace(new PrintWriter(out)); %>
</pre>
</body>
</html>
```

11.3 Sharing Data between JSP pages, Requests, and Users

An application consists of more than a single page, and multiple pages often need access to the same information and server-side resources. When multiple pages process the same request (e.g., one page that retrieves the data the user asked for and another that displays it) there must be a way to pass data from one page to another. In an application in which the user is asked to provide information in multiple steps, such as an online shopping application, there must be a way to collect the information received with each request and get access to the complete set when the user is ready. Other information and resources need to be shared among multiple pages, requests, and all users. Examples are information about currently logged-in users, database connection pool objects, and cache objects to avoid frequent database lookups.

11.4 Passing Control and Data between Pages

Using different JSP pages as Controller and View means that more than one page is used to process a request. To make this happen, you need to be able to do two things:

- Pass control from one page to another
- Pass data from one page to another

11.4.1 Passing Control between Pages

<jsp:forward> action transfers the control to a static or dynamic resource. The static or dynamic resource to which control is transferred is represented as a URL. The user can forward the control to an HTML file, another JSP file, or a servlet. It should be noted that the target file must be in the same application context as the forwarding JSP file. For example,

```
<html>
<%
  double freeMem = Runtime.getRuntime().freeMemory();
  double totlMem = Runtime.getRuntime().totalMemory();
  double percent = freeMem/totlMem;
  if (percent < 0.5) {
%>
<jsp:forward page="/forward/one.jsp"/>
<% } else { %>
<jsp:forward page="two.html"/>
<% } %>
</html>
```

NOTES

11.4.2 Pass data from one page to another

JSP provides different scopes for sharing data objects between pages, requests, and users. The scope defines how long the object is available and whether it's available only to one user or to all application users. The following scopes are defined: page, request, session, and application.

The <jsp:useBean> action has a scope attribute you use to specify the scope for the bean.

The <jsp:useBean> action ensures that the bean already exists in this scope or that a new one is created and placed in the specified scope. It first looks for a bean with the name specified by the id attribute in the specified scope. If it already exists, for instance created by a previously invoked <jsp:useBean> action or by a servlet, it does nothing.1 If it can't find it, it creates a new instance of the class specified by the class attribute and makes it available with the specified name within the specified scope.

Example :

Jsp code

```
<HTML>
<HEAD>
 <TITLE>Getting a Property Value</TITLE>
</HEAD>
<BODY>
<H1>Getting a Property Value</H1>
<jsp:useBean id="bean1" class="beans.Test" /> The message is:
<jsp:getProperty name="bean1" property="message" />
<BR>
 <jsp:setProperty name="bean1" property="message" value="Hello again!"
/>
Now the message is: <jsp:getProperty name="bean1" property="message"
/>
 </body>
</html>
```

Test.java

```
package beans;
import java.io.Serializable;
public class Test implements Serializable
{
   public Test()
   {
    }
    private String message = "Hello from JSP!";
    public void setMessage(String message)
    {
       this.message = message;
    }
    public String getMessage()
    {
       return this.message;
    }
}
```

11.5 Sharing Session and Application Data

The request scope makes data available to multiple pages processing the same request. But in many cases, data must be shared over multiple requests. Some information is needed by multiple pages independent of who the current user is. JSP supports access to this type of shared information through the application scope. Information saved in the application scope by one page can later be accessed by another page, even if the two pages were requested by different users.

Keeping track of which requests come from the same user isn't as easy as it may look. HTTP is a stateless, request-response protocol. What this means is that the browser sends a request for a web resource; the web server processes the request and returns a response. The server then forgets this transaction ever happened. So when the same browser sends a new request; the web server has no idea that this request is related to the previous one. This is fine as long as you're dealing with static files, but it's a problem in an interactive web application.

There are two ways to solve this problem, and they have both been used extensively for web applications with a variety of server-side technologies. The server can either return all information related to the current user (the client state) with each response and let the browser send it back as part of the next request, or it can save the state somewhere on the server and send back only an identifier that the browser returns with the next request. The identifier is then used to locate the state information saved on the server. In both cases, the information can be sent to the browser in one of three ways:

- As a cookie
- Embedded as hidden fields in an HTML form
- Encoded in the URLs in the response body, typically as links to other application pages (this is known as URL rewriting)

Cookie

A cookie is a name/value pair that the server passes to the browser in a response header. The browser stores the cookie for the time specified by the cookie's expiration time attribute. When the browser sends a request to a server, it checks its "cookie jar" and includes all cookies it has received from the same server (that have not yet expired) in the request headers. Cookies used for state management don't have an expiration time and expire as soon as the user closes the browser. Using cookies is the easiest way to deal with the state issue, but some browsers don't support cookies.

URL rewriting

The session ID needed to keep track of requests within the same session can be transferred between the server and the browser in a number of different ways. One way is to encode it in the URLs created by the JSP pages. This is called URL rewriting.

11.6 Memory Usage Considerations

We should be aware that all objects you save in the application and session scopes take up memory in the server process. It's easy to calculate how much memory is used for the application scope because you have full control over the number of objects you place there. But the total number of objects in the session scope depends on the number of concurrent sessions, so in addition to the size of each object; you also need to know how many concurrent users you have and how long a session lasts.

Points to consider keeping the memory requirements:

- Place only objects that really need to be unique for each session in the session scope.

- Set the timeout period for sessions to a lower value than the default. If you know it's rare that your users leave the site for 30 minutes and then return, use a shorter period. You can change the timeout for all sessions in an application through the application's deployment descriptor, or by calling session.setMaxInactiveInterval( ) in a custom action, bean, or servlet to change it for an individual session.

- Provide a way to end the session explicitly. A good example is a logout function, or invalidation of the session when something is completed (for instance when an order form is submitted). In a JSP page, you can use the <ora:invalidateSession> custom action to invalidate the session. In a servlet or other custom code, you can use the HttpSession invalidate( ) method. Invalidating a session makes all objects available for garbage collection.

11.7 Check Your Progress

1. Write True or False for the following :

   (a) <jsp:forward> action transfers the control to a static or dynamic resource.

   (b) Invalidate session makes all objects available for garbage collection.

11.8 Answers to Check Your Progress Questions

1. (a) True (b) True

11.9 Summary

Error Handling and Debugging, Sharing Data between JSP pages, Requests, and Users, Sharing Session and Application Data and Memory Usage Considerations are also discussed.

11.10 Key Words

Cookie

A cookie is a name/value pair that the server passes to the browser in a response header.

URL rewriting

The session ID needed to keep track of requests within the same session can be transferred between the server and the browser in a number of different ways. One way is to encode it in the URLs created by the JSP pages. This is called URL rewriting.

11.11 Self-Assessment Questions and Exercises

Q: What is a Scriptlet?

A: A scriptlet can contain any number of language statements, variable or method declarations, or expressions that are valid in the page scripting language.Within scriptlet tags, you can

1.Declare variables or methods to use later in the file (see also Declaration).

2.Write expressions valid in the page scripting language (see also Expression).

3.Use any of the JSP implicit objects or any object declared with a <jsp:useBean> tag.
You must write plain text, HTML-encoded text, or other JSP tags outside the scriptlet.

Scriptlets are executed at request time, when the JSP engine processes the client request. If the scriptlet produces output, the output is stored in the out object, from which you can display it.

Q: What are the different scope valiues for the <jsp:useBean>?

A: The different scope values for <jsp:useBean> are
1. page
2. request
3.session
4.application.

11.12 Further Readings

1. Thomas A. Powell, HTML: the complete reference, McGraw-Hill, 2001.
2. Chris Bates, Web Programming, Building Internet Applications, 2nd Edition, Dreamtech
3. Patrick Naughton and Herbert Schildt, The complete Reference Java 2, 5th Edition, Tata McGraw Hill.
4. Jason Hunter Java Servlet Programming, O’Reilly
5. Hans Bergsten, Java Server Pages, O’Reilly
6. Ramesh Bangia, Web Technology, Firewall media, 2006.

BLOCK V

DATABASE ACCESS AND STRUCTS FRAMEWORK

UNIT 12

Database Access

Structure

12.0 Introduction

A database is a collection of information that is organized so that it can easily be accessed, managed, and updated. In one view, databases can be classified according to types of content: bibliographic, full-text, numeric, and images.

In computing, databases are sometimes classified according to their organizational approach. The most prevalent approach is the relational database, a tabular database in which data is defined so that it can be reorganized and accessed in a number of different ways. Java Database Connectivity or in short JDBC is a technology that enables the java program to manipulate data stored into the database. It is a java API which enables the java programs to execute SQL statements.

12.1 Objectives

After going through this lesson, you would be able to:

- Know about the database programming
- understand the need for JDBC Drivers
- list and use the types of JDBC Drivers
- know the java.sql.* package
- to connect front end with backend database using JDBC
- Access a Database from a JSP Page

12.2 Database Programming using JDBC

JDBC is similar to Open Database Connectivity (ODBC) which is used for accessing and managing database, but the difference is that JDBC is designed specifically for Java programs, whereas ODBC is not depended upon any language. It is an application programming interface that defines how a java programmer can access the database in tabular format from Java code using a set of standard interfaces and classes written in the Java programming language.

The Java application programming interface provides a mechanism for dynamically loading the correct Java packages and drivers and registering them with the JDBC Driver Manager that is used as a connection factory for creating JDBC connections which supports creating and executing statements such as SQL INSERT, UPDATE and DELETE. Driver Manager is the backbone of the JDBC architecture. In short JDBC helps the programmers to write java applications that manage these three programming activities:

1. It helps us to connect to a data source, like a database.
2. It helps us in sending queries and updating statements to the database
3. Retrieving and processing the results received from the database in terms of answering to your query.

12.2.1 JDBC Drivers:

JDBC Drivers are set of classes that enable the Java application to communicate with the Databases. Java.sql contains various classes for using relational databases. But these classes do not provide any implementation. JDBC Drivers are available for most database platforms.

12.2.2 Types of JDBC Drivers:

JDBC Drivers are divided into four types, each type defines a JDBC driver implementation with higher level of platform independence, performance and deployment administration. The four types of JDBC Drivers are:

i. *Type 1 : JDBC – ODBC Bridge*
ii. *Type 2 : Native API / Partly java technology enabled driver.*
iii. *Type 3 : Net Protocol / all–java driver.*
iv. *Type 4 : Native protocol / all – java driver.*

Type 1 : JDBC – ODBC Bridge

It translates all JDBC calls into ODBC calls and sends them to ODBC Drivers. The JDBC – ODBC Bridge, from Sun and InterSolv is the only existing example of Type 1 Driver.

Merits:

JDBC – ODBC Bridge allows us to almost any database since the database ODBC Drivers are already available.

Demerits:

- The performance is degraded, since the JDBC Calls goes through the data bridge to the native database [i.e] Type 1 drivers may not be useful for large scale application.
- The ODBC Driver must already be installed on the client machine.

Type 2 : Native API / Partly java technology enabled driver.

It converts JDBC calls into the database specific calls for the databases such as sql server, informatics, oracle or cybase. Type 2 drivers are implemented with Native code. Type 2 Drivers communicates directly with the drivers. [it requires some binary code]

Merits:

It offers significantly better performance than the JDBC – ODBC Bridge.

Demerits:

- The vendor database library needs to be loaded on each client machine.
- Type 2 drivers cannot be used for internet.

Type 3 : Net Protocol / all–java driver.

Type 3 follows, a three-tiered approach whereby the JDBC Database requests are passed through the network to the middle tier server. The middle tier server then translates the request to the database specific native connectivity interface, to further the request to the database server. Example: BEA's Weblogic. These servers are especially used for applet deployment.

Merits:

- There is no need for any vendor database library.
- It is highly portable, performance and scalable. It is used for very small and fast to load application.

Demerits:

Traversing the RecordSet may take long sinvce the data comes through the back end server.

Type 4 : Native protocol / all – java driver.

Type 4 Drivers are written entirely in java. They understand database specific networking protocols and can access database directly without any additional software.

Merits:

- There is no need to install special software on the client or the server.
- These drivers can be downloaded dynamically.
- Better performance than type 1 and type 3.

Demerits:

The user needs a different driver for each database.

JDBC Vs ODBC

1. ODBC is not appropriate for direct use from java. Because it uses 'C' interface. Calls from java to native C – code, a number of drawbacks in

the security, implementation, robustness and automatic portability of applications.

2. A literal translation of ODBC – C – API into a java API would not be desirable. For eg: Java has no pointers. But ODBC makes copious use of them. You can use JDBC as ODBC translated into an object oriented interface (i.e) natural for java programmers.

3. ODBC is hard to learn. It mixes simple and advanced features together and it has complex options even for simple queries. JDBC was designed to keep simple things simple, while allowing more advanced capabilities where required.

4. When ODBC is used, the ODBC driver manager and drivers must be manually installed on every client machine. JDBC is automatically installable, portable and secure on any java platform.

12.3 Java.sql package

This package provides the APIs for accessing and processing data which is stored in the database especially relational database by using the java programming language. It includes a framework where we different drivers can be installed dynamically to access different databases especially relational databases.

Java.sql package contains the following set of interfaces and classes for easy implementation of database access.

Interfaces

- Callable Statement

  Execution of sql stored procedure. Both IN and OUT parameters are supported

- Connection

  Maintenance and status monitors of database sessions

- Database Metadata

  Information regarding database itself such as version information, table names, supported functions, etc.

- Driver

  To create Connection objects, collection of JDBC Driver MetaData and JDBC Driver Status Checking.

- PreparedStatement

  To execute pre-complied sql statements.

- ResultSet:

  It provides methods for retrieval of data returned by SQL Statement execution, contains methods for sql data type and JDBC Data type Conversion.

- ResultSet Metadata:

  It is used for collection of metadata information associated with last result set object.

- Statement:

  To execute sql statements and retrieve data into the result set.

Classes:

- Date class: It contains methods to perform sql date formats and java date objects.
- DriverManager: To handle the loading and unloading of drivers and establish the connection with the database.
- DriverPropertyInfo: For the retrieval of or insertion of driver properties.
- Time: To perform sql time and java time object.
- Conversions: This class is similar to date class.
- TimeStamp: Additional precision to the java date object by adding a nanosecond field.
- Type : Defines the constant used to identify sql types.

This java.sql package contains API for the following:

1) Making a connection with a database with the help of DriverManager class

a) DriverManager class:
It helps to make a connection with the driver.

b) SQLPermission class:
It provides a permission when the code is running within a Security Manager, such as an applet. It attempts to set up a logging stream through the DriverManager class.

c) Driver interface :
This interface is mainly used by the DriverManager class for registering and connecting drivers based on JDBC technology.

d). DriverPropertyInfo class :
This class is generally not used by the general user.

2). Sending SQL Parameters to a database:

a). **Statement interface:**
It is used to send basic SQL statements.

b). PreparedStatement interface:
It is used to send prepared statements or derived SQL statements from the Statement object.

c). CallableStatement interface :
This interface is used to call database stored procedures.

d). Connection interface :
It provides methods for creating statements and managing their connections and properties.

e). Savepoint :
It helps to make the savepoints in a transaction.

3). Updating and retrieving the results of a query:

a). **ResultSet interface:** This object maintains a cursor pointing to its current row of data. The cursor is initially positioned before the first row. The next method of the resultset interface moves the cursor to the next row and it will return false if there are no more rows in the ResultSet object. By default ResultSet object is not updatable and has a cursor that moves forward only.

4.) Providing Standard mappings for SQL types to classes and interfaces in Java Programming language.

5). Metadata

a). DatabaseMetaData interface: It keeps the data about the data. It provides information about the database.

NOTES

b). ResultSetMetaData: It gives the information about the columns of a ResultSet object.
c). ParameterMetaData: It gives the information about the parameters to the PreparedStatement commands.

6). Exceptions

a). SQLException: It is thrown by the mehods whenever there is a problem while accessing the data or any other things.
b). SQLWarning: This exception is thrown to indicate the warning.
c). BatchUpdateException: This exception is thrown to indicate that all commands in a batch update are not executed successfully.
d). DataTruncation: It is thrown to indicate that the data may have been truncated.

7). Custom mapping an SQL user- defined type (UDT) to a class in the java programming language.

a) SQLData interface: It gives the mapping of a UDT to an intance of this class.
b) SQLInput interface: It gives the methods for reading UDT attributes from a stream.
c) SQLOutput: It gives the methods for writing UDT attributes back to a stream.

The JDBC Driver Manager

The JDBC Driver Manager is a very important class that defines objects which connect Java applications to a JDBC driver. By calling the Class.forName() method the driver class get automatically loaded. The driver is loaded by calling the Class.forName() method. JDBC drivers are designed to tell the DriverManager about themselves automatically when their driver implementation class get loads.

This class has many methods. Some of the commonly used methods are given below:

1. deregisterDriver(Driver driver) : It drops the driver from the list of drivers registered in the DriverManager class.
2. registerDriver(Driver driver) : It registers the driver with the DriverManager class.
3. getConnection(String url) : It tries to establish the connection to a given database URL.
4. getConnection(String url, Sting user, String password) : It tries to establish the connection to a given database URL.

5. getConnection(String url, Properties info) : It tries to establish the connection to a given database URL.
6. getDriver(String url) : It attempts to locate the driver by the given string.
7. getDrivers() : It retrieves the enumeration of the drivers which has been registered with the DriverManager class.

The JDBC API provides the DataSource interface as an alternative to the DriverManager for establishing the connection. A DataSource object is the representation of database or the data source in the Java programming language. DataSource object is mostly preferred over the DriverManager for establishing a connection to the database.

DataSource object can be thought as a factory for making connections to the particular database that the DataSource instance represents.

A Connection object represents a connection with a database. When we connect to a database by using connection method, we create a Connection Object, which represents the connection to the database.

We can use the Connection object for the following things:

1). It creates the Statement, PreparedStatement and CallableStatement objects for executing the SQL statements.

2). It helps us to Commit or roll back a jdbc transactionn.

3). If you want to know about the database or data source to which you are connected then the Connection object gathers information about the database or data source by the use of DatabaseMetaData.

4). It helps us to close the data source. The Connection.isClosed() method returns true only if the Connection.close() has been called. This method is used to close all the connection.

Syntax:

```
String url = "jdbc: odbc: makeConnection";
Connection con = DriverManager.getConnection(url, "userID",
"password");
```

Example program to retrieve records from database and display the results

Before explaining you the JDBC Steps for making connection to the database and retrieving the employee from the tables, we will provide you

NOTES

the structure of the database and sample data.

1)Creating Database table use any RDBMS

Employee table

Id	**Name**	**shift**
101	Kumar	1
105	Rajan	2
109	Sekaran	1

2) Creating a new ODBC Data Source Name (DSN)

It is a straight-forward operation. If you're connecting to SQL Server via Microsoft Access or a scripting language such as Perl using Win32:ODBC, you'll need a valid DSN. Using the OBDC Data Source Administrator, you can create User, System, and File DSNs as you need them.

Data Source Name Categories

ODBC has three different categories or types of DSNs:

- User DSN
- System DSN
- File DSN

Each DSN category serves a specific purpose and has a specific scope.

User DSN

A user DSN is just that, a DSN for a specific user. If I create a user DSN under my user account, no other user can see it or use it. The DSN is for me and me alone. If you need a connection to a data source that only you should use, choose a user DSN.

System DSN

A system DSN is a DSN that is seen by the entire system. Any user can see it, as well as any process or service. If you need a data source connection that should be seen more than just your user account, choose to use a system DSN. This is especially true if you are trying to establish a connection through IIS or some other service.

File DSN

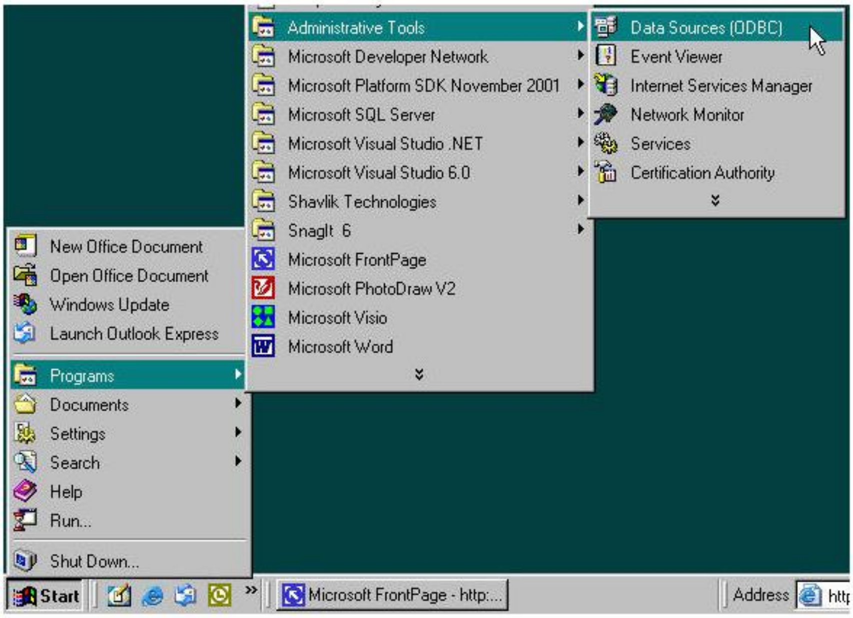
A file DSN is simply where the connection settings are written to a file. The reason for having a file DSN is if you want to distribute a data source connection to multiple users on different systems without having to configure a DSN for each system. For instance, I can create a file DSN to a reporting database on my desktop. I can then send the file to my users. My users can save the file DSN to their hard drives and then point their reporting applications at the file DSN.

Creating a DSN

I'll be creating a simple user DSN, but the process is basically the same for a file or system DSN. The only difference with a file DSN is you'll be prompted to save the DSN (since you're saving to a file), a step that isn't necessary with the other two DSNs.

Open the Data Source Administrator

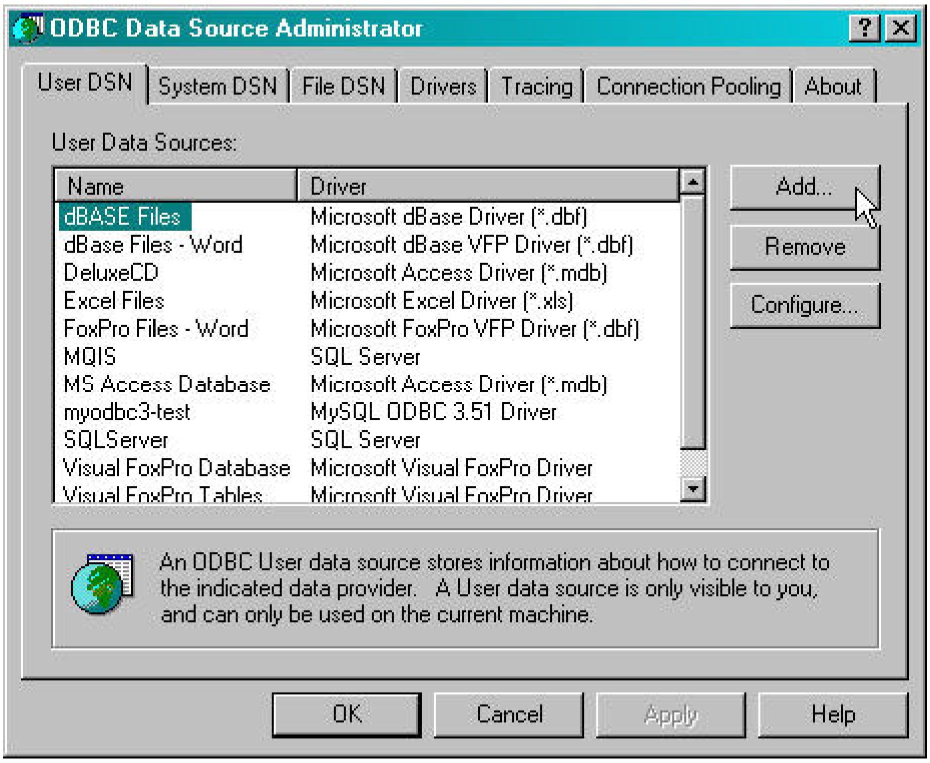
The first step in creating a DSN is to open us the Data Source Administrator. On Windows 2000 Server systems (and other Server builds), one method to navigate via Start | Programs | Administrative Tools | Data Sources (ODBC) as in Figure.



Choose Data Sources (ODBC) and that'll bring up the Data Source Administrator.

Adding a New DSN

Once the Data Source Administrator opens up, choose the tab appropriate for the DSN you want to create. we leave mine as a User DSN since we just showing how to create a DSN in the first place. To begin creating a new DSN, choose the Add... as in Figure

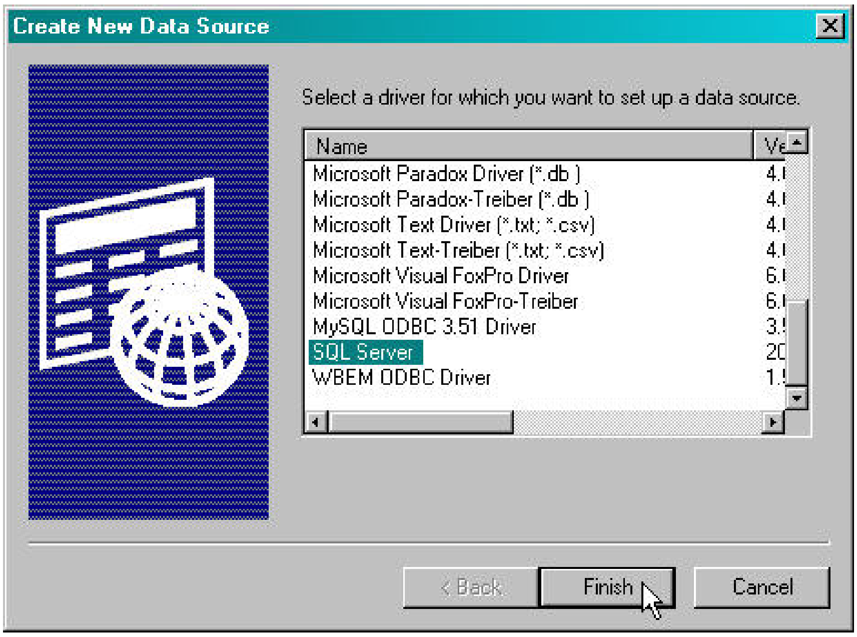


Name	Driver
dBASE Files	Microsoft dBase Driver (*.dbf)
dBase Files - Word	Microsoft dBase VFP Driver (*.dbf)
DeluxeCD	Microsoft Access Driver (*.mdb)
Excel Files	Microsoft Excel Driver (*.xls)
FoxPro Files - Word	Microsoft FoxPro VFP Driver (*.dbf)
MQIS	SQL Server
MS Access Database	Microsoft Access Driver (*.mdb)
myodbc3-test	MySQL ODBC 3.51 Driver
SQLServer	SQL Server
Visual FoxPro Database	Microsoft Visual FoxPro Driver
Visual FoxPro Tables	Microsoft Visual FoxPro Driver

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you, and can only be used on the current machine.

Choosing the Server Driver

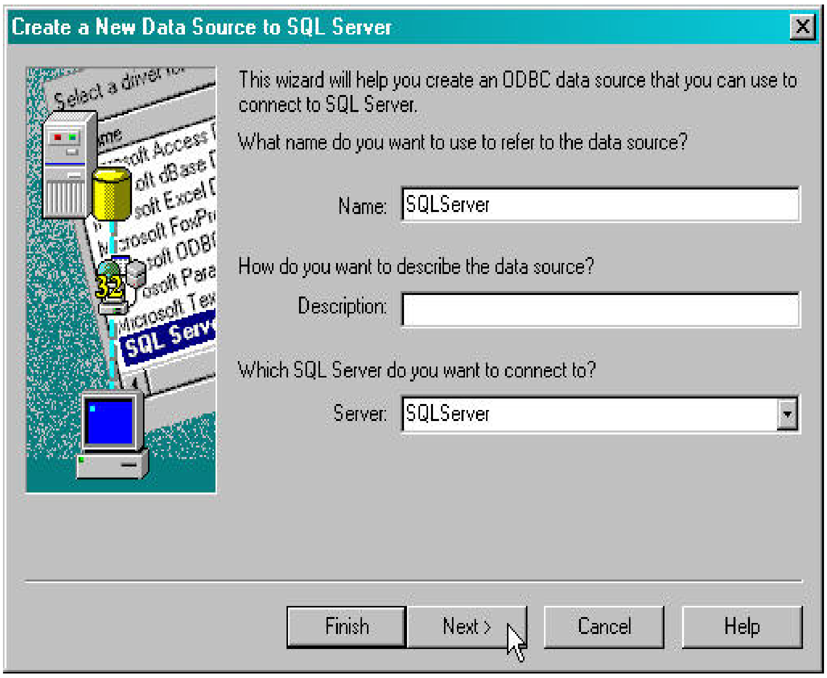
Once we have chosen to add a new DSN, the first thing I'll have to pick is the correct driver. Since we connect to a Microsoft SQL Server database, we will choose the SQL Server driver. This tends to be towards the bottom of the driver list, as in Figure



Select a driver for which you want to set up a data source.

Name	Ve
Microsoft Paradox Driver (*.db)	4.
Microsoft Paradox-Treiber (*.db)	4.
Microsoft Text Driver (*.txt; *.csv)	4.
Microsoft Text-Treiber (*.txt; *.csv)	4.
Microsoft Visual FoxPro Driver	6.
Microsoft Visual FoxPro-Treiber	6.
MySQL ODBC 3.51 Driver	3.
SQL Server	2
WBEM ODBC Driver	1.

Naming the DSN and Picking the Data Source and choose the authentication method



This wizard will help you create an ODBC data source that you can use to connect to SQL Server.

What name do you want to use to refer to the data source?

Name: SQLServer

How do you want to describe the data source?

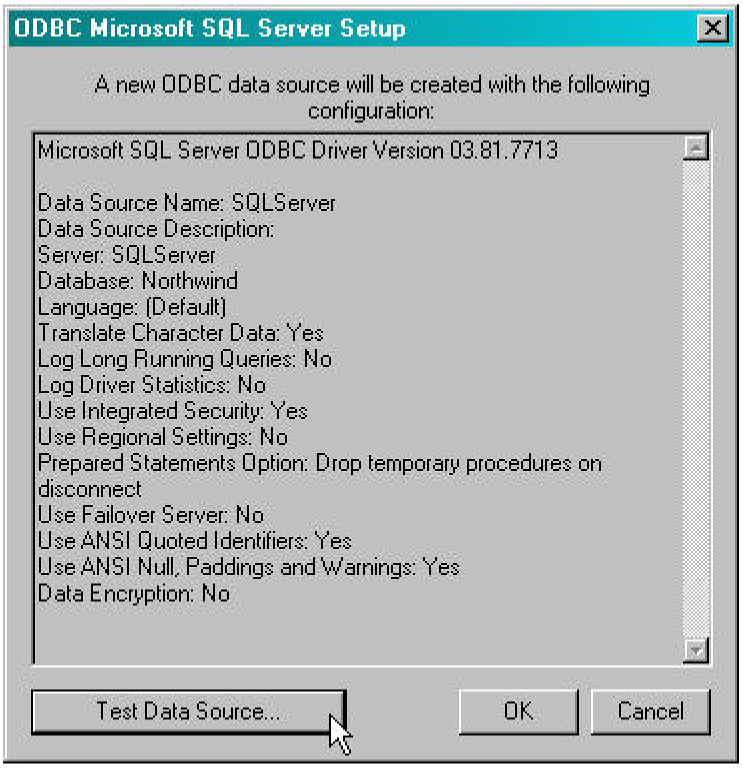
Description:

Which SQL Server do you want to connect to?

Server: SQLServer

Choose the additional settings and test the data

NOTES



A new ODBC data source will be created with the following configuration:

```
Microsoft SQL Server ODBC Driver Version 03.81.7713

Data Source Name: SQLServer
Data Source Description:
Server: SQLServer
Database: Northwind
Language: (Default)
Translate Character Data: Yes
Log Long Running Queries: No
Log Driver Statistics: No
Use Integrated Security: Yes
Use Regional Settings: No
Prepared Statements Option: Drop temporary procedures on
disconnect
Use Failover Server: No
Use ANSI Quoted Identifiers: Yes
Use ANSI Null, Paddings and Warnings: Yes
Data Encryption: No
```

And that's all there is to creating a new ODBC Data Source Name. In this example the connection dsn string used is **demo**.

databasetest.java

```
// To retrieve records using JDBC

 import java.sql.*;
 public class databasetest
  {
    public static void main(String args[])
    {
      try
      {
       Class.forName(“sun.jdbc.JdbcOdbcDriver”);
       Connection db = DriverManager.getConnection(“jdbc:odbc:demo”,
“sa”, “sa”);
       Statement s = db.createStatement();
      String str = “select * from emp”;
      ResultSet rs = s.executeQuery(str);
      while(rs.next())
      {
        int i = rs.getInt(“id”);
        String name = rs.getString(“name”);
        int shi = rs.getInt(“shift”);
        System.out.println(“id=”+i);
        System.out.println(“name=”+name);
        System.out.println(“shift=”+shift);
      }
    }catch(Exception e){}
  }
```

Output:

```
C:\>javac databasetest.java
C:\>java databasetest
        id=10
        name=Kumar
        shift=4
```

Using ‘Prepared Statement’ [Pre–Compiled Statements]:

PreparedStatement object contains not just an SQL Statement, but an SQL Statement that has been pre-compiled. If you want to execute Statement object many times,it will normally increase the execution time; to reduce the execution time, use a prepared statement object instead as

PreparedStatement update_sales = con.prepareStatement(“update COFFEES SET sales = ? where COF_NAME like ? ”)

PreparedStatement object is used to execute a statement object many times.

```
PreparedStatement update_sales = con.prepareStatement(“update
COFFEES SET sales = ? where COF_NAME like ?”)
int SalesForWeek = {175,150,60}\
String Coffees = { “Red”, “Bru”, “Sunrise”)
int len = Coffees.length;
for(i = 0;i<len;i++)
{
 update_sales.setInt(1,SalesForWeek[i]);
 update_sales.setString(2,Coffees[i]);
 update_sales.executeUpdate();
}
```

Stored Procedure: -

A procedure stored in database. A stored procedure is a set of SQL Statements that form a logical unit and perform a particular task.

Stored procedures are used to encapsulate a set of queries or set of

NOTES

operations to execute on a database server.

Stored procedures can be compiled and executed with different parameters. They may have any combination of input, output and input/output combinations.

SQL Statement for creating a stored procedure:

create procedure show_suppliers as
select suppliers.supname, coffees.cofname from suppliers, coffees where suppliers.supid = Coffees.supid order by supname

In java,

```
String cp = “create procedure show_suppliers ” + “as ” + “select
suppliers.supname…”
Statement st = con.createStatement();
st.executeUpdate(cp);
```

Calling Stored procedure from JDBC (Callable Statement)

JDBC allows you to call a database, stored procedure from an application written in the java programming language. The first step is to create a CallableStatement object. The CallableStatement object contains a call to stored procedure.

```
CallableStatement cs = con.prepareCall(“{call SHOWSUPPLIERS}”);
ResultSet rs = cs.executeQuery();
```

CallableStatement is a sub class of the PreparedStatement class.

12.4 Accessing Database from JSP

We can access the database available in any database software like MYSQL. we take a example of Books database. This database contains a table named books_details. This table contains three fields- id, book_name& author. we starts from very beginning. First we learn how to create tables in MySQl database after that we write a html page for inserting the values in 'books_details' table in database. After submitting values a table will be showed that contains the book name and author name.

Database

The database in example consists of a single table of three columns or fields. The database name is "books" and it contains information about books names & authors.

Table:books_details

ID	Book Name	Author
1.	Java 2 Complete reference	Partick Norton
2.	Java & XML,2 Edition	Brett McLaughlin
3.	Web Technology	Ramesh

Start MYSQL prompt and type this SQL statement & press Enter-

MYSQL>CREATE DATABASE `books` ;

This will create "books" database.
Now we create table a table "books_details" in database "books".

```
MYSQL>CREATE TABLE `books_details` (
`id` INT( 11 ) NOT NULL AUTO_INCREMENT ,`book_name` VARCHAR(
100 ) NOT NULL , `author` VARCHAR( 100 ) NOT NULL , PRIMARY
KEY ( `id` )) TYPE = MYISAM ;
```

This will create a table "books_details" in database "books"

JSP Code

The following code contains html for user interface & the JSP backend-

```
<%@ page language="java" import="java.sql.*" %>
<%
Connection con=null;
ResultSet rst=null;
Statement stmt=null;
try{
      Class.forName("sun.jdbc.JdbcOdbcDriver");
       con=  DriverManager.getConnection("jdbc:odbc:demo", "sa", "sa");
          stmt=con.createStatement();
          }
          catch(Exception e){
                    System.out.println(e.getMessage());
          }
          if(request.getParameter("action") != null){
          String bookname=request.getParameter("bookname");
String author=request.getParameter("author");
stmt.executeUpdate("insert into books_details(book_name,
author) values('"+bookname+"','"+author+"')");
rst=stmt.executeQuery("select * from books_details");
%>
        <html>
        <body>
        <center>
        <h2>Books List</h2>
        <table border="1" cellspacing="0" cellpadding ="0">
                <tr>
                        <td><b>S.No</b></td>
                        <td><b>Book Name</b></td>
                        <td><b>Author</./b></td>
                </tr>
                        <%
                        int no=1;
                        while(rst.next()){
                        %>
                <tr>
                         td><%=no%></td>
                       <td>
                        <%=rst.getString("book_name")%>
                        </td>
                       <td>
                         <%=rst.getString("author") %>
                        </td>
                </tr>
                        <%
                                no++;
                                }
                                rst.close();
                                stmt.close();
                                con.close();
                        %>
                </table>
                </center>
        </body>
</html>
        <% } else { %>
<html>
<head>
        <title>Book Entry FormDocument</title>
        <script language="javascript">
           function validate(objForm){
                if(objForm.bookname.value.length==0){
```

```
                              alert("Please enter Book Name!");
                              objForm.bookname.focus();
                              return false;
                              }
                              if(objForm.author.value.length==0){
                              alert("Please enter Author name!");
                              objForm.author.focus();
                              return false;
                              }
                              return true;
                              }
                              </script>
                    </head>
                    <body>
                    <center>
<form action="BookEntryForm.jsp" method="post" name="entry"
onSubmit="return  validate(this)">
          <input type="hidden" value="list" name="action">
          <table border="1" cellpadding="0" cellspacing="0">
          <tr>
          <td>
          <table>
          <tr>
          <td colspan="2" align="center">
           <h2>Book Entry Form</h2></td>
          </tr>
          <tr>
          <td colspan="2">&nbsp;</td>
          </tr>
          <tr>
          <td>Book Name:</td>
          <td><input name="bookname" type="text" size="50"></td>
          </tr>
          <tr>
          <td>Author:</td><td><input name="author" type="text"
size="50">
</td>
</tr>
<tr>
<td colspan="2" align="center">
<input type="submit" value="Submit"></td>
</tr>
</table>
</td>
</tr>
</table>
</form>
```

```
</center>
</body>
</html>
<% }  %>
```

Steps for the Database access using JSP

Declaring Variables:

Java is a strongly typed language which means, that variables must be explicitly declared before use and must be declared with the correct data types. In the above example code we declare some variables for making connection. These variables are-

Connection con=null;
ResultSet rst=null;
Statement stmt=null;

The objects of type Connection, ResultSet and Statement are associated with the Java sql. "con" is a Connection type object variable that will hold Connection type object. "rst" is a ResultSet type object variable that will hold a result set returned by a database query. "stmt" is a object variable of Statement .Statement Class methods allow to execute any query.

Connection to database:

The first task of this programmer is to load database driver. This is achieved using the single line of code

Class.forName("sun.jdbc.JdbcOdbcDriver");
con= DriverManager.getConnection("jdbc:odbc:demo", "sa", "sa");

Executing Query or Accessing data from database:

stmt=con.createStatement(); //create a Statement object
*rst=stmt.executeQuery("select * from books_details");*

stmt is the Statement type variable name and rst is the RecordSet type variable. A query is always executed on a Statement object.A Statement object is created by calling createStatement() method on connection object con.

The two most important methods of this Statement interface are executeQuery() and executeUpdate(). The executeQuery() method executes an SQL statement that returns a single ResultSet object. The executeUpdate() method executes an insert, update, and delete SQL statement. The method returns the number of records affected by the SQL statement execution.

After creating a Statement, a method executeQuery() or executeUpdate() is called on Statement object stmt and a SQL query string is passed in method executeQuery() or executeUpdate().This will return a ResultSet rst related to the query string.

Reading values from a ResultSet:

```
while(rst.next())
{
  %>
<tr><td><%=no%></td><td><%=rst.getString("book_name")%></td><td
><%=rst.getString("author")%></td></tr>
  <%
}
```

The ResultSet represents a table-like database result set. A ResultSet object maintains a cursor pointing to its current row of data. Initially, the cursor is positioned before the first row. Therefore, to access the first row in the ResultSet, you use the next() method. This method moves the cursor to the next record and returns true if the next row is valid, and false if there are no more records in the ResultSet object.

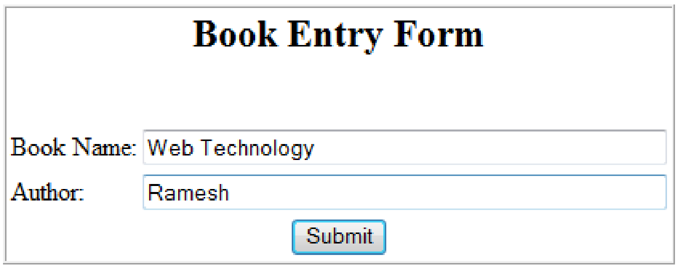
Other important methods are getXXX() methods, where XXX is the data type returned by the method at the specified index, including String, long, and int. The indexing used is 1-based. For example, to obtain the second column of type String, you use the following code:

resultSet.getString(2);

You can also use the getXXX() methods that accept a column name instead of a column index. For instance, the following code retrieves the value of the column LastName of type String.

resultSet.getString("book_name");

The above example shows how you can use the next() method as well as the getString() method. Here you retrieve the 'book_name' and 'author' columns from a table called 'books_details'. You then iterate through the returned ResultSet and print all the book name and author name in the format " book name | author " to the web page.

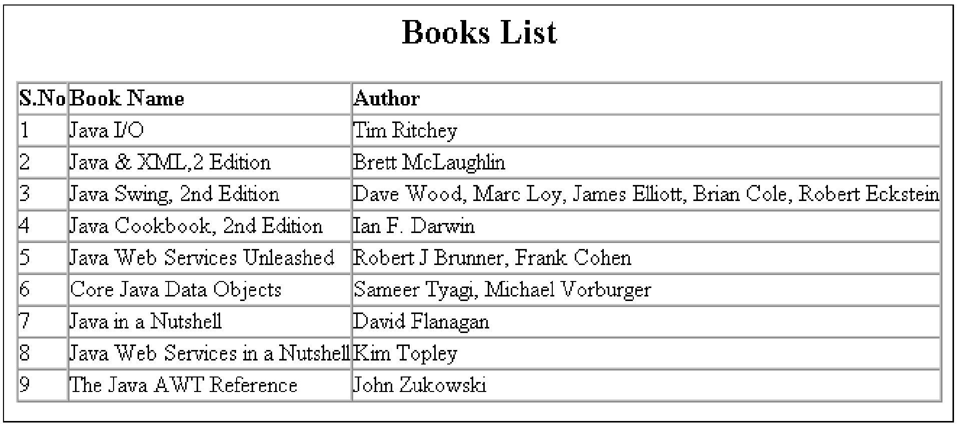


Book Entry Form

Book Name: Web Technology

Author: Ramesh

Fill the book name and author fields and press Submit button. A page will open and show a table of book name and authors like the following;



Books List

S.No	Book Name	Author
1	Java I/O	Tim Ritchey
2	Java & XML,2 Edition	Brett McLaughlin
3	Java Swing, 2nd Edition	Dave Wood, Marc Loy, James Elliott, Brian Cole, Robert Eckstein
4	Java Cookbook, 2nd Edition	Ian F. Darwin
5	Java Web Services Unleashed	Robert J Brunner, Frank Cohen
6	Core Java Data Objects	Sameer Tyagi, Michael Vorburger
7	Java in a Nutshell	David Flanagan
8	Java Web Services in a Nutshell	Kim Topley
9	The Java AWT Reference	John Zukowski

12.5 Check Your Progress

1) What are the steps involved in establishing a database connection?
2) How can you load the drivers?
3) What Class.forName will do while loading drivers?
4) How can you make the connection?

12.6 Answers to Check Your Progress Questions

1) This involves two steps: (1) loading the driver and (2) making the connection.

2) Loading the driver or drivers you want to use is very simple and involves just one line of code. If, for example, you want to use the JDBC-ODBC Bridge driver, the following code will load it:

Eg. *Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");*

Your driver documentation will give you the class name to use. For instance, if the class name is jdbc.DriverXYZ , you would load the driver with the following line of code: Eg.

```
        Class.forName("jdbc.DriverXYZ");
```

3) It is used to create an instance of a driver and register it with the DriverManager. When you have loaded a driver, it is available for making a connection with a DBMS.

4) In establishing a connection is to have the appropriate driver connect to the DBMS. The following line of code illustrates the general idea: Eg.

```
String url = "jdbc:odbc:Fred";
Connection con = DriverManager.getConnection(url, "Fernanda", "J8");
```

12.7 Summary

In this unit we have learnt about how to connect with database using Java Database Connectivity. We have learnt about steps involved in establishing a connection. We have also learnt about JDBC Drivers and its types.

12.8 Key Words

A **database** is a collection of information that is organized so that it can easily be accessed, managed, and updated.

JDBC is similar to Open Database Connectivity (ODBC) which is used for accessing and managing database, but the difference is that JDBC is designed specifically for Java programs, whereas ODBC is not depended upon any language

JDBC Drivers are set of classes that enable the Java application to communicate with the Databases.

12.9 Self-Assessment Questions and Exercises

1. Explain about Types of JDBC Drivers.
2. Define Database connection.
3. What are the Steps using Database access using JSP?
4. Discuss about Types of DSN.

NOTES

12.10 Further Readings

1. Thomas A. Powell, HTML: the complete reference, McGraw-Hill, 2001.
2. Chris Bates, Web Programming, Building Internet Applications, 2nd Edition, Dreamtech
3. Patrick Naughton and Herbert Schildt, The complete Reference Java 2, 5th Edition, Tata McGraw Hill.
4. Jason Hunter Java Servlet Programming, O'Reilly
5. Hans Bergsten, Java Server Pages, O'Reilly
6. Ramesh Bangia,Web Technology, Firewall media, 2006.

UNIT 13

Application – Specific Database Actions

Structure

13. 0 Introduction

You can use the JSTL (JSP Standard Tag Libraries) database actions to develop many types of interesting web applications, such as product catalog interfaces, employee directories, or online billboards, without being a Java programmer.

These types of applications account for a high percentage of the web applications developed today. But at some level of complexity, putting SQL statements directly in the web pages can become a maintenance problem.

13. 1 Objectives

After going through the unit, you will be able to;

- Know Application – Specific Database Actions
- Understand about Database access actions
- Know about XML Processing actions
- Study about Internationalization and Formatting Actions
- Learn to Deploy JAVA Beans in a JSP Page

13.2 Application – Specific Database Actions

You can use the JSTL (JSP Standard Tag Libraries) database actions to develop many types of interesting web applications, such as product catalog interfaces, employee directories, or online billboards, without being a Java programmer. These types of applications account for a high percentage of the web applications developed today. But at some level of complexity, putting SQL statements directly in the web pages can become a maintenance problem. The SQL statements represent business logic, and for more complex applications, business logic is better developed as separate Java classes. For a complex application, it may be better to use application-specific custom actions instead of the JSTL database actions. The core library contains actions for control-flow, URL manipulation, importing resources, and other general-purpose tasks.for example actions are,

JSP ACTIONS

The <c:catch> action catches an exception thrown by JSP elements in its body, providing fine-grained error control. The exception can optionally be saved as a page scope variable.

Syntax

<c:catch [var="var"]>

</c:catch>

The <c:choose> action controls the processing of nested <c:when> and <c:otherwise> actions. It allows only the first <c:when> action with a test expression that evaluates to true to be processed; it gives the go-ahead to the single <c:otherwise> action if none do.

Syntax

<c:choose>
<c:when> actions and optionally one <c:otherwise> action
</c:choose>

Example

```
<c:choose>
<c:when test="${product.onSale}">
<c:out value="${product.salesPrice}" /> On sale!
</c:when>
<c:otherwise>
<c:out value="${product.price}" />
</c:otherwise>
```

```
</c:choose>
```

The <c:forEach> action

The <c:forEach> action evaluates its body a fixed number of times or once for each element in a collection. The current element (or the current index if no collection is specified), and the iteration status can be exposed to action elements in the body through nested variables. The action accepts collections of the types listed in the Attributes table. The type of the current element is the type of the underlying collection, with two exceptions. For an array of a primitive type, the current element is exposed as an instance of the corresponding wrapper class (Integer, Float, etc.) For a java.util.Map, the current element is exposed as a java.util.Map.Entry.

The <c:forTokens> action

The <c:forTokens> action evaluates its body once for each token in a String, delimited by one of the specified delimiter characters. The current token and the iteration status can be exposed to action elements in the body through nested variables.

Syntax

```
<c:forTokens items="stringOfTokens" delims="delimiters"
[var="var"] [varStatus="varStatus"]
[begin="startIndex"] [end="stopIndex"] [step="increment"]>
JSP elements
</c:forTokens>
```

<c:if> action

The <c:if> action evaluates its body only if the specified expression evaluates to true.Alternatively, the evaluation result can be saved as a scoped Boolean variable.

Syntax 1: Without a body

```
<c:if test="booleanExpression"
var="var" [scope="page|request|session|application"]/>
```

Syntax 2: With a body

```
<c:if test="booleanExpression">
JSP elements
</c:if>
```

INTERNATIONALIZATION AND FORMATTING ACTIONS

<fmt:bundle>

The <fmt:bundle> action establishes as localization context for actions in its body. The localization context contains a locale and the best match for the specified resource bundle. The locale is either the locale defined by the locale configuration setting or the best match for the user preferences specified by the Accept-Language HTTP request header.

Syntax

```
<fmt:bundle basename="resourceBundleBasename"
[prefix="keyPrefix"]>
JSP elements
</fmt:bundle>
```

Example

```
<fmt:bundle basename="labels">
<h1><fmt:message key="title" /></h1>
</fmt:bundle>
```

DATABASE ACCESS ACTIONS

<sql:dateParam>

The <sql:dateParam> action is used as a nested action for <sql:query> and <sql:update> update to supply a date and time value for a value placeholder. If a null value is provided, the value is set to SQL NULL for the placeholder. To ensure portability between different database engines, this action must be used when setting values for DATE, TIME, and TIMESTAMP columns. The value must be of type java.util.Date or one of the SQL specific subclasses: java.sql.Date, java.sql.Time, or java.sql.Timestamp. If it's a java.util.Date, the action converts it to the specified subclass.

Syntax

```
<sql:dateParam value="parameterValue"
[type="timestamp|time|date"] />
```

Example

```
<sql:update>
UPDATE Employee SET EmpDate = ? WHERE EmpId = ?
<sql:dateParam value="${empDate}" />
<sql:param value="${empId}" />
</sql:update>
```

XML PROCESSING ACTIONS

<x:choose>

The <x:choose> action controls the processing of nested <x:when> and <x:otherwise> actions. It allows only the first <x:when> action with a test expression that evaluates to true to be processed, or gives the go-ahead to the single <x:otherwise> action if none do.

Syntax

```
<x:choose>
<x:when> actions and optionally one <x:otherwise> action
</x:choose>
```

Example

```
<x:choose>
<x:when select="category[. = 'General']">
<td bgcolor="lightgreen">
</x:when>
<x:otherwise>
<td>
</x:otherwise>
</x:choose>
```

13.3 Deploying Java Beans in JSP Page

Java Beans are reusable components. They are used to separate Business logic from the Presentation logic. Internally, a bean is just an instance of a class. JSP’s provide three basic tags for working with Beans.

- *<jsp:useBean>*
- *<jsp:setProperty>*
- *<jsp:getProperty>*

<jsp:useBean>

<jsp:useBean id="bean name" class="bean class" scope = "page | request | session |application "/>

bean name = the name that refers to the bean.
Bean class = name of the java class that defines the bean.

<jsp:setProperty>

<jsp:setProperty name = "id" property = "someProperty" value = "someValue" />

Id = the name of the bean as specified in the useBean tag.
property = name of the property to be passed to the bean.
value = value of that particular property.

An variant for this tag is the property attribute can be replaced by an " * ". What this does is that it accepts all the form parameters and thus reduces the need for writing multiple setProperty tags. The only consideration is that the form parameter names should be the same as that of the bean property names.

<jsp:getProperty>

<jsp:getProperty name = "id" property = "someProperty" />

Here the property is the name of the property whose value is to be obtained from the bean.

BEAN SCOPES :

These defines the range and lifespan of the bean. The different options are:

Page scope :

Any object whose scope is the page will disappear as soon as the current page finishes generating. The object with a page scope may be modified as often as desired within the particular page but the changes are lost as soon as the page exists. By default, all beans have page scope.

Request scope :

Any objects created in the request scope will be available as long as the request object is. For example if the JSP page uses an jsp:forward tag, then the bean should be applicable in the forwarded JSP also, if the scope defined is of Request scope.

The Session scope :

In JSP terms, the data associated with the user has session scope. A session does not correspond directly to the user; rather, it corresponds with a particular period of time the user spends at a site. Typically, this period is defined as all the visits a user makes to a site between starting and existing his browser.

Example

In this example we want to describe you a code that explain JavaBeans in JSP.For this we have to create a package my, inside this package, we define a class My Bean. The class define a newly created instance of String class.

get Name ( )

This method is used to return name of the entity i.e. class, interface, array class, void represented by the Class object as a String.

set Name ( )

This method is used to hold the value of the Class object as a String.

The next step is to create a JSP page. This JSP page uses My bean package . Let us go through the list of steps to be carried out to call a My bean class. The Jsp page is saved as UseBean.jsp,

Note :

To run Use Bean .jsp,we have to place this file inside TOMCAT HOME\webapps\Use Bean and start Tomcat. Once Tomcat is started, type the URL in the browser and run your application.

MyBean.java

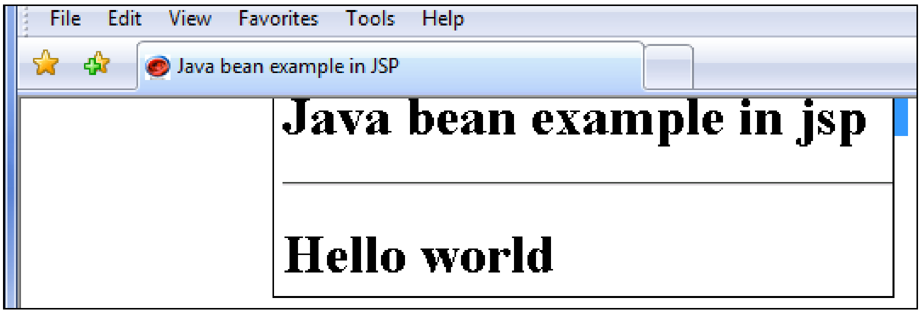
```
package my;
public class MyBean
{
         private String name=new String();
            public String getName()
           {
            return name;
        }
       public void setName(String name)
```

```
        {
            this.name = name;
         }
    }
```

UseBean.jsp

```
<html>
<title>Java bean example in jsp</title>
<head>
<h1>Java bean example in jsp</h1>
<hr></hr>
</head>
<body>
<jsp:useBean id="mybean" class="my.MyBean" scope="session" >
<jsp:setProperty name="mybean" property="name" value=" Hello world"
/>
</jsp:useBean>
<h1> <jsp:getProperty name="mybean" property="name" /></h1>
</body>
</html>
```

Output:



Java bean example in jsp

Hello world

Example 2

An example for a Database connection bean is as shown :

DbBean.Java

```
package SQLBean;
import java.sql.*;
import java.io.*;
public class DbBean
 {
```

```
 private Connection dbCon;
 public DbBean()
   {
     super();
     }
 public boolean connect() throws ClassNotFoundException,SQLException
  {
Class.forName("sun.jdbc.JdbcOdbcDriver");
dbCom = DriverManager.getConnection("jdbc:odbc:demo", "sa", "sa");
        return true;
       }
 public void close() throws SQLException
      {
       dbCon.close();
      }

 public ResultSet execSQL(String sql) throws SQLException
                {
                 Statement s = dbCon.createStatement();
                 ResultSet r = s.executeQuery(sql);
                 return (r == null) ? null : r;
                 }
 public int updateSQL(String sql) throws SQLException{
                Statement s = dbCon.createStatement();
                int r = s.executeUpdate(sql);
                return (r == 0) ? 0 : r;
              }

}
```

The description is as follows :

This bean is packaged in a folder called as "SQLBean". The name of the class file of the bean is DbBean. For this bean we have hardcoded the Database Driver and the URL. All the statements such as connecting to the database, fetching the driver etc are encapsulated in the bean.

There are two methods involved in this particular bean:

- Executing a particular query.
- Updating a database.

The execSQL(String sql) method accepts the SQL query in the form of a string from the JSP file in which this bean is implemented.

Then the createStatement() method initiates the connection with the dbCon connection object.

Further the executeQuery(sql) method executes the query which is passed on as a string.

return (r == null) ? null : r ;

What this statement does is that, if the value of r is null, it returns a null value and if it is a non null value, it returns the value of r. Though this statement seems redundant, it is useful for preventing any errors that might occur due to improper value being set in r.

```
 <HTML>
 <HEAD><TITLE>DataBase Search</TITLE></HEAD>
 <BODY>
 <%@ page language="Java" import="java.sql.*" %>
 <jsp:useBean id="db" scope="request" class="SQLBean.DbBean" />
 <jsp:setProperty name="db" property="*" />
  <%!
      ResultSet rs = null ;
      ResultSetMetaData rsmd = null ;
      int numColumns ;
      int i;
  %>
<center>
<h2> Results from </h2>
<hr>
<br><br>
<table>
<%
  db.connect();
try {
        rs = db.execSQL("select * from EMPLOYEE");
        i = db.updateSQL("UPDATE employee set FIRSTNME = 'hello
world' where EMPNO='000010'");
        out.println(i);
      }catch(SQLException e)
      {
          throw new ServletException("Your query is not working", e);
       }
        rsmd = rs.getMetaData();
        numColumns = rsmd.getColumnCount();
        for(int column=1; column <= numColumns; column++)
        {
               out.println(rsmd.getColumnName(column));
        }
%>

<%   while(rs.next())
{ %>
        <%= rs.getString("EMPNO") %>
        <BR>
        <%
 }
%>
<BR>
<%
  db.close();
%>
Done
</table>
</body>
</HTML>
```

The corresponding tags used in the JSP are as follows :

<jsp:useBean id="db" scope="request" class="SQLBean.DbBean" />

This tag specifies that the id of this bean is “db”. This id is used throughout the page to refer to this particular bean. The scope of this bean is limited to the request scope only. The class attribute points to the class of the bean.

Here the class file is stored in the SQLBean folder.

<jsp:setProperty name="db" property="" />*

This property is used for passing on all the values which are obtained from the form. In this program, the SQL query can be passed on to the program as a part of the request.getParameter so that the query can be modified according to the requests.

*rs = db.execSQL("select * from EMPLOYEE");*

We can access the execSQL() method by using the bean id. Also the SQL is passed on to this method.

i = db.updateSQL("UPDATE employee set FIRSTNME = 'hello world' where EMPNO='000010'");

The updateSQL() method can also be used in the same JSP program. Here we are updating the employee table and resetting the FIRSTNME field where the EMPNO is 000010.

The major difference between an executeQuery and executeUpdate is that, an executeQuery returns the result set and an executeUpdate returns an integer value corresponding to the number of rows updated by the current query.

13.4 Check Your Progress

1) How can you create JDBC statements?
2) What are the different types of Statements?

13.5 Answers to Check Your Progress Questions

1) A Statement object is what sends your SQL statement to the DBMS. You simply create a Statement object and then execute it, supplying the appropriate execute method with the SQL statement you want to send. For a SELECT statement, the method to use is executeQuery. For statements that create or modify tables, the method to use is executeUpdate. Eg.

It takes an instance of an active connection to create a Statement object. In the following example, we use our Connection object con to create the Statement object stmt :

```
Statement stmt = con.createStatement();
```

2) Types of statements are;

1. Statement (use createStatement method)
2. Prepared Statement (Use prepareStatement method) and
3. Callable Statement (Use prepareCall)

13.6 Summary

We have learnt about how to deploy Java Beans in a JSP Page. Java Beans are reusable components. They are used to separate Business logic from the Presentation logic. Internally, a bean is just an instance of a class. JSP's provide three basic tags for working with Beans.

- *<jsp:useBean>*
- *<jsp:setProperty>*

- *<jsp:getProperty>*

13.7 Key Words

Prepared Statement object contains not just an SQL Statement, but an SQL Statement that has been pre-compiled

A **stored procedure** is a set of SQL Statements that form a logical unit and perform a particular task.

13.8 Self-Assessment Questions and Exercises

1. Explain about Bean Scopus.
2. Write the difference between getname() and setname() property.
3. Write program using Javabeans.

13.9 Further Readings

1. Thomas A. Powell, HTML: the complete reference, McGraw-Hill, 2001.
2. Chris Bates, Web Programming, Building Internet Applications, 2nd Edition, Dreamtech
3. Patrick Naughton and Herbert Schildt, The complete Reference Java 2, 5th Edition, Tata McGraw Hill.
4. Jason Hunter Java Servlet Programming, O'Reilly
5. Hans Bergsten, Java Server Pages, O'Reilly
6. Ramesh Bangia,Web Technology, Firewall media, 2006.

NOTES

UNIT 14

Introduction to struts framework.

Structure

14. 0 Introduction

Struts is an open source framework used for developing J2EE web applications using Model View Controller (MVC) design pattern. It uses and extends the Java Servlet API to encourage developers to adopt an MVC architecture.

14. 1 Objectives

After going through the unit you will be able to;

- Learn Introduction to struts framework.
- Understand the strut controller

14.2 Introduction to Strut Framework

Struts is an open source framework used for developing J2EE web applications using Model View Controller (MVC) design pattern. It uses and extends the Java Servlet API to encourage developers to adopt an MVC architecture. Struts framework provides three key components:

- A request handler provided by the application developer that is used to mapped to a particular URI.

- A response handler which is used to transfer the control to another resource which will be responsible for completing the response.

NOTES

- A tag library which helps developers to create the interactive form based applications with server pages.

Struts provides you the basic infrastructure infrastructure for implementing MVC allowing the developers to concentrate on the business logic.

MVC Architecture

The main aim of the MVC architecture is to separate the business logic and application data from the presentation data to the user.

Here are the reasons why we should use the MVC design pattern.

They are resuable : When the problems recurs, there is no need to invent a new solution, we just have to follow the pattern and adapt it as necessary. They are expressive: By using the MVC design pattern our application becomes more expressive.

1) Model

The model object knows about all the data that need to be displayed. It is model who is aware about all the operations that can be applied to transform that object. It only represents the data of an application. The model represents enterprise data and the business rules that govern access to and updates of this data. Model is not aware about the presentation data and how that data will be displayed to the browser.

2) View

The view represents the presentation of the application. The view object refers to the model. It uses the query methods of the model to obtain the contents and renders it. The view is not dependent on the application logic. It remains same if there is any modification in the business logic. In other words, we can say that it is the responsibility of the of the view's to maintain the consistency in its presentation when the model changes.

3) Controller

Whenever the user sends a request for something then it always go through the controller. The controller is responsible for intercepting the requests from view and passes it to the model for the appropriate action. After the action has been taken on the data, the controller is responsible for directing the appropriate view to the user. In GUIs, the views and the controllers often work very closely together.

Overview of the Struts Framework

The Struts framework is composed of approximately 300 classes and interfaces which are organized in about 12 top level packages. Along with the utility and helper classes framework also provides the classes and interfaces for working with controller and presentation by the help of the custom tag libraries. It is entirely on to us which model we want to choose.

The Struts Controller Components:

Whenever a user request for something, then the request is handled by the Struts Action Servlet. When the ActionServlet receives the request, it intercepts the URL and based on the Struts Configuration files, it gives the handling of the request to the Action class. Action class is a part of the controller and is responsible for communicating with the model layer.

The Struts View Components:

The view components are responsible for presenting information to the users and accepting the input from them. They are responsible for displaying the information provided by the model components. Mostly we use the Java Server Pages (JSP) for the view presentation. To extend the capability of the view we can use the Custom tags, java script etc.

The Struts model component:

The model components provides a model of the business logic behind a Struts program. It provides interfaces to databases or back- ends systems. Model components are generally a java class. There is not any such defined format for a Model component, so it is possible for us to reuse Java code which are written for other projects. We should choose the model according to our client requirement.

Struts Process flow

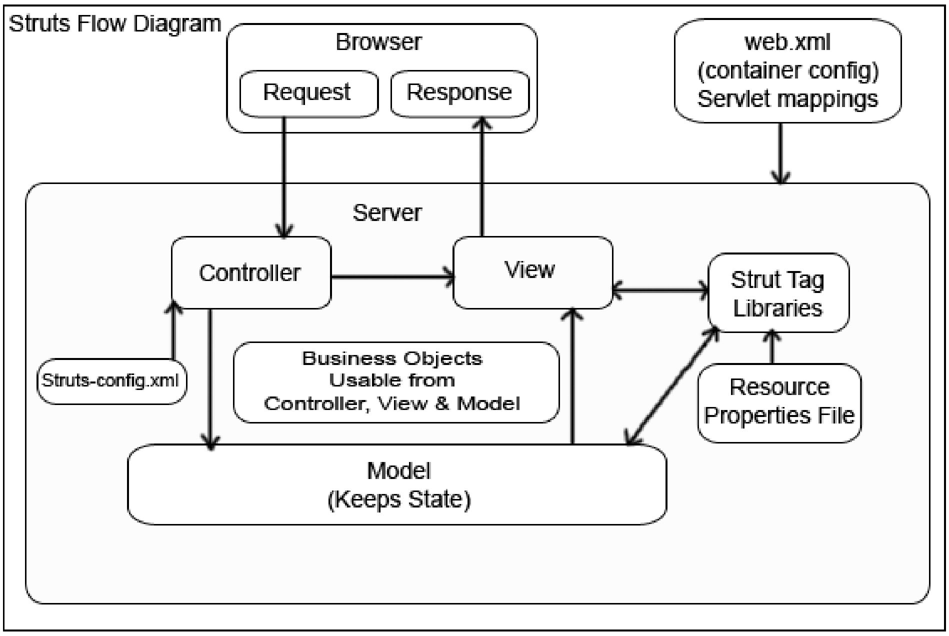
web.xml : Whenever the container gets start up the first work it does is to check the web.xml file and determine what struts action Servlets exist. The container is responsible for mapping all the file request to the correct action Servlet.

A Request : This is the second step performed by the container after checking the web.xml file. In this the user submits a form within a browser and the request is intercepted by the controller.

The Controller : This is the heart of the container. Most Struts application will have only one controller that is ActionServlet which is responsible for directing several Actions. The controller determines what action is required and sends the information to be processed by an action Bean. The key advantage of having a controller is its ability to control the flow of logic through the highly controlled, centralized points.

struts.config.xml : Struts has a configuration file to store mappings of actions. By using this file there is no need to hard code the module which will be called within a component. The one more responsibility of the controller is to check the struts.config.xml file to determine which module to be called upon an action request. Struts only reads the struts.config.xml file upon start up.

Model : The model is basically a business logic part which takes the response from the user and stores the result for the duration of the process. This is a great place to perform the preprocessing of the data received from request. It is possible to reuse the same model for many page requests. Struts provides the ActionForm and the Action classes which can be extended to create the model objects.



View : The view in struts framework is mainly a jsp page which is responsible for producing the output to the user.

Struts tag libraries : These are struts components helps us to integrate the struts framework within the project's logic. These struts tag libraries are used within the JSP page. This means that the controller and the model part can't make use of the tag library but instead use the struts class library for strut process control.

Property file : It is used to store the messages that an object or page can use. Properties files can be used to store the titles and other string data. We can create many property files to handle different languages.

Business objects : It is the place where the rules of the actual project exists. These are the modules which just regulate the day- to- day site activities.

The Response : This is the output of the View JSP object.

Understanding Struts Controller

This section we will describe the Controller part of the Struts Framework and to configure the struts-config.xml file to map the request to some destination servlet or jsp file.

The class org.apache.struts.action.ActionServlet is the heart of the Struts Framework. It is the Controller part of the Struts Framework. ActionServlet is configured as Servlet in the web.xml file as shown in the following code snippets.

```
<!-- Standard Action Servlet Configuration (with debugging) -->
<servlet>
    <servlet-name>action</servlet-name>
     <servlet-class>org.apache.struts.action.ActionServlet</servlet-class>
     <init-param>
       <param-name>config</param-name>
       <param-value>/WEB-INF/struts-config.xml</param-value>
     </init-param>
    <init-param>
       <param-name>debug</param-name>
       <param-value>2</param-value>
       </init-param>
       <init-param>
       <param-name>detail</param-name>
       <param-value>2</param-value>
    </init-param>
   <load-on-startup>2</load-on-startup>
</servlet>
```

This servlet is responsible for handing all the request for the Struts Framework, user can map the specific pattern of request to the

ActionServlet. <servlet-mapping> tag in the web.xml file specifies the url pattern to be handled by the servlet. By default it is *.do, but it can be changed to anything. Following code form the ***web.xml*** file shows the mapping.

```
<!-- Standard Action Servlet Mapping -->
<servlet-mapping>
   <servlet-name>action</servlet-name>
   <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

The above mapping maps all the requests ending with .do to the ActionServlet. ActionServlet uses the configuration defined in struts-config.xml file to decide the destination of the request. Action Mapping Definitions (described below) is used to map any action. For this lesson we will create Welcome.jsp file and map the "Welcome.do" request to this page.

Welcome.jsp

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<html:html locale="true">
<head>
  <title><bean:message key="welcome.title"/></title>
  <html:base/>
</head>
 <body bgcolor="white">
 <h3><bean:message key="welcome.heading"/></h3>
 <p><bean:message key="welcome.message"/></p>
</body>
</html:html>
```

Forwarding the Welcome.do request to Welcome.jsp

The "Action Mapping Definitions" is the most important part in the struts-config.xml. This section takes a form defined in the "Form Bean Definitions" section and maps it to an action class.

Following code under the <action-mappings> tag is used to forward the request to the Welcome.jsp.

```
<action  path="/Welcome"
      forward="/pages/Welcome.jsp"/>
```

To call this Welcome.jsp file we will use the following code.

<html:link page="/Welcome.do">First Request to the controller</html:link>

Once the use clicks on on First Request to the controller link on the index page, request (for Welcome.do) is sent to the Controller and the controller forwards the request to Welcome.jsp. The content of Welcome.jsp is displayed to the user.

Action Class

An Action class in the struts application extends Struts 'org.apache.struts.action.Action" Class. Action class acts as wrapper around the business logic and provides an interface to the application's Model layer. It acts as glue between the View and Model layer. It also transfers the data from the view layer to the specific business process layer and finally returns the processed data from business layer to the view layer.

An Action works as an adapter between the contents of an incoming HTTP request and the business logic that corresponds to it. Then the struts controller (ActionServlet) selects an appropriate Action and creates an instance if necessary, and finally calls execute method.

To use the Action, we need to Subclass and overwrite the execute() method. In the Action Class don't add the business process logic, instead move the database and business process logic to the process or dao layer.

The ActionServlet (commad) passes the parameterized class to Action Form using the execute() method. The return type of the execute method is ActionForward which is used by the Struts Framework to forward the request to the file as per the value of the returned ActionForward object.

Developing our Action Class:

Our Action class (TestAction.java) is simple class that only forwards the TestAction.jsp. Our Action class returns the ActionForward called "testAction", which is defined in the struts-config.xml file (action mapping is show later in this page). Here is code of our Action Class:

TestAction.java

```
package alagappa.net;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.ActionMapping;

public class TestAction extends Action
{
  public ActionForward execute(
    ActionMapping mapping,
    ActionForm form,
    HttpServletRequest request,
    HttpServletResponse response) throws Exception{
      return mapping.findForward("testAction");
  }
}
```

Understanding Action Class

Here is the signature of the Action Class.

```
public ActionForward execute(ActionMapping mapping,
                             ActionForm form,
                             javax.servlet.http.HttpServletRequest request,
                             javax.servlet.http.HttpServletResponse response)
                              throws java.lang.Exception
```

Action Class process the specified HTTP request, and create the corresponding HTTP response (or forward to another web component that will create it), with provision for handling exceptions thrown by the business logic. Return an ActionForward instance describing where and how control should be forwarded, or null if the response has already been completed.

Parameters:

mapping - *The ActionMapping used to select this instance*
form - *The optional ActionForm bean for this request (if any)*
request - *The HTTP request we are processing*
response - *The HTTP response we are creating*

Throws:

Action class throws java.lang.Exception - if the application business logic throws an exception

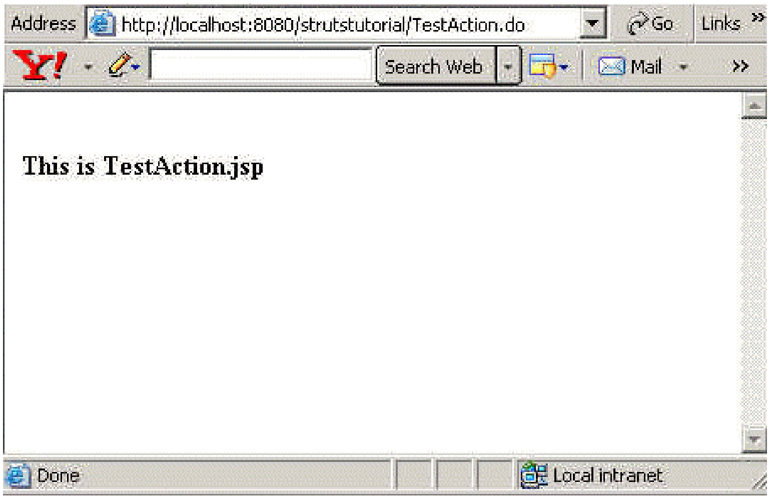
Adding the Action Mapping in the *struts-config.xml*
To test the application we will add a link in the index.jsp

```
<html:link page="/TestAction.do">Test the Action</html:link>
```

Following code under the <action-mappings> tag is used to for mapping the TestAction class.

```
<action    path="/TestAction"    type="alagappa.net.TestAction">
  <forward name="testAction" path="/pages/TestAction.jsp"/>
</action>
```

To test the new application click on Test the Action link on the index page. The content of TestAction.jsp should be displayed on the user browser.



This is TestAction.jsp

14.3 Check Your Progress

1. Write True or False for the following :

(a) Struts is an open source framework.
(b) The Mapping action is used to select this instance.

14.4 Answers to Check Your Progress Questions

1. (a) True (b) True.

14.5 Summary

In this unit we have learnt about Struct Framework and also MVC Architecture of Database. The Struts model components provides a model of the business logic behind a Struts program. It provides interfaces to databases or back- ends systems.

14.6 Key Words

Struts is an open source framework used for developing J2EE web applications using Model View Controller (MVC) design pattern. It uses and extends the Java Servlet API to encourage developers to adopt an MVC architecture.

14.7 Self-Assessment Questions and Exercises

1. What are the Struts Process Flows?
2. Write short note on : MVC architecture.

14.8 Further Readings

1. Chris Bates, Web Programming, Building Internet Applications, 2nd Edition, Dreamtech
2. Patrick Naughton and Herbert Schildt, The complete Reference Java 2, 5th Edition, Tata McGraw Hill.
3. Ramesh Bangia, Web Technology, Firewall media, 2006.

Sub.Code 31341

MODEL QUESTION PAPER

DISTANCE EDUCATION
M.Sc (INFORMATION TECHNOLOGY) DEGREE EXAMINATION
WEB TECHNOLOGY

Second Year - Fourth Semester
(CBCS – 2018-19 Academic Year Onwards)

Time : 3 hours Max Marks :75

PART - A (10 x 2=20 Marks)
Answer all questions.

1. write down HTML tags to explain frame within a frame.
2. What are the various styles in CSS?
3. How to declare variables in Javascript?
4. Difference between javabean and bean.
5. Define Document Object Module (DOM).
6. What is meant by Jave Servlet Deveklopment Kit (JSDK)?
7. Differentiate cookies from Session variables.
8. Explain the various steps of JSP Compilation.
9. Explain the general form of an URL.
10. Difference between JDBC and ODBC.

PART - B (5 x 5 Marks = 25 Marks)
Answer all questions choosing either (a) or (b)

11. (a)Discuss about various built-in Objects in javascript.
(Or)
(b)Explain about various types of List tags in HTML.

12. (a)Briefly explain the types of Enterprise javabeans.
(Or)
(b) Write short note on : Bean Persistence

13. (a)Describe servlet life cycle with neat diagram.
(Or)
(b) Write short note on: Servlet API

14. (a) Write a JSP code for pass data from one page to another.
(Or)
(b) Explain about various types of JDBC Drivers.

15. (a) Explain about MVC Architecture.
(Or)
(b) Write short note on: Database actions in JSP

Part – C (3 x 10 = 30 Marks)
Answer any three questions.

16. What is CSS and briefly explain about CSS and DTD?
17. Explain about various events handling mechanism in Javascript.
18. How will you download and install tomcat server? Explain
19. Design a JSP Page code to perform shopping cart website using JDBC connectivity.
20. Discuss in detail about the framework of Struts.

****