10. FORMS USED BY A WEB SITE

An HTML form provides data gathering functionality to a web page. This is very useful if the web site is used to advertise and sell products. HTML forms provide a full range of GUI controls. Additionally, HTML forms can automatically submit data collected in its controls to a web server.

The data submitted can be processed at the web server by CGI programs, server side JavaScripts, Java Servlets and so on.

JavaScript allows the validation of data entered into a form at the client side. JavaScript can be used to ensure that only valid data is returned to a web server for further processing.

This chapter focuses on:

- The JavaScript FORM object created when the HTML <FORM> </FORM> tags are encountered in an HTML program.
- Describes how JavaScript can be associated with an HTML form's GUI controls.

After working through chapter examples the following will be understood:

- The properties and methods of the JavaScript FORM object and it's associated GUI controls.
- How JavaScript is used to handle form related events and perform (*client side - in the browser window*) local processing of form data.

THE FORM OBJECT

When creating an interactive web site for the Internet it is necessary to capture user input and process this input. Based on the result of this processing, appropriate information from a web site can be dispatched to be viewed. Both the capturing of user input and the rendering of appropriate web pages takes place in the client side, browser's window.

Traditionally, user input is captured in a **Form**. HTML provides the **<FORM> ... </FORM>** tags with which an HTML form can be created to capture user input.

As soon as the **<FORM> ... </FORM>** tags are encountered in an HTML program by a JavaScript enabled browser, the browser creates a '*forms array*' in memory. This array tracks the number of form objects described in the HTML program.

Each form object in the HTML page will be described between its own <FORM> ... </FORM> HTML tags. Should there be multiple forms (i.e. multiple occurrences of the <FORM> ... </FORM> tags) described in the HTML page then the forms array will have multiple (*indexed*) elements, each holding a reference to an HTML form object.

The first form object described in the HTML file being held as array index[0], the second form object described in the HTML file being held in the array index[1] and so on. By referencing a specific index number of the forms array a specific form object can be accessed. The JavaScript forms array also holds information about each object used within the <FORM> ... </FORM> tags.

Common HTML objects used between the <FORM> ... </FORM> tags are Text, TextArea, Radio Buttons, Buttons, Check Boxes and so on. An HTML form is used extensively in creating interactive web pages. Using the associated arrays created by a JavaScript enabled browser and JavaScript code, all form elements (*objects contained in the form*) are accessible. Once accessible their properties can be manipulated so as to control the functionality of the form at run time.

There are two forms in the HTML file; **Form1** and **Form2**. Refer to diagram 10.1. These will be held in the first two elements of the Forms array. **Form1** will be an address, which points to where the Form elements array is located.

Form1 in the Forms array, will be a reference to the context area where the elements of Form1 are located. Form1 has three form elements.

Form2 in the Forms array, is a reference to the context area where the elements of Form2 are located. Form2 has five form elements.

To understand this model, let us write a JavaScript procedure that will read the elements of the Form object's array, and return the number of actual form objects held.

Once the reference to the form's elements are known let us write a JavaScript procedure that will read the each Form's Element array and return the names of the form elements held in the array. The elements held in each of the arrays must be exactly the same as the elements described between the **<FORM>...</FORM>** tags in the HTML file running in the browser.

Forms Object's (Array)

Index	Value
0	Form1
1	Form2

Form1.Elements

Index	Value
0	Form1.Element1
1	Form1.Element2
2	Form1.Element3

Form2.Elements

Index	Value
0	Form2.Element1
1	Form2.Element2
2	Form2.Element3
3	Form2.Element4
4	Form2.Element5

Diagram 10.1: The Form Element's Array

Exercise 1:

Focus: Count the number of elements in a Form's, elements array. Check the number returned against the number of form elements described between the **<FORM>...</FORM>** tags in the HTML page that is running in the Browser. Recognize that the number of elements in the elements array match the number of elements described between the <FORM>...</FORM> tags in the HTML page exactly.

The diagram 10.2.1 shows an Alert box that displays a message indicating the number of form elements of implemented by the First Form in the HTML file. Refer to the HTML and JavaScript code described in Exercise 1.

Pop up an Alert that displays the individual form elements of the array and recognize that these are the same as are specified between the <FORM>...</FORM> tags in the HTML program

The diagram 10.2.2 shows an Alert box displaying a message that the Textbox named **Text1** of the First Form is at position **0** in the Elements array.

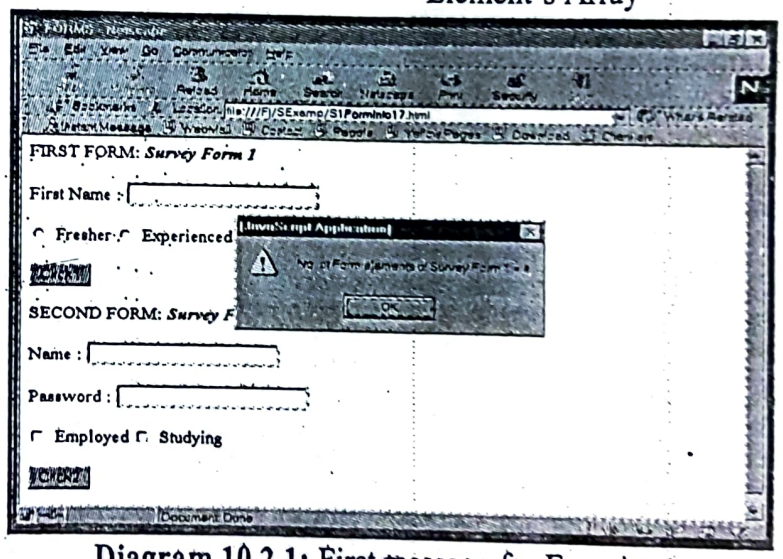

Diagram 10.2.1: First message for Exercise 1.

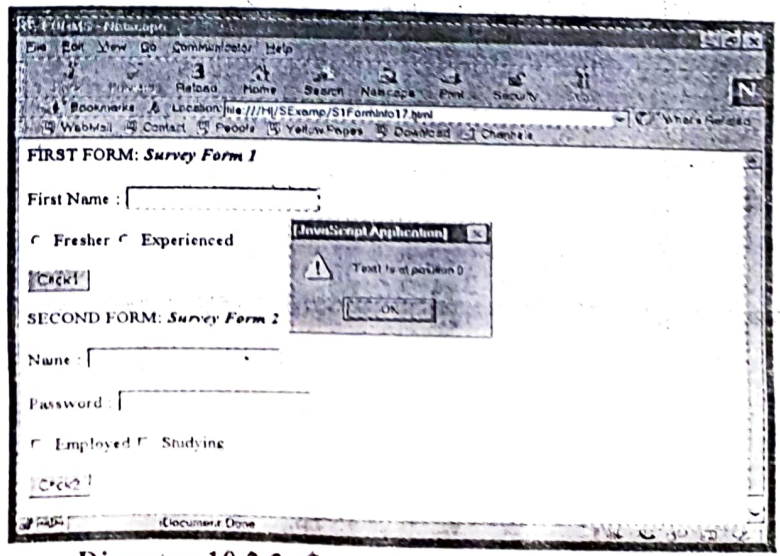

Diagram 10.2.2: Second message for Exercise 1.

The Code Listing For Exercise 1:

```
<HMTL>
   <HEAD><TITLE>FORMS</TITLE>
<!-- The code allows to access the Form objects Elements Array //-->
   <SCRIPT Language="JavaScript">
      function Ver(form1) {
         v = form1.elements.length;
         if (form1.elements[3].name == "Button1") {
            alert('First form name : ' + document.forms[0].name);
            alert('No. of Form elements of ' + document.forms[0].name + ' = ' + v);
         }
         else if (form1.elements[4].name == "Button2") {
            alert('Second form name : ' + document.forms[1].name);
            alert('No. of Form elements of ' + document.forms[1].name + ' = ' + v);
         }
         for(i=0; i < v; i++)
            alert(form1.elements[i].name + ' is at position ' + i);
      }
   </SCRIPT></HEAD>
   <BODY>
      <FORM Name="Survey Form 1">
         FIRST FORM: <I><B>Survey Form 1 </B></I><BR/><BR/>
         First Name : <INPUT Name="Text1" Type="Text" Value="" /><BR/><BR/>
         <INPUT Name="Radio1" Type="Radio" Value="" /> Fresher
         <INPUT Name="Radio2" Type="Radio" Value="" /> Experienced<BR/><BR/>
         <INPUT Name="Button1" onClick="Ver(form)" Type="Button" Value="Click1" />
      </FORM>
      <FORM Name="Survey Form 2">
         SECOND FORM: <I><B> Survey Form 2 </B></I><BR/><BR/>
         Name : <INPUT Name="Text2" Type="Text" Value="" /> <BR/><BR/>
         Password : <INPUT Name="Pass2" Type="Password" Value="" /> <BR/><BR/>
         <INPUT Name="Check1" Type="CheckBox" Value="" /> Employed
         <INPUT Name="Check2" Type="CheckBox" Value="" /> Studying <BR/><BR/>
         <INPUT Name="Button2" onClick="Ver(form)" Type="Button" Value="Click2" />
      </FORM>
   </BODY>
</HTML>
```

Exercise 2: Illustrates the use of a form object's Elements array

Focus: The state of a Radio button and a Checkbox on the HTML form can be programmatically changed using event based JavaScript, *i.e. on the clicked event of a command button.*

When the JavaScript program runs an Alert draws attention to the fact that the Checkbox, **checked** property has been set to true via JavaScript code.

The diagram 10.3.1 shows an Alert box that displays a message indicating that the Checkbox is checked, on clicking the `Set Element Array Value` button.

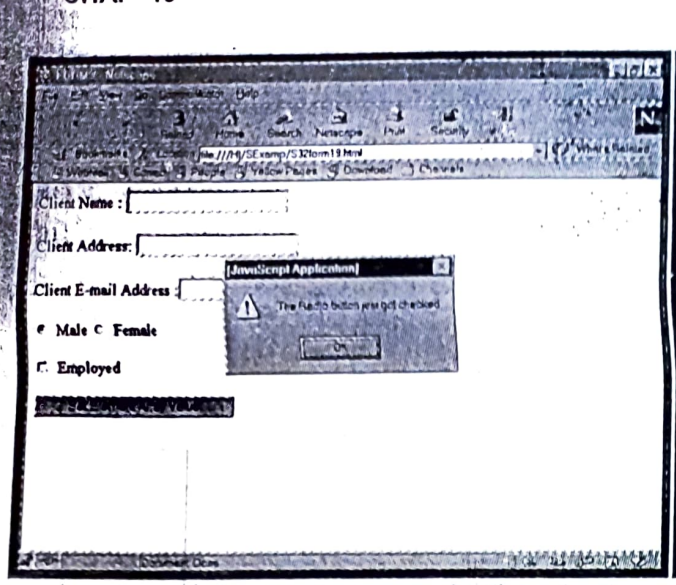

Diagram 10.3.1: First message for Exercise 2.

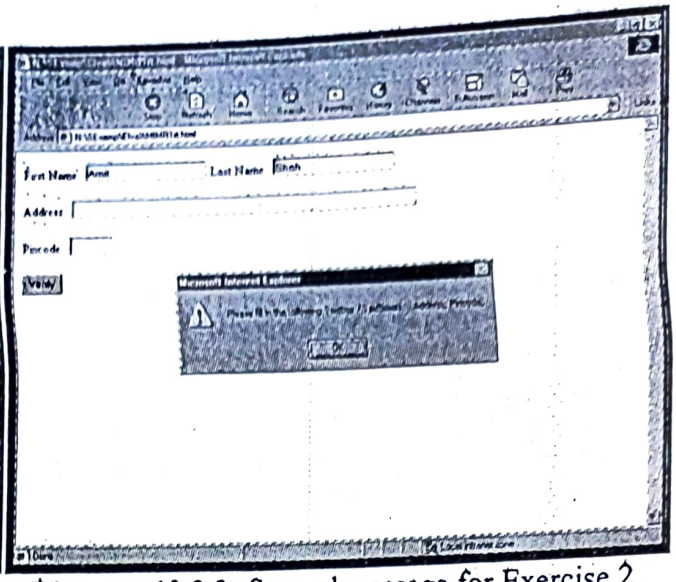

Diagram 10.3.2: Second message for Exercise 2.

On clicking the first Alert's OK button, another Alert pops up that displays a message indicating that the Radio button is checked, when the same 'Set Element Array Value' button was clicked.

Conceptually, without clicking on the HTML form objects their 'Checked' properties have been set. This illustrates how JavaScript code can access a form's element via the form elements array and manipulate an element's properties.

As soon as the elements properties change the Browser will display the object with its changed property in the HTML page. Refer to the HTML and JavaScript code described in Exercise 2.

The Code Listing For Exercise 2:

```
<HMTL>
   <HEAD><TITLE>FORMS</TITLE>
<!-- The code allows to access the Form objects Elements Array //-->
   <SCRIPT Language='JavaScript'>
      function Chk(f1) {
         f1.Check.checked=true;
         alert("The Checkbox just got checked");
         f1.Check.checked=false;
         f1.Radio[0].checked=true;
         f1.Radio[1].checked=false;
         alert("The Radio button just got checked");
      }
  </SCRIPT></HEAD>
  <BODY><FORM>
      Client Name : <INPUT Name="Text" Type="Text" Value="" /><BR/><BR/>
      Client Address : <INPUT Name="Text1" Type="Text" Value="" /><BR/><BR/>
      Client E-mail Address : <INPUT Name="Text2" Type="Text" Value="" /><BR/><BR/>
      <INPUT Name="Radio" Type="radio" Value="" /> Male
      <INPUT Name="Radio" Type="radio" Value="" /> Female<BR/><BR/>
      <INPUT Name="Check" Type="CheckBox" Value="" /> Employed <BR/><BR/>
      <INPUT Name="Bt" onClick="Chk(this.form)" Type="Button"
            Value="Set Element Array Value" >
  </FORM></BODY>
</HTML>
```

Exercise 3:

Focus: Create a HTML form that has a number of Textboxes. When the form runs in the Browser fill the Textboxes with data. Write JavaScript code that verifies that all Textboxes have been filled. If a Textbox has been left empty, popup an Alert indicating which Textbox has been left empty. When the Alert's OK button is clicked on, Set focus to that specific Textbox. If all the Textboxes are filled, display a Thank You alert.

The diagram 10.4.1 shows an Alert box that displays a message indicating that the **Address** and the **Pincode** Textboxes were not filled in.

The diagram 10.4.2 shows an Alert box that displays a message indicating that all the Textboxes have been filled.

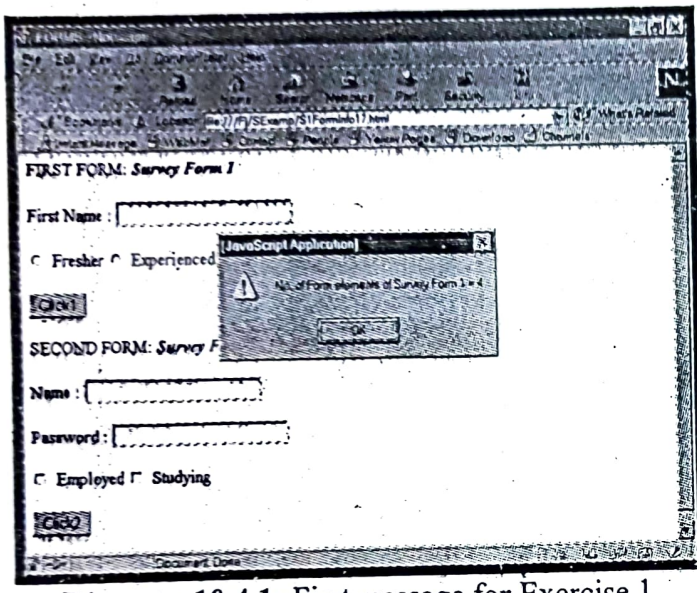

Diagram 10.4.1: First message for Exercise 1.

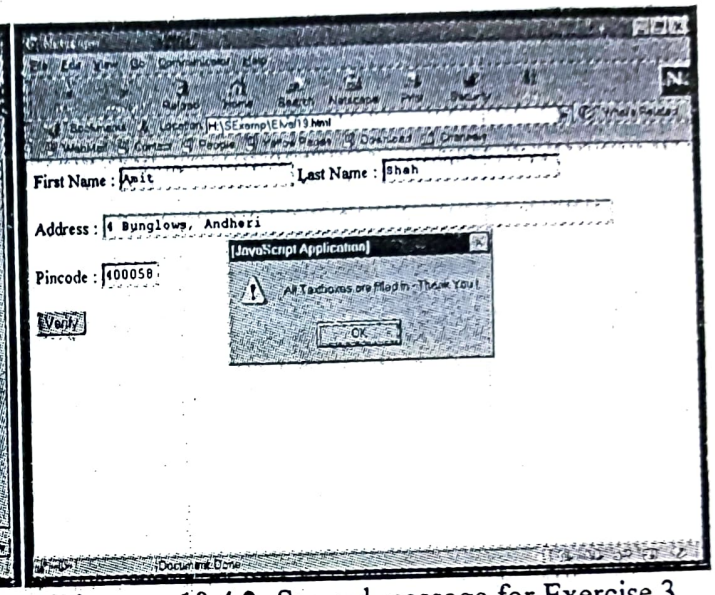

Diagram 10.4.2: Second message for Exercise 3.

This is a simple exercise, which illustrates how JavaScript code can be used to validate data that is entered in a Form.

Based on this concept, business rules can be implemented in any HTML form using JavaScript.

Thus data being entered into an HTML form can be validated on the client's machine before being dispatched to a web server for further processing. Refer to the HTML and JavaScript code described in Exercise 3.

The code listing for Exercise 3:

```
<HTML>
    <HEAD><SCRIPT LANGUAGE="JAVASCRIPT">
        function verifyData() {
            a=0;      r="";
            for (i=0; i<=4; i++) {
                if (document.forms[0].elements[i].value == "") {
                    a=1;
                    r = r + " " + document.forms[0].elements[i].name + ";" ;
                }
                else if ((i > 3)&&(a==0)) {
                    alert("All Textboxes are filled in - Thank You !");
                }
```

```
            }
            for (i=0; i<=4; i++) {
                if (document.forms[0].elements[i].value == "") {
                    alert("Please fill in the following Textbox / Textboxes :- " + r);
                    document.forms[0].elements[i].focus( );
                    break;
                }
            }
        }
    </SCRIPT></HEAD>
    <BODY><FORM>
        First Name : <INPUT Name="Firstname" Type="Text" Size=20 />
        Last Name : <INPUT Name="Lastname" Type="Text" Size=20 />
        <P>Address : <INPUT Name="Address" Type="Text" Size=60 /></P>
        <P>Pincode : <INPUT Name="Pincode" Type="Text" Size=6 /></P>
        <INPUT Name="act" onClick="verifyData( )" Type="button"Value="Verify" />
    </FORM></BODY>
    <SCRIPT Language="JavaScript">
        document.forms[0].Firstname.focus( );
    </SCRIPT>
</HTML>
```

The Form Object's Methods

HTML forms can be made up of a variety of HTML elements that accept user input. The <FORM> </FORM> HTML tags enclose the HTML elements that make up the form. Once a JavaScript enabled browser encounters these tags in an HTML file the JavaScript enabled browser creates a **form object** in memory, which is held as an element of the forms array. The form object has properties like **Name, Method** and **Action**.

Method

The **Method** property of a form is used to specify the method used to send data captured by various form elements back to the web server. The method used can be either **Get** or **Post**.

The **Get** method sends the data captured by form elements to the web server encoded into a URL, which points to a web server. The data captured in form elements is appended to the URL.

This technique is used when there is a small amount of data being sent back to the web server. The maximum amount of data that can be sent back to the web server using this method is 1024 bytes.

The **Post** method sends the data captured by form elements back to the web server as a separate bit-stream of data. When there is a large amount of data to be sent back to the web server, this is the method used.

If the method attribute is not specified within the <FORM> </FORM> tags, the default method used by the browser to send data back to the web server is the *Get* method, *i.e. as an encoded URL.*

Action

The Action attribute of the <FORM>...</FORM> tags points to the URL (*address*) of a program on the web server that will process the form data captured and being sent back. The server side program that processes this data can be written in any scripting language that the web server understands.

Commonly used web server side scripting languages are:

- JavaScript - (*used with the Netscape suite of web servers*)
- VB Script and ASP - (*used with the Microsoft suite of web servers*)
- Perl, ANSI C - (*used with the Unix based suite of web servers*) and so on.

HTML elements used to capture form data are specified as attributes of the <INPUT>...</INPUT> tags used within the <FORM>...</FORM> tags.

The HTML form elements that can be specified as attributes to the <INPUT> tag are:

Form Elements	Description & Syntax
Text	A text field (<INPUT Type="**Text**">)
Password	A password text field in which each keystroke appears as an asterisk(*) (<INPUT Type="**Password**">)
Button	A new element that provides a button other than a submit or reset button (<INPUT Type="**Button**">)
Checkbox	A checkbox (<INPUT Type="**Checkbox**">)
Radio	A radio button (<INPUT Type="**Radio**">)
Reset	A reset button (<INPUT Type="**Reset**">)
Submit	A submit button (<INPUT Type="**Submit**">)
Select	A selection list (<SELECT><OPTION>option1</OPTION> <OPTION>option2</OPTION></SELECT>)
TextArea	A multi line text entry field (<TEXTAREA>Default Text</TEXTAREA>)
Hidden	A field that may contain a value but is not displayed within a form (<INPUT Type="**Hidden**">)

Table 10.1

Each of these form elements can be named. Once named, their names can then be used for referencing them in JavaScript. 'Name' is a property associated with every HTML object used in a form.

There are several other Properties and Methods associated with each of these form objects. These Properties and Methods, along with the objects with which they are associated is summarized as follows:

Properties Of Form Elements

Form Element Name	Property Name	Description
Text Password TextArea Button Radio CheckBox Select Submit Reset Hidden Fileupload	Name	Indicates the name of the object. This name can be used for referencing the object in future, when required.

Table 10.2: Properties Of Form Elements

Form Element Name	Property Name	Description
Text	Value	Indicates the current value of the element.
Password		
TextArea		
Button		
Radio		
CheckBox		
Submit		
Reset		
Hidden		
Fileupload		
Select		It contains any value indicated in the option tag.
Text	DefaultValue	Indicates the default value of the object.
Password		
TextArea		
Radio Button	Checked	Indicates the current status of the object, whether checked or unchecked.
Check Box		
Radio Button	DefaultChecked	Indicates the default status of the element.
Check Box		
Radio	Length	Indicates the number of radio buttons in a group.
Radio Button	Index	Indicates the index value of the currently selected radio button.
Select		Contains the index value in the current option of the options array.
	Text	Contains the value of the text displayed in the menu for the specific option
	SelectedIndex	Contains the index number of the currently selected option.
	DefaultSelected	Indicates whether the option is selected by default in the option tag.
	Selected	Indicates the current status of the option.

Table 10.2: Properties Of Form Elements (Continued)

Methods of Form Elements

Form Element Name	Method Name	Description
Text	onFocus()	Fires when the form cursor enters into an object.
Password		
TextArea		
Text	onBlur()	Fires when the form cursor is moved away from an object.
Password		
TextArea		
Text	onSelect()	Fires when text is selected in an object.
Password		
TextArea		
Text	onChange()	Fires when text is changed in an object.
Password		
TextArea		
Button	onClick()	Fires when an object is clicked on.
Radio		
Checkbox		
Submit		
Reset		

Table 10.3: Methods Of Form Elements

The Text Element

Text elements are data entry fields used in HTML forms. Text fields accept a single line of text entry.

Properties

The text object has the following properties:

- name
- value

Methods

The text object has the following methods:

- focus()
- blur()
- select()
  (Selects the text in the data entry field, i.e. causes the text to be highlighted).

Events

- Focus()
- Blur()
- Select()
- Change()

JavaScript provides the following event handlers for the text object's events:

- onFocus()
- onBlur()
- onSelect()
- onChange()

Syntax:

<INPUT Name="`<NameOfTheObject>`**" Type="Text" Value="**`<DefaultValue>`**">**

Example:

```
<INPUT Name="txt_age" Type="Text" Value="18">
```

This places a **Text** field (i.e. a single line text edit area) within an HTML form, which can be referenced by the name **txt_age**. The text field will immediately display the value **18**.

The Password Element

The password element is a unique type of text entry field. All keystrokes for this field are displayed as an asterisk [*]. This makes the password element ideal for accepting input of confidential information, such as a password, bank account number or a personal identification number.

Properties

The password object has the following properties:

- defaultValue
- name
- value

Methods

The password object has the following methods:

- focus()
- blur()
- select()

(Selects text in the password element, i.e, causes selected text to be highlighted).

Events

The password object has the following methods:

- Focus()
- Blur()
- Select()
- Change()

JavaScript provides the following event handlers for the password object's events:

- onFocus()
- onBlur()
- onSelect()
- onChange()

Syntax:

```
<INPUT Name="<NameOfTheObject>" Type="Password" Value="<DefaultValue>">
```

Example:

```
<INPUT Name="txt_usr_pswd" Type="Password" Value="">
```

This places a **Password** field within an HTML form, which can be referenced by the name **txt_user_name**.

The Button Element

An HTML button element is a commonly used form object. It is generally used to trigger appropriate form level processing.

Properties

- name
- value

Method

- click()

Event

- click()

JavaScript provides the following event handler for the button object's event:

- onClick().

Syntax:

```
<INPUT Name="<NameOfTheObject>" Type="Button" Value="<ButtonLabel>">
```

Example:

```
<INPUT Name="btn_check" Type="Button" Value="Verify...">
```

This places a **Button** on the HTML form named **btn_check**. The button will display the text **Verify...** on its face as a label.

Exercise 4:

Focus: Develop a HTML Page, which accepts:

- Any mathematical expression
- Evaluates the expression
- Displays the result of the evaluation

To achieve this a form is created which has two **Text** objects and one **Button** object as shown in diagram 10.5. The first **Text** object is used to accept a mathematical expression for evaluation. When the **Button** object (Calculate) is clicked on, the second **Text** object will display the output of the evaluation of the expression entered into the first **Text** object. Refer to the HTML and JavaScript code described in Exercise 4.

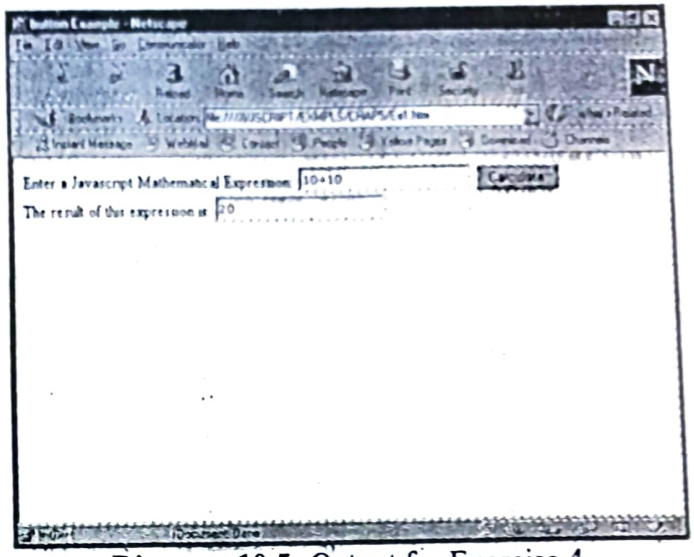

Diagram 10.5: Output for Exercise 4.

The Code listing for Exercise 4:

```
<HTML>
    <HEAD><TITLE>Using Text and Button objects in an HTML Form</TITLE>
    <SCRIPT Language="JavaScript">
        function calculate(form) {
            form.results.value = eval(form.entry.value);
        }
    </SCRIPT></HEAD>
    <BODY><FORM>Enter a Javascript Mathematical Expression:
        <INPUT Type="Text" Name="entry" Value="" />
        <INPUT Type="button" Value="Calculate" onClick="calculate(this.form);" /><BR/>
        The result of this expression is:
        <INPUT Type="Text" Name="results" onFocus="this.blur();" />
    </FORM></BODY>
</HTML>
```

When this program is executed in a JavaScript enabled browser, the used defined JavaScript function **calculate()** is registered by the browser first as it is written between the **<HEAD>... </HEAD>** tags.

The lines of code between the <FORM> </FORM> tags creates a form as seen in diagram 10.5. Enter an expression in the first 'Text' object on the form for evaluation and with the mouse cursor click on the button 'Calculate'.

When the button Calculate is clicked on its **click()** event fires. This in turn will activate the JavaScript event handler **onClick**. This invokes the function **calculate()**.

The function **calculate()** evaluates the expression entered in the first **Text** object and displays its output in the second Text object on the form.

The Submit (Button) Element

The submit button is a special purpose button. The submit button submits the **current** data held in **each** data aware, form element to a Web Sever for further processing.

Properties

The submit button is associated with 2 properties:

- name
- value

Method

The submit button's method is:

- click()

Event

The submit button's event is:

- click()

JavaScript provides the following event handler for the Submit button's event:

- onClick().

Example:

```
<INPUT Name="btn_submit" Type="Submit" Value="SUBMIT DATA">
```

This will place a **Submit** button on an HTML form, named **btn_submit**. The Submit button will display **SUBMIT DATA**. When this button is pressed the contents of each data aware, form element will be sent back to a web server for further processing.

Note

There is no example to the functionality of the HTML **submit** button at this stage since this material is only focused on client side JavaScript exclusively.

In subsequent chapters where PERL CGI programming is covered the **GET** and **POST** methods of the **Form** object will be covered with appropriate examples.

The Reset (Button) Element

Properties

The reset button has 2 properties, (*the same as the button*):

- name
- value

Methods

The reset button has only one method associated with it:

- click()

Events

The reset button has only one event associated with it:

- click()

The reset button's JavaScript event handler is:

- onClick( )

Example:

```
<INPUT Name="btn_reset" Type="Reset" Value=" RESET FORM ">
```

This places a **Reset** button on the HTML form, named **btn_reset**. When this button is pressed each data aware form object will be reset to their default values. All user-input values will be initialized.

The Submit and Reset buttons are usually used together on an HTML form. The Submit button will cause the contents of all the data aware form elements to be dispatched to the web server for further processing. The Reset button will empty the contents of the form objects so that the form is ready for reuse.

Note

It is good programming practice to simulate the reset button in a form's unLoad() event.

This will ensure that whenever a form is closed its fields will be emptied so that the next time the form is opened, its form elements will be empty, ready for reuse.

Exercise 5: To illustrate how the **Reset** button on a form functions.

Focus: Create a form having Textboxes, Radio buttons, a Checkbox and a Reset button as shown in diagram 10.6.1. On clicking the Reset button, the entire form is reset (i.e. cleared). Refer to the HTML and JavaScript code described in Exercise 5.

When the Reset button is clicked the form will appear as in diagram 10.6.2.

The diagram 10.6.2 shows an Alert box that displays a message indicating that all the form elements have been cleared.

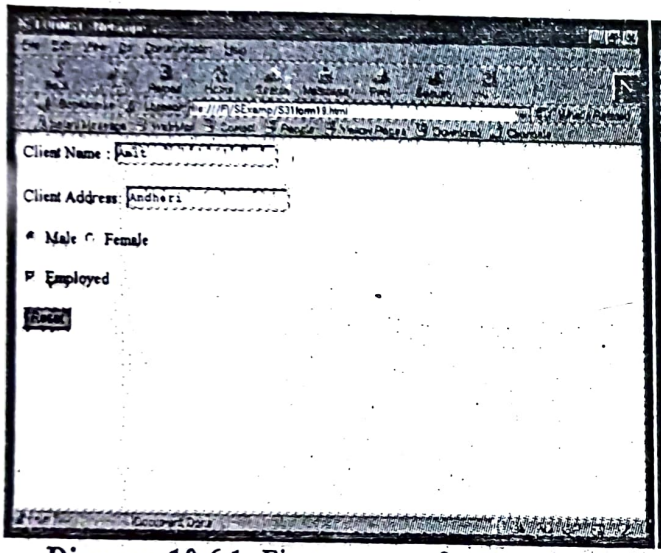

Diagram 10.6.1: First message for Exercise 5.

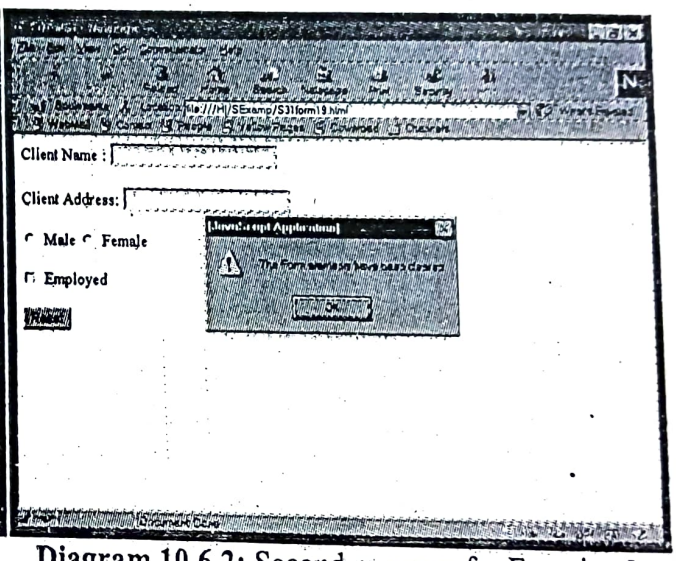

Diagram 10.6.2: Second message for Exercise 5.

The code listing for Exercise 5:

```
<HTML>
   <HEAD><TITLE>FORMS1</TITLE>
<!-- Using Reset Button //-->
   <SCRIPT>
      function func(f1) {   alert("The Form Elements have been cleared");      }
   </SCRIPT></HEAD>
   <BODY><FORM onReset="func(this.form)">
      Client Name : <INPUT Name="Text1" Type="Text" Value="" /><BR/><BR/>
      Client Address : <INPUT Name="Text2" Type="Text" Value="" /><BR/><BR/>
      <INPUT Type="Radio" Name="Radio" Value="" />Male
      <INPUT Type="Radio" Name="Radio" Value="" />Female<BR/><BR/>
```

```
        <INPUT Type="CheckBox" Name="Check" Value="" />Employed<BR/><BR/>
        <INPUT Name="Rst" Type="Reset" Value="Reset" />
    </FORM></BODY>
</HTML>
```

The Checkbox Element

A checkbox is an HTML form object that behaves as a toggle switch. This means a checkbox can be in either one of two states, either **checked** or **unchecked**. A checkbox is used to return a single specific value to a web server. Either **T** or **F** or **1** or **0** can be returned depending upon whether the checkbox is checked or unchecked. Based on the value returned from the HTML form, a web server script can decide what further processing the web server should do.

Properties

- name
- value
- checked

Method

- click()

Event

- click()

A checkbox's JavaScript event handler is:

- onClick()

Syntax:

<INPUT Name="`<NameOfTheObject>`**" Type="checkbox" Value="**`<Yes/No>`**"** CHECKED>

Example:

<INPUT Name="Employed" Type="Checkbox" VALUE="Yes" CHECKED>

This places a **Checkbox** on an HTML form, which can be referenced by the name **Employed**. The Value attribute assigns a meaning to the checkbox. This is the value that is returned if the check box is **Checked**.

The checkbox is checked on (*i.e. marked checked*) by default. Hence, if the check box is not unchecked and the form is submitted to a web server this check box will automatically return the value **Yes**.

Exercise 6:

Focus: Develop a HTML Page, which uses two text fields and a checkbox. The **first** text object accepts a numeric value. Depending on the **checked** or **unchecked** state of the **checkbox**, the **second** text object displays:

Checkbox is not checked	Double the numeric value entered
Checkbox is checked	The square of the numeric value entered

If the **second** text object is loaded with a numeric value depending on the **checked** or **unchecked** state of the **checkbox**, the first text object displays:

Checkbox is not checked	Half the numeric value entered
Checkbox is checked	The square root of the numeric value entered

Whenever the state of the checkbox is changed, recalculation and display of values takes place appropriately. The diagram 10.7 shows a form created to perform mathematical calculations as specified above. Refer to the HTML and JavaScript code described in Exercise 6.

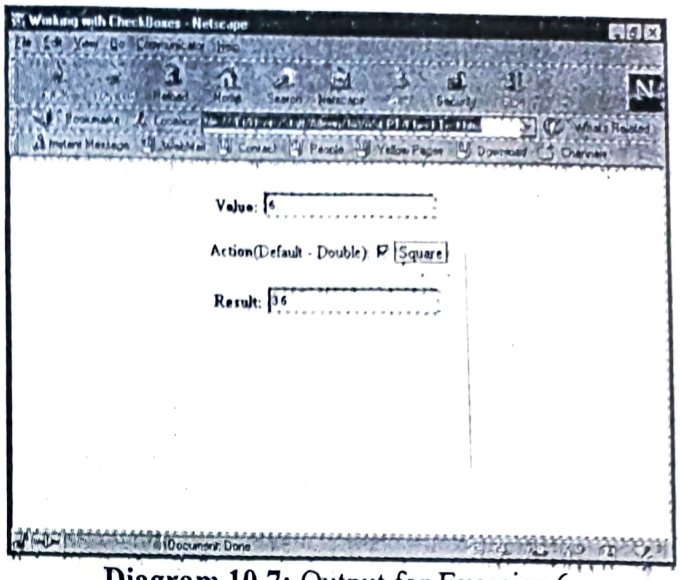

Diagram 10.7: Output for Exercise 6.

The Code Listing for Exercise 6:

```
<HTML>
    <HEAD><TITLE>Working with Check
            Boxes</TITLE>
    <SCRIPT>
        function calculate(form, callingField) {
            if (callingField == "result") {
                if(form.square.checked) {
                    form.entry.value = Math.sqrt(form.result.value);
                }
                else {    form.entry.value = form.result.value/2;      }
            }
            else {
                if(form.square.checked) {
                    form.result.value = form.entry.value * form.entry.value;
                }
                else {    form.result.value = form.entry.value * 2;    }
            }
        }
    </SCRIPT></HEAD>
    <BODY><FORM><CENTER><BR/>
        <B>Value:</B>
        <INPUT Name="entry" onChange="calculate(this.form, this.name);" Type="Text" Value="0" />
        <BR/><BR/><B>Action</B>(Default - Double):
        <INPUT Name="square" onClick="calculate(this.form, this.name);" Type="Checkbox" />Square
        <BR/><BR/><B>Result:</B>
        <INPUT onChange="calculate(this.form, this.name);" Name="result" Type="Text" Value=0 />
    </CENTER></FORM></BODY>
</HTML>
```

Analysis

This example illustrates the use of the **onClick** event handler of a check box. If the check box is *checked* or *not* passes an appropriate Boolean value back to the JavaScript, user defined, function **calculate()**. Depending upon this value a decision is made.

A checkbox named **square** is placed on the HTML form it can be **checked** or **unchecked**.

If a value is placed in the **first** text field on the form and the checkbox is *checked*, the *square* of the value in the first text field will be displayed in the **second** text field. If the checkbox is unchecked *double* the numeric value of the first text field will be displayed in the second text field.

If a value is placed in the **second** text field on the form and the checkbox is checked, the *half* the value held in the second text field will be displayed in the **first** text field. If the **checkbox** it is unchecked then the *square root* of the value in the second text field will be displayed in the first text field.

The *onClick* event handler of the *checkbox* ensures that when checkbox is clicked, to change (*toggle*) its state, all values are recalculated.

The onChange event handler of the *text boxes* ensures that when the values in the text fields change, the form is recalculated.

The Radio Element

The radio button element has two states and can toggle between them. The one special exception is that when several radio buttons are combined into a radio (button) group only a single radio button can be selected at any given time. Giving the **same name** to all the radio buttons places them in the same radio group.

Properties

- checked
- index
- length
- name

Method

- clicked( )

Event

- clicked()

A radio button's JavaScript event handler is:

- onClicked( )

Syntax:

```
<INPUT Type="Radio" Name="<RadioGroupName>" Value="<1/0>"
    CHECKED="CHECKED" />
```

Example:

```
<INPUT Type="Radio" Name="Numbers" Value="1" Checked="Checked" >1<BR/>
<INPUT Type="Radio" Name="Numbers" Value="2" >2<BR/>
<INPUT Type="Radio" Name="Numbers" Value="3" >3<BR/>.
```

This places three *radio buttons* on the HTML form, belonging to the same radio group called *'Numbers'*. The first radio button will be set as *active* as soon as the HTML page is visible.

Exercise 7:

Focus: An HTML form displays two text boxes and two radio buttons. The first text box accepts a numeric value. If the **first** radio button if active, **double** the number entered in the first text field will be displayed in the second text field. If the **second** radio button is active the **square** of the number entered in the first field will be displayed in the second text field.

Whenever the radio buttons are toggled, recalculation takes place and the output is displayed appropriately.

The diagram 10.8 show a form created to perform mathematical calculations as specified above. Refer to the HTML and JavaScript code described in Exercise 7.

The Code Listing for Exercise 7:

```
<HTML>
    <HEAD><TITLE> Working with Radio
            Buttons </TITLE>
    <SCRIPT>
        function calculate(form) {
            if(form.elements[1].checked) {
                form.result.value =
                                form.entry.value
                                * 2;
            }
            else {
                form.result.value =
form.entry.value * form.entry.value;
            }
        }
    </SCRIPT></HEAD>
    <BODY><FORM><CENTER><BR/>
        <B>Value:</B>
        <INPUT Name="entry" Type="Text" Value="0" /><BR/><BR/><SPACER Size="190" />
        <B>Action:<B><BR/><SPACER Size="225">
        <INPUT Name="action1" onClick="calculate(this.form);" Type="Radio" Value="twice" />
        Double
        <BR/><SPACER Size = "225" />
        <INPUT Name="action1" onClick="calculate(this.form);" Type="Radio" Value="square" />
        Square<BR/><BR/>
        <B>Result:</B>
        <INPUT Name="result" onFocus="this.blur();" Type="Text" />
    </CENTER></FORM></BODY>
</HTML>
```

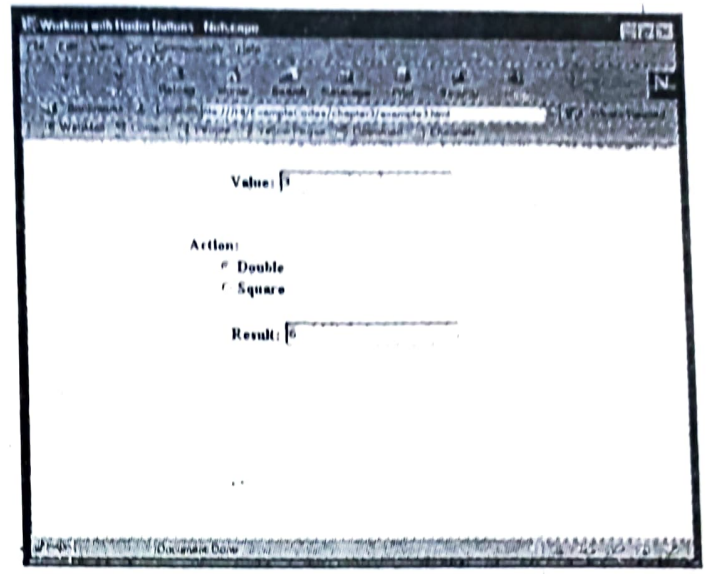

Diagram 10.8: Output for Exercise 7.

Analysis

This example illustrates the use of the *onClick* and *onChange* event handlers.

Two radio buttons are displayed. These radio buttons belong to the same radio group because each of the radio buttons has been given the same **name** as specified in the <INPUT> tag. Each radio button is named *square*.

In the above example, if the first radio button is checked, the program will double the numeric value held in the first text field. On checking the second radio button, the program will square the value held in the first text box. In either case the results of this processing will be displayed in the second text box.

The *onClick* event handler in the radio button ensures that when radio buttons are toggled, the recalculation takes place.

The *onChange* event handler in the text boxes ensures that when changes to values in the text boxes takes place, recalculation is done.

The TextArea Element

The textArea form element provides a way to create a custom sized, multiple line, text entry object, which can be placed on an HTML form.

Properties

- defaultChecked
- name
- value

Methods:

- Focus()
- Blur()
- Select()

Events:

- Focus()
- Blur()
- Select()

The Javacript eventhandlers of a TextArea are:

- onFocus()
- onBlur()
- onSelect()

Syntax:

```
<INPUT TYPE="TextArea" Name="<MyTextArea>" Row="10" Cols="25">
    <H2>Enter Data Here</H2></TEXTAREA>
```

This will place a textArea object on a form. It can be referenced by the name *MyTextArea*. The textarea can accommodate 25 characters per line and 10 such lines inside its user defined boundaries. The boundaries are set by the values passed to the ROW and COLS attributes of the TextArea object.

The Select And Option Elements

A Select object on an HTML form appears as drop-down list or a scrollable list of selectable items. To conserve form space, the scrollable list of selectable items is used.

The list of items to choose from is described between the **<SELECT>...</SELECT>** tags using the **<OPTION>** tag. The <OPTION> tag is not a paired tag.

Example:

```
<SELECT Name="Items">
    <OPTION SELECTED> French Fries
    <OPTION>Hamburgers
    <OPTION>Hot Dogs
</SELECT>
```

This will place a Select object on an HTML form. The select object can be referenced by the name "Items". There will be three choices French Fries, Hamburgers, and Hot Dogs available in the select object. Using the <SIZE> attributes the number of items visible in a select list can be controlled.

If the <SIZE> attribute is set to a value less than the actual choices available in the select list a scrollable list will be created.

The following code snippet will display a drop down list on an HTML form, which displays 2 items and has a vertical scrollbar.

Example:

```
<SELECT Name = "Items" Size="2">
    <OPTION SELECTED>French Fries</ OPTION>
    <OPTION>Hamburgers</ OPTION>
    <OPTION>Hot Dogs</ OPTION>
    <OPTION>Ice Cream Cones</ OPTION>
    <OPTION>Salads</ OPTION>
</SELECT>
```

Using a Select object, only **one** item can be chosen, from the list of items.

Multi Choice Select Lists

To use a Select object, from which multiple choices can be made from within the list the **MULTIPLE** attribute must be set in the select object.

```
<SELECT Name = "Items"  Size ="2" MULTIPLE>
    <OPTION SELECTED>French Fries</ OPTION>
    <OPTION>Hamburgers</ OPTION>
    <OPTION>Hot Dogs</ OPTION>
    <OPTION>Ice Cream Cones</ OPTION>
    <OPTION >Salads</ OPTION>
</SELECT>
```

Selection lists are accessible in JavaScript through the *Select* object. When the <SELECT> </SELECT> HTML tags is encountered in an HTML page a JavaScript enabled browser's creates an array in memory that holds the items available in the list.

In each of the sample code snippets an array of the name *Items* is created in memory. This array has an array index which starts with '0' (i.e. *zero*). The value of any element in the array can be obtained in JavaScript by using:

```
memvarname="arrayname.indexvalue"
```

Properties

- selectedIndex
- defaultSelected
- index
- selected
- text
- value

Methods

- Blur
- Focus
- Change

Events

- Blur()
- Focus()
- Change()

JavaScript Event handlers bound to these events:

- onBlur()
- onFocus()
- onChange()

Exercise 8: Illustrate the use of Select and Option elements on a HTML form.

Focus: The form consists of a two Multiple choice lists and one single choice list.

- The first Multiple choice list displays the Major dishes available.
- The second Multiple choice list displays the Starters available.
- The single choice list displays the Miscellaneous (Milkshakes, Soft drinks, Softy) available.

The selected items from all the lists should be captured and displayed in a Text Area along with their respective costs. On clicking the 'Total Cost ' button, the total cost of all the selected items is calculated and displayed at the end in the Text Area. A 'Clear' button is provided to clear the Text Area.

The diagram 10.9 shows a form that displays the basic form elements required to capture data in a commercial application as specified above.

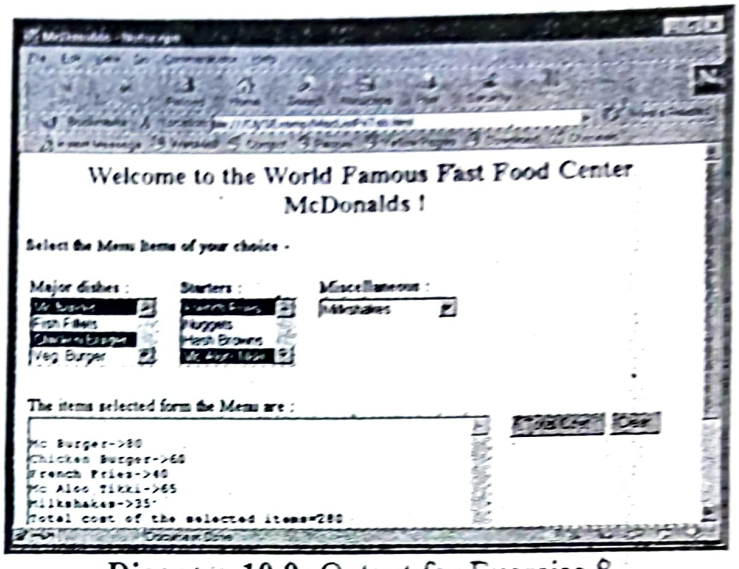

Diagram 10.9: Output for Exercise 8.

The code listing for Exercise 8:

```
<HTML>
   <HEAD><TITLE>McDonalds</TITLE>
   <SCRIPT Language="JavaScript">
      var m;
      function pick(F1) {
         var z=" ";
         for(j=0; j<3; j++) {
            for(i=0; i<F1.elements[j].length; i++) {
               if (F1.elements[j][i].selected) {
                  var y=F1.elements[j].options[i].value;
                  z=z + "\n" + y;
                  F1.elements[3].value=z;
               }
            }
         }
         m=z;
      }
      function cal(F1) {
         var d=0;
         for(j=0; j<3; j++) {
            for(i=0; i<F1.elements[j].length; i++) {
               if (F1.elements[j][i].selected) {
                  var y=F1.elements[j].options[i].value;
                  s=new String(y);
                  var a=s.indexOf(">");
                  var b=s.substring(a+1,a+3);
                  c=parseInt(b);
```

```
                    d=d+c;
                }
            }
        }
        p="Total cost of the selected items=" + d;
        m=m + "\n" + p;
        F1.elements[3].value=m;
    }
    function clr(F1) {
        F1.elements[3].value=" ";
    }
</SCRIPT></HEAD>
<BODY>
    <H2><CENTER>
        <SPAN Style="color: blue; ">Welcome to the World Famous Fast Food Center </SPAN>
        <SPAN Style="color: red;">McDonalds !</SPAN>
    </CENTER></H2>
    <FORM Name="F1">Select the Menu Items of your choice - <BR/><BR/>
    <TABLE><TR valign="Top"><TD>Major dishes :<BR/>
        <SELECT Name="s1" MULTIPLE onBlur="pick(this.form)">
            <OPTION Value="Mc Burger->80" SELECTED> Mc Burger</OPTION>
            <OPTION Value="Fish Fillets->70"> Fish Fillets</OPTION>
            <OPTION Value="Chicken Burger->60"> Chicken Burger</OPTION>
            <OPTION Value="Veg. Burger->45"> Veg. Burger</OPTION>
        </SELECT><BR/><BR/>
    </TD><TD></TD><TD></TD><TD>
        Starters :<BR/>
        <SELECT Name="s2" MULTIPLE onBlur="pick(this.form)">
            <OPTION Value="French Fries->40"> French Fries</OPTION>
            <OPTION Value="Nuggets->50">Nuggets</OPTION>
            <OPTION Value="Hash Browns->55">Hash Browns</OPTION>
            <OPTION Value="Mc Aloo Tikki->65">Mc Aloo Tikki</OPTION>
        </SELECT><BR/><BR/>
    </TD><TD></TD><TD></TD><TD>
        Miscellaneous :<BR/>
        <SELECT Name="s3" onBlur="pick(this.form)">
            <OPTION Value=" ">'Check these out'
            <OPTION Value="Milkshakes->35">Milkshakes</OPTION>
            <OPTION Value="Soft drinks->20">Soft drinks</OPTION>
            <OPTION Value="Softy->25">Softy</OPTION>
        </SELECT><BR/><BR/>
    </TD><TD></TD><TD></TD></TR></TABLE><BR/>
    <TABLE><TR valign="Top"><TD>
        The items selected form the Menu are :
        <TEXTAREA Name="TA1" Rows="10" Cols="50"></TEXTAREA><BR/><BR/>
    </TD><TD></TD><TD></TD><TD>
        <BR/><Input Type="button" Value="Total Cost" onClick="cal(this.form)" />
        <Input Type="button" Value="Clear" onClick="clr(this.form)" />
```

```
        </TD></TR></TABLE>
    </FORM></BODY>
</HTML>
```

OTHER BUILT-IN OBJECTS IN JAVACRIPT

JavaScript provides a few other objects that are not related to the current window or the document loaded in the current window. These objects are used quite extensively for data processing in JavaScript.

The String Object

Every string in JavaScript is an object. The string object has a number of properties, methods, which helps perform a variety of manipulations on a given string. These include methods for searching for a string, extracting sub-strings from a string and applying various HTML tags to string contents and so on.

Properties

The string object has only one property:

Property	Description
Length	An integer value indicating the number of characters in the string.

Table 10.4

Methods

The flexibility and power of the string object rests in the wide variety of methods available to manipulate the contents of the string. Some of the methods available for string manipulation are as follows:

Method	Description
big( )	Surrounds the string with the HTML big tag
blink( )	Surrounds the string with the HTML blink tag
bold( )	Surrounds the string with the HTML bold tag
charat( )	Given an index as an argument, returns the character at the Specified index
italics( )	Surrounds the string with the HTML <I> tag.
tolowercase( )	Makes the entire string lowercase.
touppercase( )	Makes the entire string uppercase.
substring( )	Given two indexes, returns the substring starting at the first index and ending with the character before the last index. If the second index is greater, the substring with the second index and ends with the character before the first index; if the two indexes are equal, returns the empty string.

Table 10.5

Example:

If a variable named **sample contains** a value **"Hello"**, then:

sample.substring(0,3)	returns	Hel
sample.toLowercase()		hello
sample.CharAt(3)		l
sample.italics()		*Hello*

sample.substring(2,4)	returns	ll
sample.toUpperCase()		HELLO
sample.bold()		**Hello**

The Math Object

The math object provides methods and properties to move beyond simple arithmetic manipulations offered by arithmetic operators.

Among the features offered by the **math** object are several special values such as pi, natural logarithms, common square roots, trigonometric methods, rounding methods, an absolute value method, and more.

Properties

Property	Description
E	Euler's constant - the base of natural logarithms.
LN10	The natural logarithms of 10(roughly 2.302).
LN2	The natural logarithms of 2 (roughly 0.693).
PI	The ratio of the circumference of a circle to the diameter of the same circle (roughly 3.1415).

Table 10.6

Methods

Method	Description
abs( )	Calculates the absolute value of a number.
ceil( )	Returns the next integer greater than or equal to a number.
cos( )	Calculates the cosine of a number.
floor( )	Returns the next integer less than or equal to a number.
pow( )	Calculates the value of one number to the power of a second number – takes two arguments.
random( )	Returns a random number between zero and one.
sin( )	Calculates the sine of a number.
sqrt( )	Calculates the square root of a number.
tan( )	Calculates the tangent of a number.

Table 10.7

Example:

abs(-15)	returns	15	ceil(15.45)	returns	16
tan(45)		1	pow(2,2)		4

The Date Object

The Date object enables JavaScript programmers to create an object that contains information about a particular date and provides a set of methods to work with that information. To create an instance of the date object, use the keyword **new** as follows:

var my_date = **new Date(**<parameters>**);**

The parameter, if left empty, indicates today's date & time. The parameter could be any specific date format.

Methods

The date object provides the following methods:

Method	Description
getDate( )	Returns the day of the month as an integer from 1 to 31
setDate( )	Sets the day of the month based on an integer argument from 1 to 31
getHours( )	Returns the hours as an integer from 0 and 23
setHours( )	Sets the hours based on an argument from 0 to 23
getTime( )	Returns the number of milliseconds since 1 January 1970 at 00:00:00.
setTime( )	Sets the time based on an argument representing the number of milliseconds since 1 January 1970 at 00:00:00.

Table 10.8

Exercise 9:

Focus: Write JavaScript code to display the current date and time in a Browser.

The diagram 10.10 shows a form created to display the current date and time in a Browser as specified above. Refer to the HTML and JavaScript code described in Exercise Nine.

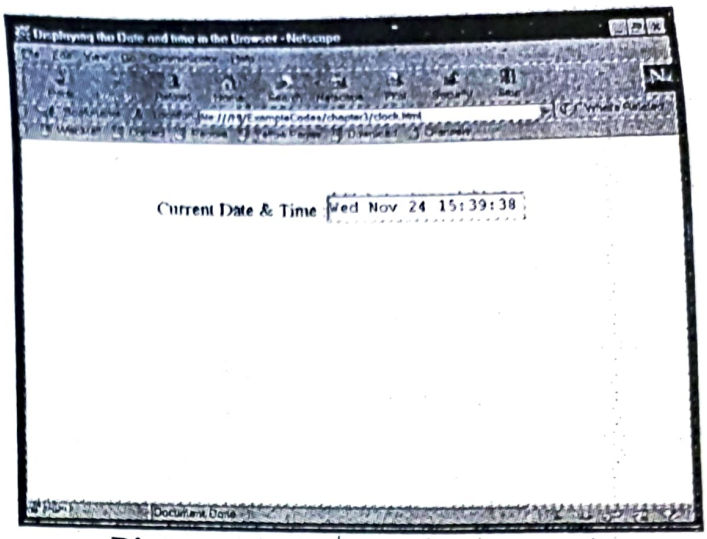

Diagram 10.10: Output for Exercise 9.

The code listing for Exercise 9:

```
<HTML>
    <HEAD>
        <TITLE>Displaying the Date and time in
                the Browser</TITLE>
    </HEAD>
    <BODY>
        <SCRIPT Language="JavaScript">
        function begin(form) {
            form_name = form;
            time_out=window.setTimeout("display_date()",500)
        }
        function display_date() {
            form_name.date.value=new Date();
            time_out=window.setTimeout("display_date()",1000)
        }
        function display_clock() {
            document.write('<CENTER><SPAN Style="color:red;font-size:14px;" >
            <FORM Name=time_form><BR/> <BR/> Current Date & Time :')
            document.write('<INPUT Name="date" Size="19" Value=""
            /></FORM></SPAN></CENTER>')
            begin(document.time_form);
        }
        display_clock();
        </SCRIPT>
    </BODY>
</HTML>
```

Other JavaScript objects can be summarized as follows:

Name	Description
String	The string object enables programs to work with and manipulate strings of text, including extracting substrings and converting text to upper or lowercase characters
Math	The math object provides methods to perform trigonometric functions (sine and tangent) as well as general mathematical functions, such as square roots.
Date	With the date object, programs can work with the current date or create instances for specific dates. The object includes methods for calculating the difference between two dates and working with times.

Table 10.9

USER DEFINED OBJECTS

In addition to the wide range of built-in objects, JavaScript permits the creation of user defined objects. As with every other object, a user-defined object will also be associated with properties and methods, which belong to it. After creation of such an object, any number of instances of this object can be created and used.

For example, if the name, age, and marks obtained by a student needs to be stored and there are fifty such students. It is completely possible in JavaScript to create and object named **Student,** which has three **properties** name, age and marks.

This user-defined object would also require **methods** that will allow the storage of name, age and the marks obtained by a student as properties of the object.

Creating A User Defined Object

A user-defined object called **student** is to be created with three properties:

- Name
- Age
- Marks

The object **student** also will include a method **insert()**.

Just as variables are named containers of data in traditional languages, similarly properties are named containers used to hold data, such as numbers or text in an object oriented approach.

Properties

The properties of the object student can be referenced as:

- Student.Name
- Student.Age
- Student.Marks

Methods

The methods of the object student can be referenced as:

- Student.insert()

Example:

```
function student(name, age, marks) {
    this.name = name;
    this.age = age;
    this.marks = marks;
}
```

This creates an object called student with three properties, name, age and marks.

Note

- **this:** refers to the current object in focus. For example, **this.name** will refer to the name of the current object.
- **Parent:** refers to the parent of the current object.

Instances

In object oriented programming, once an object is defined and created, any number of copies of the object can be created and used to store information. If there are fifty students whose data has to be maintained then 50 copies of the object **student** must be created to store the information of the fifty students.

Every new copy of the object is called an **Instance** of the object. The process of creating an instance of an object is termed as **Instantiation** of the object.

Example:

```
student1 = new student("Anil",10,75);
student2 = new student("Chhaya",9,82);
```

Objects Within Objects

Just as objects consist of properties and methods, they can also consist of other objects. For instance, the marks of a student will further depend on the marks of individual subjects Science, Math and English. Hence, another object can be created called as **Marks**, which has three properties:

- English
- Science
- Math

This object can now be included in the student object, where marks is no more a singular property, but an object called **Marks**, which consists of three individual properties English, Science and Math.

In such a case, a new instance of the object **Marks** must be created first, and this new instance must be included in the object **student**. The properties of this compound object can now be referenced as:

- Student.Name
- Student.Age
- Student.Marks.English
- Student.Marks.Science
- Student.Marks.Maths

Example:

Creating the Marks object:

```
function Marks(English, Science, Math) {
    this.Math = Math;
    this.English = English;
    this.Science = Science;
}
```

Creating the Student object:

```
function Student(name, age, marks) {
    this.name = name;
    this.age = age;
    this.marks = marks;
}
```

The technique of creating an object instance embedded in another object is as follows:

```
anilGrade = new marks(75, 80, 77);
chhayaGrade = new marks(82, 88, 75);
student1 = new student("Anil", 10, anilGrade);
student2 = new student("Chhaya", 9, chhayaGrade);
```

Under these circumstances the property marks of the object student actually holds another object 'anilGrade' or 'chhayaGrade' which in turn hold marks for Math, English or Science.

To access the Math marks of Anil the technique would be:

```
myvar = student1.anilGrade.Math
```

The variable **myvar** will now hold the Math marks obtained by Anil.

Example10:
This example illustrates opening a new window when a link on a web page is clicked. The new window opened, is closed by placing a button on the new window and writing JavaScript code in the **onClick** event of the button

```
<HTML>
   <HEAD><SCRIPT>
      function makeNewWindow() {
         var newwindow = prompt("Enter the document to go to in format http://www.url.com", "");
         if (!(newwindow == null || newwindow == "" )) {
            window.open(newwindow , "", "status, menubar=yes,resizable=no, toolbar=no")
         }
         else {
            var blankwindow=window.open("" ,"", "status, menubar=yes,resizable=no, toolbar=no")
            var windowcontent="<HTML><BODY Style='background-color:#FFFFFF'><H1>
            New window</H1>"
            windowcontent += "<FORM><INPUT Name=close onClick=window.close()"
            windowcontent += " Type=Button Value=Close /></FORM></BODY></HTML>"
         blankwindow.document.write(windowcontent)
         }
      }
   </SCRIPT></HEAD>
   <BODY><FORM>
      <INPUT Name="open" onClick="makeNewWindow()" Type="button" Value="Open URL">
   </FORM></BODY>
</HTML>
```

Example 11:
This example illustrates an application with two frames. The top frame contains entry fields for the background color, text color, link color, active link color and visited link color and a button to enable users to test their color combinations. When the user presses the button, the script loads a simple document using specified colors, into the lower frame.

The Parent Frameset for the Color Tester:

```
<HTML>
   <HEAD><TITLE>Example 11 On Document Object</TITLE></HEAD>
   <FRAMESET Rows="45%,*">
      <FRAME Src="pick.html">
      <FRAME Name="output" Src="blank.html">
   </FRAMESET>
</HTML>
```

Pick.html file

```
<HTML>
   <HEAD><SCRIPT Language="JavaScript">
   <!-- HIDE FROM THE BROWSERS
      function display(form) {
         doc = open(" ","output");
         doc.document.write('<BODY Style="background-color:'+ form.bg.value
         +';color:'+form.fg.value+"");
         doc.document.write('" Link="' + form.link.value + '" ALink="'+ form.alink.value);
```

```
            doc.document.write('" VLink="'+ form.vlink.value + '">');
            doc.document.write("<H1>This is a Test</H1>You Have Selected These Colors<BR/>");
            doc.document.write('<A HRef="# ">This is a Test Link</A></BODY>');
            doc.document.close();
        }
    // STOP HIDING SCRIPT -->
    </SCRIPT></HEAD>
    <BODY><CENTER><SCRIPT Language="JavaScript">
    <!-- HIDE FROM OTHER BROWSERS
        document.write('<H1>The Color Picker</H1><FORM Method=POST>');
        document.write('Enter Colors:<BR/>');
        document.write('Background: <INPUT Name="bg" Type=Text Value="' + document.bgColor +
                       '" /><BR/>');
        document.write('Text: <INPUT Name="fg" Type=Text Value="' + document.fgColor + '"
                       /><BR/>');
        document.write('Link: <INPUT Name="link" Type=Text Value="' + document.linkColor + '"
                       /><BR/>');
        document.write('Active Link: <INPUT Name="alink" Type=Text Value="' +
                       document.alinkColor + '" /><BR/>');
        document.write('Followed Link: <INPUT Name="vlink" Type=Text Value="' +
                       document.vlinkColor + '" /><BR/>');
        document.write('<INPUT onClick="display(this.form);" Type=Button Value="Test" />');
        document.write('</FORM>');
        display(document.forms[0]);
    // STOP HIDING FROM OTHER BROWSERS -->
    </SCRIPT></CENTER></BODY>
</HTML>
```

Example 12:

This example illustrates the use of the History object. Using this object we can navigate through the previous or next pages opened in the browser.

```
<HTML>
    <BODY><FORM>
        <INPUT Name="back" onClick="history.back()" Type="Button" Value="Back" />
        <INPUT Name="forward" onClick="history.forward()" Type="Button" Value="Forward" />
    </FORM></BODY>
</HTML>
```

Example 13:

Capture and displays the properties of the browser using the navigator object.

```
<HTML>
    <HEAD><TITLE>Displaying A Browser's Attributes</TITLE>
    <SCRIPT language="JavaScript">
    <!
        function displayNavigatorProperties() {
        // to avoid "document.write" every where below
            with(document) {
                write("<B>appName:</B>")
                writeln(navigator.appName + "<BR/>")
```