

004  
B33

004.4

**Web Enabled Commercial Applications  
Development Using...  
HTML, DHTML,  
JavaScript, Perl CGI  
3rd Revised Edition**

3232 ✓

By  
Ivan Bayross

764 20344289

Toshkent Axborot Texnologiyalari Universitet  
17. 2765  
Axborot Resurs Markazi



**BPB PUBLICATIONS**

B-14, CONNAUGHT PLACE, NEW DELHI-1

**FIRST INDIAN EDITION 2005**

*Distributors:*

**MICRO BOOK CENTRE**

2, City Centre, CG Road,  
Near Swastic Char Rasta,  
**AHMEDABAD-380009** Phone: 26421611

**COMPUTER BOOK CENTRE**

12, Shrungar Shopping Centre, M.G. Road,  
**BANGALORE-560001** Phone: 5587923, 5584641

**MICRO BOOKS**

Shanti Niketan Building, 8, Camac Street,  
**KOLKATTA-700017** Phone: 22826518, 22826519

**BUSINESS PROMOTION BUREAU**

8/1, Ritchie Street, Mount Road,  
**CHENNAI-600002** Phone: 28410796, 28550491

**DECCAN AGENCIES**

4-3-329, Bank Street,  
**HYDERABAD-500195** Phone: 24756400, 24756967

**MICRO MEDIA**

Shop No. 5, Mahendra Chambers, 150 D.N. Road,  
Next to Capital Cinema V.T. (C.S.T.) Station,  
**MUMBAI-400001** Ph.: 22078296, 22078297

**BPB PUBLICATIONS**

B-14, Connaught Place, **NEW DELHI-110001**  
Phone: 23325760, 23723393, 23737742

**INFO TECH**

G-2, Sidhartha Building, 96 Nehru Place,  
**NEW DELHI-110019**  
Phone: 26438245, 26415092, 26234208

**INFO TECH**

Shop No. 2, F-38, South Extension Part-1  
**NEW DELHI-110049**  
Phone: 24691288, 24641941

**BPB BOOK CENTRE**

376, Old Lajpat Rai Market,  
**DELHI-110006** PHONE: 23861747

**NOTE: THE CD-ROM INCLUDED WITH THE  
BOOK HAS NO COMMERCIAL VALUE AND  
CANNOT BE SOLD SEPARATELY.**

Copyright © **BPB PUBLICATIONS**

All Rights Reserved. No part of this publication can be stored in any retrieval system or reproduced in any form or by any means without the prior written permission of the publishers.

**LIMITS OF LIABILITY AND DISCLAIMER OF WARRANTY**

The Author and Publisher of this book have tried their best to ensure that the programmes, procedures and functions described in the book are correct. However, the author and the publishers make no warranty of any kind, expressed or implied, with regard to these programmes or the documentation contained in the book. The author and publishers shall not be liable in any event of any damages, incidental or consequential, in connection with, or arising out of the furnishing, performance or use of these programmes, procedures and functions. Product name mentioned are used for identification purposes only and may be trademarks of their respective companies.

All trademarks referred to in the book are acknowledged as properties of their respective owners.

Price : Rs. 390/-

ISBN 81-8333-008-8

Published by Manish Jain for BPB Publications, B-14, Connaught Place,  
New Delhi-110 001 and Printed by him at Pressworks, New Delhi.

# Foreword

All programmers know that programming for the Internet is the place to be. This employment window will be open for several years. Being a new and ever evolving environment to code in it will retain its fascination for a long long time.

It really is a fairly traditional Client / Server programming environment. However, separate programming environments are used for Client and Server side programming. This is a strange break from the traditional Client / Server programming environments where both the Client and Server side programming environments have been created by the same vendor.

This I believe is in keeping with the Internet. My perception of the Internet is a huge number of heterogeneous computers with heterogeneous operating systems and programming environments all working together in complete harmony. Hence, if the Client side-programming environment is different from the Server side-programming environment, so be it.

The traditional Client side-programming environment for the Internet is HTML, DHTML, and JavaScript. The traditional Server side-programming environment being CGI scripting using PERL.

Using this rather strange mix for programming environments interactive web sites can be created and deployed on the Internet. There are many books written that cover these environments, so what makes this book different?

Its written in such a way:

- That you will gradually build a static web site using HTML
- Move this skill upward by creating an interactive web site using JavaScript
- Finally, shift to Server side data processing by creating CGI scripts written in PERL.

By the end of your studies and working through the examples in the book a lot of the skills you require to create sound, User interactive Web Sites will be firmly in place. Growth beyond this is by surfing the web and trying to figure out how specific functionality was brought to a website.

There are also a very large number of web sites on the Internet that give away excellent PERL CGI scripts for free. Download these scripts, open them in any ASCII editor (Notepad is a good one) and decipher how they work. Most of these freebies are very well documented using 'Rem' statements in the code block or have complete 'Help' files in HTML or PDF (Acrobat Reader files) format. This will really make you fly.

Every single programming technique using HTML, DHTML, JavaScript, PERL CGI has not been covered. Indeed if I try to actually do that I would fail since I believe that I myself have not encountered every single HTML, DHTML, JavaScript, PERL CGI programming problem. However, I have chosen several key areas in commercial web site development and tried to address a set of issues that most commercial web site developers require.

Concepts are built using simple language. Examples have easily understood logic. Once this is grasped the skill gained must allow any commercial application developer to create an interactive website very very quickly.

I've enclosed a CD-ROM with all the code examples used in the book. I've also added several useful goodies to help you create your first few web sites. I've also added several web site snippets, which would help give you a direction when you are creating you're a web site for the first time.

Having said this I must add that the very best way to create a good, user interactive web site is by surfing the Internet constantly and checking out web site's that interest you. Try and figure how the web site was coded to provide the functionality that it actually has.

My approach, (where possible) right click on an interesting web page and 'View Source'. This will give you an unparalleled insight into the coding techniques used by others to create the web site.

This foreword will not be complete without my thanking the many people who encouraged me and put up with the many revisions and updations of the manuscript with patience and tolerance.

My sincere **THANKS** go to:

- ❑ My publisher Mr. Manish Jain. He has brought enormous changes in my life. This is a debt of gratitude I will never be able to pay in full.
- ❑ Mr. Sharanam Shah, the Tech lead on this project, and a friend, you've done a really splendid job, thanks for all the planning and effort that went into this.
- ❑ To Jennifer Rodrigues, who traveled from Panvel each day, tested each example in this book and formatted this book to my complete satisfaction. Jennifer, honestly this book would have never made it without you and the tender, loving, care you gave it.
- ❑ Mr. Austin Fernandes who criticized the flow of topics in this book and forced Jennifer and I to rearrange chapter and chapter content several times.
- ❑ Mrs. Vaishali Shah who tested the logic flow, grabbed the screens for much of the material. Your attention to detail was superb, you've done a really terrific job.
- ❑ Mr. Hansel Colaco who is in charge of quality control. Who personally took care that everything in the manuscript was rigidly bound to the specifications of our quality manual. You've done a really fine job.
- ❑ The many programmers who read this material, without you all I would not be an author. I welcome both your brickbats and bouquets. You can contact me via BpB, New Delhi, else you could check out my web site appropriately named [ivanbayross.com](http://ivanbayross.com). Regretfully it's forever in a development stage.
- ❑ Finally, my wife Cynthia who has always encouraged me whenever I thought that I'd never get this manuscript ready for publishing. You have always helped to keep my feet firmly on the ground, with you I am truly blessed.

**Ivan N. Bayross**

# TABLE OF CONTENTS

## SECTION – I: Hyper Text Markup Language

<b>1. INTERNET BASICS.....</b>	<b>1</b>
BASIC CONCEPTS.....	1
COMMUNICATING ON THE INTERNET .....	2
INTERNET DOMAINS.....	4
INTERNET SERVER IDENTITIES.....	4
<i>Registering A Virtual Domain With InterNIC</i> .....	4
<i>Domain Name Extension</i> .....	5
ESTABLISHING CONNECTIVITY ON THE INTERNET .....	5
CLIENT IP ADDRESS .....	6
<i>How Client IP Addresses Are Assigned</i> .....	6
<i>How ISP's Achieve The Task Of Assigning IP Addresses</i> .....	6
<i>Getting A Temporary IP Address</i> .....	6
HOW IP ADDRESSING CAME INTO EXISTENCE?.....	7
A BRIEF OVERVIEW OF TCP/IP AND ITS SERVICES .....	8
<i>Internet Protocol</i> .....	9
TRANSMISSION CONTROL PROTOCOL.....	10
<i>World Wide Web</i> .....	10
<i>FTP</i> .....	10
<i>TELNET</i> .....	11
SELF REVIEW QUESTIONS .....	11
<b>2. INTRODUCTION TO HTML .....</b>	<b>12</b>
INFORMATION FILES CREATION .....	12
WEB SERVER.....	12
WEB CLIENT / BROWSER.....	13
Understanding How A Browser Communicates With A Web Server .....	13
Establish Connection .....	13
Client Issues A Request And Server Sends A Response .....	14
Server Terminates The Connection .....	15
HYPER TEXT MARKUP LANGUAGE (HTML).....	15
<i>HTML Tags</i> .....	15
<i>Paired Tags</i> .....	15
<i>Singular Tags</i> .....	15
COMMONLY USED HTML COMMANDS .....	19
<i>The Structure Of An HTML program</i> .....	19
Document Head.....	19
Document Body.....	20
TITLES AND FOOTERS .....	21
Title .....	21
Footer.....	21

TEXT FORMATTING .....	21
Paragraph breaks .....	21
Line Breaks .....	22
EMPHASIZING MATERIAL IN A WEB PAGE .....	24
Heading Styles .....	24
Drawing Lines .....	24
TEXT STYLES .....	25
Bold .....	25
Italics .....	25
Underline .....	25
OTHER TEXT EFFECTS .....	27
<i>Centering (Text, Images etc.)</i> .....	27
<i>Spacing (Indenting Text)</i> .....	27
SELF REVIEW QUESTIONS .....	30
HANDS ON EXERCISES .....	31
<b>3. LISTS .....</b>	<b>32</b>
TYPES OF LISTS .....	32
<i>Unordered List (Bullets)</i> .....	32
<i>Ordered Lists (Numbering)</i> .....	32
<i>Definition Lists</i> .....	33
SELF REVIEW QUESTIONS .....	35
HANDS ON EXERCISE .....	36
<b>4. ADDING GRAPHICS TO HTML DOCUMENTS .....</b>	<b>37</b>
USING THE BORDER ATTRIBUTE .....	37
USING THE WIDTH AND HEIGHT ATTRIBUTE .....	38
USING THE ALIGN ATTRIBUTE .....	38
USING THE ALT ATTRIBUTE .....	39
SELF REVIEW QUESTIONS .....	42
HANDS ON EXERCISE .....	43
<b>5. TABLES .....</b>	<b>44</b>
INTRODUCTION .....	44
<i>The Caption Tag</i> .....	44
USING THE WIDTH AND BORDER ATTRIBUTE .....	45
USING THE CELLPADDING ATTRIBUTE .....	46
USING THE CELLSPACING ATTRIBUTE .....	46
USING THE BGCOLOR ATTRIBUTE .....	47
USING THE COLSPAN AND ROWSPAN ATTRIBUTES .....	47
SELF REVIEW QUESTIONS .....	51
HANDS ON EXERCISES .....	52
<b>6. LINKING DOCUMENTS .....</b>	<b>53</b>
LINKS .....	53
<i>External Document References</i> .....	53
<i>Internal document references</i> .....	54
<i>Hyper Linking To A HTML File (Starting At The Beginning Of The Document)</i> .....	55
<i>Linking To A Particular Location In A Separate Document</i> .....	56

IMAGES AS HYPERLINKS .....	57
<i>Image Maps</i> .....	58
SELF REVIEW QUESTIONS .....	66
HANDS ON EXERCISES .....	66
<b>7. FRAMES</b> .....	<b>68</b>
INTRODUCTION TO FRAMES .....	68
<i>The &lt;FRAMESET&gt; Tag</i> .....	68
<i>The &lt;FRAME&gt; Tag</i> .....	68
<i>Targeting Named Frames</i> .....	69
SELF REVIEW QUESTIONS .....	77
HANDS ON EXERCISES .....	77
<b>A - PROJECTS IN HTML</b> .....	<b>79</b>
<i>Project Specifications For The First Project In HTML</i> .....	79
<i>Project Specifications For The Second Project In HTML</i> .....	87
<i>Project Specifications For The Third Project In HTML</i> .....	93
<i>Project Specifications For The Fourth Project In HTML</i> .....	98
<b>ANSWERS TO SELF REVIEW QUESTIONS</b> .....	<b>110</b>
<b>SOLUTIONS TO HANDS ON EXERCISES</b> .....	<b>112</b>

## **SECTION – II: JavaScript**

<b>8. INTRODUCTION TO JAVASCRIPT</b> .....	<b>117</b>
JAVASCRIPT IN WEB PAGES .....	117
<i>Netscape and JavaScript</i> .....	118
<i>Database Connectivity</i> .....	118
<i>Client side JavaScript</i> .....	118
<i>Capturing User Input</i> .....	118
JAVASCRIPT .....	119
<i>The Advantages Of JavaScript</i> .....	119
WRITING JAVASCRIPT INTO HTML .....	121
BASIC PROGRAMMING TECHNIQUES .....	121
<i>Data Types And Literal</i> .....	122
Number .....	122
Boolean .....	122
String .....	122
Null .....	123
<i>Type Casting</i> .....	123
<i>Creating Variables</i> .....	123
<i>Incorporating Variables In A Script</i> .....	124
<i>The JavaScript Array</i> .....	124
Dense Arrays .....	125
The Elements Of An Array .....	126
<i>The JavaScript Array and its length Property</i> .....	126

OPERATORS AND EXPRESSIONS IN JAVASCRIPT .....	126
<i>Arithmetic Operators</i> .....	127
<i>Logical Operators</i> .....	127
<i>Comparison Operators</i> .....	128
<i>String Operators</i> .....	128
<i>Assignment Operators</i> .....	128
<i>The Conditional Expression Ternary Operator</i> .....	128
<i>Special Operators</i> .....	129
The delete Operator .....	129
The new Operator .....	129
The void Operator .....	129
JAVASCRIPT PROGRAMMING CONSTRUCTS .....	129
CONDITIONAL CHECKING .....	130
<i>The if - then - else Statement</i> .....	130
<i>Immediate if (Conditional expression)</i> .....	131
SUPER CONTROLLED - ENDLESS LOOPS .....	131
<i>For Loop</i> .....	131
<i>While Loop</i> .....	132
FUNCTIONS IN JAVASCRIPT .....	132
<i>Built-in Functions</i> .....	132
USER DEFINED FUNCTIONS .....	133
<i>Declaring Functions</i> .....	133
<i>Place Of Declaration</i> .....	133
<i>Passing Parameters</i> .....	134
<i>Variable Scope</i> .....	135
<i>Return Values</i> .....	135
<i>Recursive Functions</i> .....	136
PLACING TEXT IN A BROWSER .....	136
DIALOG BOXES .....	137
<i>The Alert Dialog Box</i> .....	137
<i>The Prompt Dialog Box</i> .....	138
<i>The Confirm Dialog Box</i> .....	139
SELF REVIEW QUESTIONS .....	141
HANDS ON EXERCISES .....	142
<b>9. THE JAVASCRIPT DOCUMENT OBJECT MODEL .....</b>	<b>143</b>
INTRODUCTION .....	143
<i>Instance</i> .....	144
<i>Hierarchy</i> .....	144
THE JAVASCRIPT ASSISTED STYLE SHEETS DOM [JSSS DOM] .....	144
UNDERSTANDING OBJECTS IN HTML .....	145
<i>Properties Of HTML Objects</i> .....	145
<i>Methods Of HTML Objects</i> .....	146
BROWSER OBJECTS .....	146
THE WEB PAGE HTML OBJECT HIERARCHY .....	148
<i>Access To Elements Of A Web Page</i> .....	148
<i>How A Web Page Element Is Manipulated</i> .....	149
HANDLING (WEB PAGE) EVENTS USING JAVASCRIPT .....	149
<i>Named JavaScript Event Handlers</i> .....	150
SELF REVIEW QUESTIONS .....	151
HANDS ON EXERCISES .....	152



<b>10. FORMS USED BY A WEB SITE .....</b>	<b>153</b>
THE FORM OBJECT .....	153
<i>The Form Object's Methods</i> .....	158
Properties Of Form Elements .....	159
Methods of Form Elements .....	160
<i>The Text Element</i> .....	161
<i>The Password Element</i> .....	161
<i>The Button Element</i> .....	162
<i>The Submit (Button) Element</i> .....	164
<i>The Reset (Button) Element</i> .....	164
<i>The Checkbox Element</i> .....	166
<i>The Radio Element</i> .....	168
<i>The TextArea Element</i> .....	170
<i>The Select And Option Elements</i> .....	170
Multi Choice Select Lists .....	171
OTHER BUILT-IN OBJECTS IN JAVACRIPT .....	174
<i>The String Object</i> .....	174
<i>The Math Object</i> .....	175
<i>The Date Object</i> .....	175
USER DEFINED OBJECTS .....	177
<i>Creating A User Defined Object</i> .....	177
Instances .....	178
<i>Objects Within Objects</i> .....	178
SELF REVIEW QUESTIONS .....	182
HANDS ON EXERCISES .....	183
<b>11. COOKIES .....</b>	<b>184</b>
WHAT ARE COOKIES .....	184
SETTING A COOKIE .....	184
SELF REVIEW QUESTIONS .....	186
<b>B - PROJECTS IN JAVASCRIPT .....</b>	<b>187</b>
<i>Project Specifications For The First Project In JavaScript – Guest Book</i> .....	187
<i>Project Specifications For The First Project In JavaScript – Pen Pals</i> .....	189
<i>Project Specifications For The First Project In JavaScript – Registration Form</i> .....	193
<b>ANSWERS TO SELF REVIEW QUESTIONS .....</b>	<b>199</b>
<b>SOLUTIONS TO HANDS ON EXERCISES .....</b>	<b>200</b>

### **SECTION – III: Dynamic Hyper Text Markup Language**

<b>12. DYNAMIC HTML .....</b>	<b>205</b>
CASCADING STYLE SHEETS .....	205
<i>Font Attributes</i> .....	206
Use Of Font Attributes .....	206
<i>Color And Background Attributes</i> .....	206
Use Of Color And Background Attributes .....	207
<i>Text Attributes</i> .....	207
Use of Text Attributes .....	208

<i>Border Attributes</i> .....	209
Use Of Border Attributes .....	209
<i>Margin Related Attributes</i> .....	209
Use Of Margin Attributes .....	210
<i>List Attributes</i> .....	210
Use Of List Attributes .....	210
CLASS.....	211
Use Of Class .....	211
USING THE <SPAN>...</SPAN> TAG.....	212
EXTERNAL STYLE SHEETS .....	213
Use Of External Style Sheet .....	213
WORKING WITH JAVASCRIPT STYLE SHEETS [JSSS] .....	214
USING THE <DIV>...</DIV> TAG.....	215
Use Of DIVs .....	215
Use Of The DIV Tag And Visibility Property .....	216
LAYERS.....	217
<i>Layer Attributes</i> .....	217
<i>Layer Methods</i> .....	217
<i>Layer Event Handlers</i> .....	218
Use Of Layers .....	218
<i>Inflow Layers</i> .....	219
Inflow Layers: Without Layers.....	219
Inflow Layers: Use Of Layers .....	219
Use Of Drag And Drop .....	221
TO MOVE FORWARD .....	223
SELF REVIEW QUESTIONS .....	223
HANDS ON EXERCISES .....	224
<b>ANSWERS TO SELF REVIEW QUESTIONS.....</b>	<b>225</b>
<b>SOLUTIONS TO HANDS ON EXERCISES.....</b>	<b>226</b>
<b><u>SECTION – IV: CGI-PERL</u></b>	
<b>13. COMMON GATEWAY INTERFACE CONCEPTS.....</b>	<b>228</b>
WHAT IS THE COMMON GATEWAY INTERFACE?.....	228
<i>Why is CGI used?</i> .....	228
CGI – HOW IT WORKS.....	229
<i>How information is transferred from the Web Browser to a CGI program</i> .....	229
The GET Method.....	229
The POST Method.....	230
<i>How a CGI URL is interpreted by the Web Server</i> .....	230
<i>Environment Variables</i> .....	230
<i>How A CGI Program Returns Information To The Server</i> .....	231
<i>Processing HTML Form Information In A CGI Program</i> .....	231
<i>What Is A CGI Program?</i> .....	232
PROGRAMMING LANGUAGES.....	232
<i>Why PERL For CGI?</i> .....	232
SELF REVIEW QUESTIONS .....	232

**14. THE PERL LANGUAGE.....233**

AN INTRODUCTION .....233

INSTALLING AND SETTING UP PERL .....233

*Starting The Install Process OF PERL.....* 233

*Binding The PERL Installed With Apache2.....* 235

*Registering The Changes Made In The httpd.conf With Apache2.....* 235

*Testing the PERL Setup.....* 235

*Changes in the httpd.conf File For The Framework.....* 236

PERL BASICS .....236

PERL STRINGS.....236

*Double Quoted Strings.....* 236

*Single Quoted Strings.....* 237

*Back Quoted Strings.....* 237

THE NEED FOR DATA STORAGE.....238

    Variables .....238

        Scalar Variables.....238

            What is a Scalar Variable?.....238

            Defining Scalar Variables.....238

        Arrays .....239

            Indexed Arrays .....239

            Hash Arrays.....240

ENVIRONMENT VARIABLES AND THE %ENV SPECIAL HASH ARRAY .....242

SELF REVIEW QUESTIONS .....243

HANDS ON EXERCISES .....243

**15. PERFORMING OPERATIONS AND CONTROLLING PROGRAM FLOW .....244**

BASIC ARITHMETIC OPERATIONS.....244

*Auto-Increment And Auto-Decrement Operators.....* 244

OPERATOR SHORTCUTS.....245

COMPARISON OPERATORS.....245

*What Is True (Or False) According To PERL?.....* 245

*Comparing Numbers and Strings.....* 245

*Performing Numeric Comparisons.....* 246

*Relational Operators For Comparing Numbers.....* 246

*Performing String Comparisons.....* 246

*Relational Operators For Comparing Strings.....* 246

CONTROLLING PROGRAM FLOW IN PERL.....247

*Making Decisions In PERL.....* 247

        Using The 'if' Statement.....247

        Using The unless Statement .....248

LOOPS .....249

*Repeating A Task By Looping.....* 249

*Using An while Loop.....* 249

*Using An until Loop.....* 249

*Using A for Loop.....* 250

*Using A foreach Loop.....* 251

*Breaking Out Of A Loop.....* 251

*Skipping An Iteration Of A Loop.....* 252

SELF REVIEW QUESTIONS .....253

HANDS ON EXERCISES .....253

<b>16. PERL FUNCTIONS</b> .....	<b>254</b>
STRING FUNCTIONS .....	254
<i>Chop The Last Character Of A String</i> .....	254
<i>Chomp A Newline Character</i> .....	254
<i>Concatenating Strings</i> .....	255
<i>Repeating Strings</i> .....	255
<i>Extracting A Substring</i> .....	256
<i>The Index Function</i> .....	257
<i>Finding The Length Of A String</i> .....	257
<i>Splitting A String Into Several Parts</i> .....	258
ARRAY FUNCTIONS.....	258
<i>Adding An Array Element</i> .....	258
<i>Removing an Array Elements</i> .....	258
<i>Sorting An Array</i> .....	260
<i>Reversing Array Elements</i> .....	260
<i>Splicing An Array</i> .....	260
<i>Deleting an Associative Array Element</i> .....	261
MATHEMATICAL FUNCTIONS.....	261
TIME FUNCTIONS .....	262
SELF REVIEW QUESTIONS .....	262
HANDS ON EXERCISES .....	263
<b>17. FILEHANDLING</b> .....	<b>264</b>
UNDERSTANDING STDIN AND STDOUT .....	264
<i>The &lt;STDIN&gt; Filehandle</i> .....	264
<i>The &lt;STDOUT&gt; Filehandle</i> .....	265
<i>Using The printf Function</i> .....	265
<i>This Is How It Works</i> .....	265
<i>Redirecting STDIN And STDOUT</i> .....	266
UNDERSTANDING FILES AND DIRECTORIES .....	266
OPENING AND CLOSING FILES .....	266
<i>Using The open Function</i> .....	266
<i>Using The close Function</i> .....	267
<i>File Open Modes</i> .....	267
<i>Reading From The File</i> .....	267
<i>Writing To The file</i> .....	267
<i>Reading From And Writing To A File</i> .....	267
<i>Reading And Writing Text Files</i> .....	268
<i>Reading And Writing Binary Files</i> .....	268
<i>Binary Mode File Access In MS-DOS</i> .....	269
WORKING WITH DIRECTORIES.....	269
<i>To Open A Directory</i> .....	269
<i>To Close A Directory</i> .....	269
<i>Creating A directory</i> .....	269
<i>To Remove A directory</i> .....	270
<i>To Go To The Beginning Of The Directory</i> .....	270
<i>Reading The Contents Of The directory</i> .....	270
TESTING FILES AND DIRECTORIES.....	270
SELF REVIEW QUESTIONS .....	271
HANDS ON EXERCISES .....	271

<b>18. REGULAR EXPRESSIONS .....</b>	<b>272</b>
<b>LEARNING BASIC REGULAR EXPRESSIONS.....</b>	<b>272</b>
<i>Understanding The Basic Form Of A Regular Expression.....</i>	<i>272</i>
<i>Anchoring Patterns.....</i>	<i>272</i>
<i>Matching The Start And End Of A Line.....</i>	<i>272</i>
<i>Matching A Word Boundary.....</i>	<i>273</i>
<i>Using Character Sets In Regular Expressions .....</i>	<i>273</i>
<i>Specifying A Range Of Characters.....</i>	<i>274</i>
<i>Excluding Some Characters .....</i>	<i>274</i>
<i>Matching Any Character .....</i>	<i>274</i>
<i>Repeating A Character Set.....</i>	<i>275</i>
<i>Matching A Specific Number Of Character Sets .....</i>	<i>275</i>
<i>Matching One Fixed Sequence Or Another.....</i>	<i>275</i>
<i>Matching Special Characters.....</i>	<i>275</i>
<i>Finding Lines That Do Not Match.....</i>	<i>276</i>
<i>The while (&lt;math&gt;\infty&lt;/math&gt;) File-Processing Code.....</i>	<i>276</i>
<b>MATCHING PATTERNS IN ANY STRING .....</b>	<b>276</b>
<i>Modifying The Pattern-Matching Criteria .....</i>	<i>276</i>
<i>Making The Pattern Match Case-Insensitive.....</i>	<i>276</i>
<i>Finding All Occurrences Of A Pattern .....</i>	<i>276</i>
<i>Replacing A Pattern .....</i>	<i>277</i>
<i>Referring To A Previous Match .....</i>	<i>277</i>
<b>SELF REVIEW QUESTIONS .....</b>	<b>277</b>
<b>HANDS ON EXERCISES .....</b>	<b>278</b>
<b>19. CREATING STRUCTURED PROGRAMS .....</b>	<b>279</b>
<b>PACKING CODE IN SUBROUTINES.....</b>	<b>279</b>
<i>What Are Subroutines?.....</i>	<i>279</i>
<i>Creating A Simple Subroutine.....</i>	<i>279</i>
<i>Invoking A Simple Subroutine .....</i>	<i>279</i>
<b>PASSING ARGUMENTS TO SUBROUTINES .....</b>	<b>279</b>
<i>Subroutine Arguments .....</i>	<i>279</i>
<i>Initializing Variables From Arguments.....</i>	<i>280</i>
<i>Declaring Local Variables .....</i>	<i>280</i>
<i>Understanding Argument Passing.....</i>	<i>281</i>
<i>Modifying The Argument Value.....</i>	<i>281</i>
<i>Passing A Filehandle To A Subroutine.....</i>	<i>282</i>
<i>Returning Values From Subroutines .....</i>	<i>283</i>
<i>Returning The Last Expression.....</i>	<i>283</i>
<i>Using The Return Function .....</i>	<i>284</i>
<b>LIBRARIES .....</b>	<b>284</b>
<i>Creating PERL Libraries.....</i>	<i>284</i>
<b>CGI PROGRAMMING WITH THE CGI-LIB.PL LIBRARY .....</b>	<b>285</b>
<i>Obtaining cgi-lib.pl .....</i>	<i>285</i>
<i>The cgi-lib.pl Includes The Following PERL Subroutines .....</i>	<i>285</i>
<i>Using cgi-lib.pl.....</i>	<i>286</i>
<b>IMPLEMENTING A FEEDBACK FORM.....</b>	<b>286</b>
<i>Designing The Feedback Form .....</i>	<i>286</i>
<i>Processing User Feedback.....</i>	<i>287</i>

SELF REVIEW QUESTIONS .....	288
HANDS ON EXERCISES .....	289
<b>20. GOING THE OBJECT WAY WITH PERL.....</b>	<b>290</b>
UNDERSTANDING REFERENCES .....	290
<i>Defining A Reference</i> .....	290
<i>Using The Arrow Operator To Deference</i> .....	291
<i>Understanding PERL Packages And Modules</i> .....	292
<i>PERL Packages</i> .....	292
<i>PERL Modules</i> .....	293
<i>Using A Module</i> .....	293
USING OBJECTS IN PERL .....	293
<i>Understanding PERL Objects</i> .....	294
CREATING AND ACCESSING PERL OBJECTS.....	294
SELF REVIEW QUESTIONS .....	295
HANDS ON EXERCISES .....	295
<b>21. DATABASE CONNECTIVITY .....</b>	<b>296</b>
DATABASE ACCESS USING PERL .....	296
THE PERL WIN32::ODBC EXTENSION .....	296
<i>Creating an ODBC Object</i> .....	296
ODBC OBJECT METHODS .....	296
<i>The new Method</i> .....	296
<i>The Connection Method</i> .....	297
<i>The Sql Method</i> .....	297
<i>The FetchRow Method</i> .....	298
<i>The Data Method</i> .....	298
<i>The DataHash Method</i> .....	298
<i>The Close Method</i> .....	299
CASE STUDY – TRAINING INFORMATION AT SCT .....	299
<i>Requirements</i> .....	299
<i>Implementation</i> .....	300
Step I - Create The Table Structures For Oracle.....	300
Step II – Insert data into the PASSWD File and COURSE_DETAIL S File .....	300
Step III – Create an HTML page for User Authentication.....	300
Step IV – Create the check.cgi Script .....	301
SELF REVIEW QUESTIONS .....	302
<b>22. DEBUGGING IN PERL .....</b>	<b>303</b>
LOADING AND LEAVING THE PERL DEBUGGER.....	303
LISTING THE PROGRAM CODE .....	304
<i>Using the l Command</i> .....	304
<i>Using The w Command</i> .....	305
<i>Using the // Command</i> .....	306
<i>Using The ?? Command</i> .....	306
<i>Using the S Command</i> .....	306
USING THE DEBUGGER TO STEP THROUGH A PROGRAM .....	307
<i>Using the s Command</i> .....	307
<i>Using the n Command</i> .....	308
<i>Using the r Command</i> .....	308

<i>Pressing ENTER with the s and n Commands</i> .....	308
<i>Using The X Command</i> .....	309
<i>Using The v Command</i> .....	309
SETTING AND WORKING WITH BREAKPOINTS.....	309
<i>Using The b Command</i> .....	310
<i>Using The c Command</i> .....	310
<i>Using the L Command</i> .....	310
<i>Using the d and D Commands</i> .....	311
DEBUGGING BY PROGRAM TRACING.....	311
DEBUGGING WITH LINE ACTIONS.....	311
<i>Using The a And A Commands</i> .....	311
<i>Using The &lt; And &gt; Commands</i> .....	312
MISCELLANEOUS DEBUGGING COMMANDS.....	312
<i>Using the R Command</i> .....	312
<i>Using The H Command</i> .....	313
<i>Using The ! Command</i> .....	313
<i>Using The p Command</i> .....	313
<i>Using The T Command</i> .....	313
SELF REVIEW QUESTIONS.....	314
<b>23. INSTALLING AND SETTING UP APACHE WEB SERVER.....</b>	<b>315</b>
THE BIRTH OF APACHE.....	315
AN INTRODUCTION TO APACHE.....	315
GETTING STARTED.....	315
<i>Download Apache2</i> .....	315
<i>The Apache 2 Installation Process</i> .....	316
<i>Testing Apache2</i> .....	318
DIRECTORY TREE STRUCTURE OF APACHE.....	318
CONFIGURATION OF APACHE SERVER.....	319
SETTINGS TO BE MADE IN THE HTTPD.CONF FILE.....	319
Understanding Some Important Entries In HTTPD.CONF File.....	320
Global Settings.....	320
VIRTUAL HOSTS.....	321
APACHE MODULES.....	323
<i>Changes in the httpd.conf File For The Framework</i> .....	323
<i>Sample Of The hosts File For The Framework</i> .....	324
<i>Registering The Changes Made In The httpd.conf With Apache2</i> .....	324
SELF REVIEW QUESTIONS.....	324
<b>C. PROJECTS IN PERL.....</b>	<b>325</b>
BUILDING A WEB SITE REGISTRATION SYSTEM.....	325
<i>Web Site Registration</i> .....	325
<i>The User Interface</i> .....	325
<i>Message Of Thanks</i> .....	326
<i>In Case Of An Erroneous Submission</i> .....	326
<i>In Case Of Other Errors</i> .....	326
<i>The Data Stored</i> .....	326

WEB SITE LOGIN .....	327
<i>Server Side Processing In Brief</i> .....	327
<i>Client Side Processing In Brief</i> .....	327
<i>The User Interface</i> .....	328
<i>In Case Of An Erroneous Submission</i> .....	328
<i>In Case Of Other Errors</i> .....	328
<i>The Welcome User HTML Page</i> .....	328
PEN PALS.....	329
<i>The User Interface</i> .....	329
<i>Message Of Thanks</i> .....	329
<i>In Case Of An Erroneous Submission</i> .....	330
<i>In Case Of Other Errors</i> .....	330
<i>Data Storage System</i> .....	330
<i>View Pen Pal Information</i> .....	331
<i>Pen Pal Information Display Format</i> .....	331
<i>The User Interface</i> .....	331
GUEST BOOK.....	331
<i>Message Of Thanks</i> .....	332
<i>In Case Of An Erroneous Submission</i> .....	332
<i>Data Storage System</i> .....	332
UNDERSTANDING THE WEB SITE REGISTRATION SYSTEM.....	333
<i>The Framework On Which A Web Site Registration System Is Built</i> .....	333
<i>Modules Under The Web Site Registration System</i> .....	333
<i>Approach</i> .....	333
<i>Directory Structure</i> .....	333
<i>Files Created For The Web Site Registration System</i> .....	333
PERL Scripts.....	333
HTML Files.....	334
Image Files (.gif).....	334
Flat Files (Storage).....	334
DATA STORAGE SYSTEM.....	334
<i>For Web Site Registration</i> .....	334
<i>For Pen Pals Registration</i> .....	334
<i>For The Web Site Guest Book</i> .....	335
STARTING THE WEB SITE REGISTRATION SYSTEM.....	335
PROCESSING OF WEB SITE REGISTRATION.....	335
PROJECT SOURCE CODE FOR THE INDEX PAGE.....	339
Source Code For index.html.....	339
PROJECT SOURCE CODE FOR THE WEB SITE REGISTRATION SYSTEM.....	340
Source Code For regstrgudis.html.....	340
Source Code For regstr.pl.....	344
PROJECT SOURCE CODE FOR THE WEB SITE LOGIN.....	346
Source Code For login.html.....	346
Source Code For login.pl.....	347
PROJECT SOURCE CODE FOR PEN PALS.....	349
Source Code For pnpls.html.....	349
Source Code For pnpls.pl.....	351
Source Code For vwchpnpls.pl.....	353
Source Code For vwpnpls.pl.....	353



PROJECT SOURCE CODE FOR GUEST BOOK .....	356
Source Code For gestbk.html .....	356
Source Code For gestbk.pl.....	357
<b>D. PROJECTS IN PERL USING A DATABASE .....</b>	<b>359</b>
OBJECTIVE.....	359
<i>Modules</i> .....	359
<i>System Study</i> .....	359
<i>Problems And Solutions</i> .....	360
Add Problem.....	360
Add Solutions .....	360
View Problems And Solutions.....	361
Tips And Tricks .....	362
Add Tips.....	362
View Tips .....	362
Employee Information.....	363
Add .....	363
Edit .....	363
Delete And View .....	364
<i>Validations To Be Considered</i> .....	364
TABLE STRUCTURES.....	365
STARTING THE TIPS AND TRICKS SYSTEM.....	366
PROJECT SOURCE CODE FOR THE TIPS AND TRICKS .....	367
Source Code For Index.htm .....	367
Source Code For login-check.pl .....	367
Source Code For cgi-lib.pl.....	368
Source Code For menu.htm .....	370
Source Code For style.css.....	371
Source Code For header.pl .....	371
Source Code For display.htm .....	371
Source Code For footer.htm .....	371
Source Code For ps.htm .....	372
Source Code For probsol.htm .....	372
Source Code For home.htm .....	372
Source Code For psabout.htm .....	373
Source Code For probadd.pl.....	373
Source Code For mylib.pl.....	374
Source Code For probadd_submit.pl.....	375
Source Code For post_prob.cgi .....	376
Source Code For plist_add.pl .....	377
Source Code For sol_add.pl.....	378
Source Code For soladd_submit.pl.....	380
Source Code For post_sol.cgi .....	381
Source Code For plist_view.pl .....	382
Source Code For prob_det.pl.....	383
Source Code For sol_det.pl .....	384
Source Code For tip.htm .....	386
Source Code For tippg.htm.....	386
Source Code For ttabout.htm .....	387
Source Code For tipadd.pl .....	387
Source Code For tipsubmit.pl.....	388

Source Code For post_tip.cgi.....	389
Source Code For tiplist.pl.....	390
Source Code For tipinfo.pl.....	391
Source Code For ei.htm.....	392
Source Code For employee.htm.....	392
Source Code For eiabout.htm.....	393
Source Code For eiadd.pl.....	393
Source Code For eiadd_submit.pl.....	395
Source Code For post_eiadd.cgi.....	396
Source Code For eieditlist.pl.....	397
Source Code For eiedit_info.pl.....	398
Source Code For eiedit_submit.pl.....	400
Source Code For post_eiedit.cgi.....	402
Source Code For eidel.pl.....	402
Source Code For eidel_info.pl.....	403
Source Code For eidel_submit.pl.....	405
Source Code For post_eidel.cgi.....	405
Source Code For eiview.pl.....	406
Source Code For eiview_info.pl.....	407
<b>ANSWERS TO SELF REVIEW QUESTIONS.....</b>	<b>410</b>
<b>SOLUTIONS TO HANDS ON EXERCISES.....</b>	<b>413</b>
<b><u>SECTION – V: Appendix</u></b>	
<b>APPENDIX - A.....</b>	<b>419</b>
<i>HTML Color Codes.....</i>	<i>419</i>
<i>Resource On World Wide Web.....</i>	<i>419</i>
<i>Resources On HTML Editors.....</i>	<i>420</i>
<i>Resources On HTML Document Development.....</i>	<i>420</i>
<b>APPENDIX - B.....</b>	<b>421</b>
<i>Resources On JavaScript.....</i>	<i>421</i>
<b>APPENDIX C.....</b>	<b>421</b>
HTTP.....	421
<i>PERL Error Messages.....</i>	<i>423</i>
<i>Resources On CGI And PERL.....</i>	<i>427</i>

# SECTION - I: HTML

## 1. INTERNET BASICS

### BASIC CONCEPTS

The Internet began in 1969, as an experimental four-computer network called ARPAnet, which was designed by the U.S. Defense Department so that research scientists could communicate. In approximately two years, ARPAnet grew to about two-dozen sites and by 1981, consisted of more than two hundred sites. In 1990, ARPAnet was officially disbanded and the network, which now consisted of hundreds of sites, came to be known as the Internet.

After a while, commercial organizations began to recognize the use of such a network which converted the whole world into a **Global Village** and allowed almost instant access to business or commerce data and a host of other services such as E-Mail and E-Commerce. The rapid growth of the Internet was due to networking giants like British Telecom, Hyundai, AT&T and others setting up fast and reliable networks that encircled the globe. The networking giants were very clear about their role. This was to setup and maintain, monitor and expand existing networks. Hence another layer was formed above this layer called ISP's (**I**nternet **S**ervice **P**roviders). The networking giants gave access to the Internet via **Gateways**. Using ISP gateways it is perfectly possible to route business or commerce data from one point of the globe to another by using a heterogeneous mix of networks owned by different networking giants who have worked out an agreement between themselves on the costs of usage.

ISP's in turn offer clients' access to the Internet via their gateways as paid for service. An ISP's gateway generally consists of a server with a permanent connection to the Internet. The Server's connection to the Internet is called its Internet **Pipeline**. Special hardware is used as the pipeline to connect an ISP's server to the Internet. ISP pipeline bandwidths of 2GB to 10GB are quite common. Multiple pipelines can be purchased by an ISP from a networking giant and used.

Clients that log into the Internet via an ISP commonly use only 33.6 Kilobyte of the ISP's bandwidth. Occasionally a client whose data traffic is very high will use special connectivity methods (ISDN) to an ISP's server and use between 64Kbps to 128Kbps of the ISP's bandwidth. If a client has huge data traffic then a client could directly negotiate with a networking giant to have its own private **Gateway** to the Internet and then make use of the huge bandwidth for its data transfers. These types of clients are not ISP's and as a general rule do not allow other clients access to their gateway. Even in such case physical access to the Internet is a server. The diagram I.1 illustrates what was said earlier:

The Internet consists of two types of computers **Servers** and **Clients**

- Computers which offer information to be read are called **Servers**
- Computers that read the information offered are called **Clients**

Servers run special software (*Web Server software*) that allows them to

- Respond to Client requests for information.
- Accept data from Clients

Some of the most popular software which Servers run to allow them to respond to Client requests for information are, *Internet Information Server (IIS)*, *Apache Web Server*, *Microsoft Personal Web Server*.

Clients run special software (*Browser software*) that allows them to

- Locate the appropriate Server
- Query the Server for the information to be read

Some of the most popular browser software that Clients run, to allow them to query Internet Servers for information are *Netscape Communicator*, *Internet Explorer*.

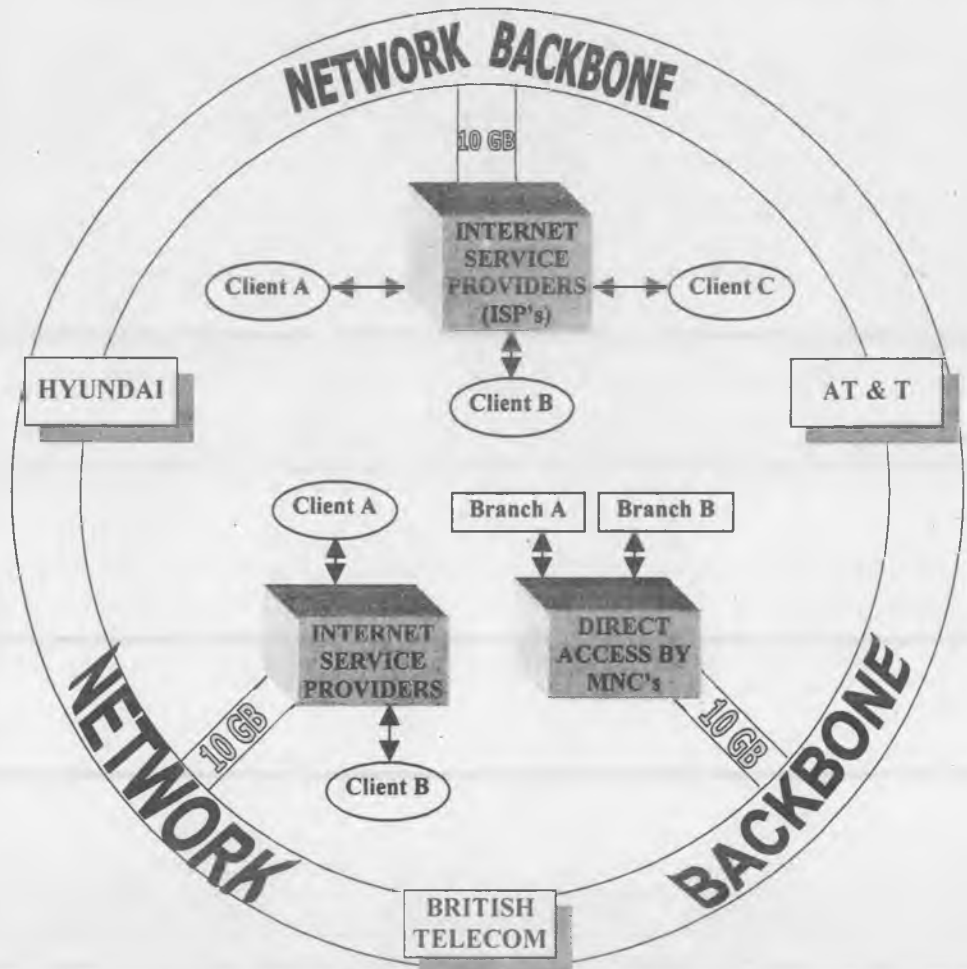


Diagram 1.1

## COMMUNICATING ON THE INTERNET

TCP/IP is the only protocol used to send data all around the Internet. TCP/IP is really two individual sections, (TCP) a set of communication protocols and (IP) a unique address. Every machine connected to the Internet must have an address by which it can be located on the Internet. This is called the *IP address* of the machine. For the Internet to function smoothly, no two machines can have the same *IP address*. Hence each machine connected to the Internet must have a unique *IP address*, which identifies that machine.

The Internet is a worldwide network of networks. As the Internet grew over the years it became increasingly important to have a governing body, which allocated unique IP addresses to organizations linked to the Internet. An international body called InterNIC, located in the USA, is responsible for registering and assigning unique *IP addresses* to organizations wishing to manage networks, which will be part of the Internet. A unique *IP address* therefore points to an *actual computer* connected via a gateway to the Internet. This computer is known as a **Domain** i.e. a place where information is available. This is a Physical Domain on the Internet.

Conceptually, a Server, which has a permanent IP address (i.e. a *Physical Domain*), can provide

- A gateway to other computers to access the Internet

**And/Or**

- Provide information for Internet clients to read

**And/Or**

- Provide a physical location on which *several* **Virtual Domains** can be **Hosted**

In many cases, when a **Web Site** provides Internet clients information to read, the site is mounted as a **Virtual Domain** on an Internet Server, which is its (*host*) **Physical Domain**.

*Virtual Domains* are identified by a name (e.g. www.microsoft.com). Just like a *Physical Domain* needs to have a unique IP address, so also *Virtual Domain Names* need to be unique on the Internet. All *Virtual Domain Names* must be registered with InterNIC. One Internet Server may host several virtual domains.

Virtual domains can be conceptualized as sub directories on an Internet server's hard disk drive. The information that Internet clients wish to read would be **Files** within the sub directory.

When an Internet client connects to an Internet server some software running on the Internet server must respond to the Internet client's request for information. The software that runs on an Internet server and responds to an Internet client's request for information is called **Web Server** software.

An Internet Web server responds to an Internet client's request for information by going to a specific sub directory on its hard disk and forwarding a *pre-determined* file to the Internet client for the first time. After that the Web server will forward files to the client depending upon specific requests from the client.

Traditionally, this sub directory on the Internet server is '*wwwroot*' and the file automatically picked up and passed to the Internet client the first time is traditionally called '*index.html*'.

## Note



---

However, web server software can be configured to go to any *user defined* sub directory and forward any *user defined* file to the Internet client.

---

### Multiple Virtual Domains hosted on a Single Physical Domain:

A single computer having a permanent IP address and connected to the Internet can host *multiple virtual domains* on its hard disk drive. There are several Internet servers (i.e. physical domains) which do nothing but **Host** multiple virtual domains.

Companies who wish to have an Internet presence without registering a physical domain with InterNIC and then creating a virtual domain on the physical domain simply rent hard disk space on these **Host** servers. Most often host servers rent space on their hard disk drives in 25MB blocks.

Host servers offer several additional services such as multiple e-mail boxes, web site creation, web site creation tools and so on. These services makes the prospect of renting hard disk space and creating a virtual domain on someone else's physical domain very attractive due to competitive pricing. The price of hosting a virtual domain on someone else's Internet server is really a very tiny fraction of the setup costs of a physical domain on the Internet.

This has given rise to a situation where the number of virtual domains outnumbers the physical domains on the Internet.

Internet clients will now not only need to connect to the correct physical domain but also to the correct virtual domain hosted on the physical domain. Hence, the Web server software run on the physical domain must perform *Virtual Domain Name* resolution as well (i.e. *must be able to connect Internet client to the correct sub-directory on the hard disk drive and forward the correct starting file from this specific sub directory*).

Conceptually, a sub-directory is created on the Internet 'Host' server. The company, which has purchased hard disk space on the Internet host server, creates its web site within this sub directory. This sub directory will hold the default startup file (e.g. *Index.html*) which the web server will send back to the client when the client connects to the physical domain and points to a specific virtual domain hosted on the physical domain.

When an Internet client requests for a connection to a virtual domain on the Internet, the request is routed to the proper Internet Server using TCP/IP. The Web Server running on this Internet Server then handles the request, resolves (*maps*) the *Virtual Domain Name* sent along with the request to an appropriate sub directory on the Internet Server where the web site (i.e. *virtual domain*) is hosted.

## INTERNET DOMAINS

Each computer that has a permanent IP address, runs Server software and offers information to clients, is considered to be a **Physical Domain** i.e. a place (*domain*) where information is available.

### The Physical Domain

All Internet Servers are connected to the Internet via an **Internet Gateway**. An Internet gateway is usually provided by one of the global networking giants that have setup these gateways. These networking giants are the companies who have spent enormous sums of money to set up the physical networks that circle the globe.

These physical networks are called the **Internet Backbone**. The Internet backbone is a heterogeneous mix of networking technologies, which have been successfully implemented and currently fully operational. The networking giants are companies like British Telecom, Hyundai, AT&T etc.

## INTERNET SERVER IDENTITIES

InterNIC, a quasi government body in the U.S.A registers and issues Internet Servers their unique IP addresses.

InterNIC also authorizes organizations in other countries to issue IP addresses. In India, NCST (National Center of Software Technology) a quasi government body is authorized to issue permanent IP addresses. This really means that the Internet Server must be located within India.

### Registering A Virtual Domain With InterNIC

Log onto an InterNIC Server. Fill up a registration form online. Pay the two-year registration fee to InterNIC. This registers a virtual domain with InterNIC.

Detailed Registration will ask for information like Personal Details, Billing Contact Information, Technical Contact Information, Administrative Contact Information and so on.

When a company registers a virtual domain name with InterNIC, InterNIC requires a unique IP address to be specified. The unique IP address specified, is generally the IP address of the Internet server, which will host the **Virtual Domain**.

InterNIC's address is <http://www.internic.net/regist.html>

- ❑ InterNIC will scan its registered database to ensure that the domain name is unique
- ❑ If the name is unique, InterNIC will inform the registering company or individual that its registration has been accepted.
- ❑ Once the Domain name is registered the registering company has 30 days within which payment has to be made to InterNIC for the registration. Once registered, the Domain registration is valid for 2 years. After a two year period the registration must be renewed or the registration lapses.
- ❑ If the name is not unique, the registration is rejected and InterNIC will inform the company the registration has been rejected due to the Domain name being duplicate.

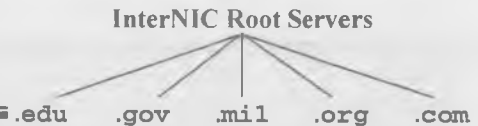
**Note**



InterNIC allows a company registering a Domain name to scan through its domain name database using a simple but elegant user interface. Hence, prior actual registration a company or individual can find out whether the domain name being registered is unique or not.

As the number of Domain registrations grew by leaps and bounds InterNIC began segregating Domains according to the services provided by the Domain being registered. This was done by adding an *extension* to the Domain name. The extension to the domain name indicated the kind of service being provided by that domain.

A Domain name extension structure is as follows:



**Note**



InterNIC has spread its *Root Servers* geographically across the United States to be able to handle the large number of physical Domain Name Servers being registered and virtual domain names being hosted on these servers.

**Domain Name Extension**

.edu	Servers that provide Educational services.
.gov	Servers that provide information about the government of a country
.mil	Servers that provide Military information
.org	Servers that provide information about the Organizations in the world.
.com	Servers providing commercial services on the Internet

Table 1.1

**ESTABLISHING CONNECTIVITY ON THE INTERNET**

The protocol used to setup communications between a Client and Server on the Internet is TCP/IP. TCP/IP, expands to read Transmission Control Protocol / Internet Protocol. TCP/IP is a connection independent protocol.

This means that TCP/IP works completely independent of the physical media used to create the network. The network can be a heterogeneous mix of any of the following networking technologies Ethernet, VSat, Fiber Optics, Infrared, VHF/UHF radio frequencies and so on, TCP/IP will work across all quite transparently. TCP/IP breaks up data into datagrams and guarantees that the data is correctly received at its destination

## CLIENT IP ADDRESS

### How Client IP Addresses Are Assigned

Computers that only read information offered, (i.e. Internet clients) need not necessarily have a permanent IP address. However, when logged into the Internet, a client requires a unique IP address. This IP address enables the Internet server called, to reply accurately.

The Internet Service Provider's (ISP's) Server via which the client connects to the Internet, temporarily assigns a unique IP address to the client.

In India, for a very long time, the only ISP was Videsh Samachar Nigam Limited. (i.e. VSNL). Whenever a client logs into the Internet via a VSNL Server, the VSNL Server temporarily (*and dynamically*) assigns a unique IP address to the computer that successfully logs in.

### How ISP's Achieve The Task Of Assigning IP Addresses

ISP's purchase a block of unique IP addresses from internationally recognized networking bodies. Thus whenever a client logs into the Internet via an ISP's Server, one of these unique IP addresses is temporarily assigned to the computer, which logs in.

### Note



Each time the same Client logs into an ISP's Server, the Client will be assigned a unique but different IP address temporarily. This is the technique of **Dynamically** assigning an IP address to a client when required.

The maximum number of computers that can log into an ISP Server and access the Internet is therefore limited to the block of unique IP addresses purchased by that ISP from the international networking body.

### Getting A Temporary IP Address

Assuming that the Client computer is a Windows machine, its TCP/IP stack is configured to Get an IP address dynamically from the ISP's DHNS Server.

The Client computer logs into the ISP's Server using a dial-up line and a modem. After a successful login is accomplished, the ISP's Server automatically passes one IP address from the block of IP addresses purchased, to the computer logged in.

Once a computer is logged on the Internet and has a unique IP address, any other computer on the Internet that is aware of this IP address can setup communications with it. *i.e. all the Clients become visible to all*

*the other computers on the Internet, via their unique IP address through the ISP's Server.*

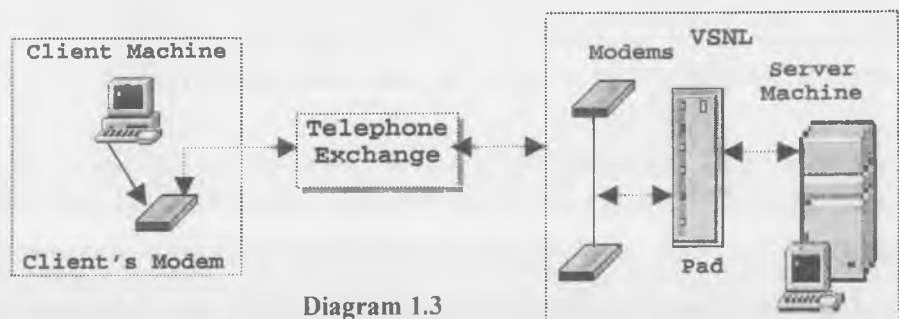


Diagram 1.3



## HOW IP ADDRESSING CAME INTO EXISTENCE?

When the Internet was in its infancy, the US military, colleges and academics largely used it to exchange data. One master Server (called **Root**) held the entire set of computer IP addresses.

These IP addresses were held as pure text in a file called **Hosts** held in a sub-directory on the **Root** Server.

### Note



---

On an M.S. Windows computer, to locate the **Hosts** text file search in: **C:\Windows**.  
(*Replace 'C' with an appropriate drive letter*)

---

The contents of the **Hosts** file is as follows:

IP address	Domain Name
202.100.17.33	Microsoft

i.e. a unique IP address of the Internet Server, followed by a single **Tab** character, followed by a domain name.

### Resolving Domain Names

When any Client wishes to communicate with any Internet Server using browser software, the Client sends out a broadcast using the Server's *Domain name* as the identifier.

The ISP's Internet Server intercepts this request first.

Should the **Domain Name** be unknown to the ISP's Internet Server.

The ISP's Internet Server will route this broadcast to an InterNIC Root Server.

In the InterNIC Root Server's *Hosts* file the Domain Name will be mapped to an IP address.

This IP address will be returned to the ISP's Server

The ISP's server will now pass this IP address back to the Client's browser

The Client's browser will now broadcast a request to connect directly to the Internet Server using its IP address.

As soon as this call is heard by the Internet Server it will respond and a link will be setup between the Client and the Internet Server.

Web Server software running on the Internet Server takes care of connecting a Client to the virtual domain as required.

Once a client is connected to the appropriate domain, the Web Server software delivers the Client the first HTML page of that domain which is traditionally *Index.html* or *Index.htm* and browsing of the specific domain can begin.

### Note



---

Web Server software, can be configured to forward any HTML page (or run a CGI script) when a Client connects.

---

In case some other Client has already connected to the same Internet Server via the same ISP then the server's IP address will be cached on the ISP's Server.

If this is true, the ISP's Server already knows the Internet Server's unique IP address. Hence when the Client broadcasts a call to the Internet Server the ISP's Server will directly route the call to the appropriate Internet Server.

If a client already knows the IP address of the Internet Server, directly keying in the IP address into the Browser's search window will establish a connection between the Client and the Internet Server in the shortest possible time. For example, 127.45.27.18, (i.e. *the actual IP address of the Internet Server*).

### The Structure Of An Internet Address

The structure of an Internet Server's address keyed into a Client's browser software is as follows:

**http://www.microsoft.com**

where; **http** is the communication protocol to be used,

**www** is the notation for World Wide Web,

**microsoft** is the registered Domain Name associated with the IP address of an Internet server, and

**com** the Server provides commercial services to Clients who connect to it.

To help speed up access, its IP address can be directly used, i.e. 127.57.13.1 instead of the domain name, *microsoft.com*. In this case no name resolution needs to take place.

### Note



Only the Domain Name (i.e. *microsoft*) will be resolved to an explicit IP address by looking up the InterNIC Server. The period used in the Domain address is a parameter delimiter.

*http*, *www*, *com* are all optional information that is built into the Domain address loaded in to the Client's browser.

*microsoft.com* is the *compulsory segment* of a domain address as it is 'resolved' into a specific and unique IP address that points to a server.

## A BRIEF OVERVIEW OF TCP/IP AND ITS SERVICES

TCP/IP is an acronym for *Transmission Control Protocol / Internet Protocol*. TCP/IP is a collection of protocols, applications and services. The protocols in TCP/IP move data from one network layer to another. There are five layers within TCP/IP.

- Application
- Transport
- Internet
- Data Link
- Physical

### The Physical Layer

The physical layer is pure hardware in any network infrastructure. This includes the cable, satellite, or any other connection medium, and the network interface card, which transmits electrical signals, and so on.

### The Data Link Layer

This is the layer that is responsible for splitting data into packets to be sent across the connection medium such as cables, satellites, and so on. Once the data is on the communication link, the Data Link Layer handles any interference, which may arise due to noise, solar flares, and so on. The Data Link Layer works hard to make sure that the physical link does not garble the electrical signals carrying the data.

### The Network Layer

This layer gets packets from the Data Link Layer and sends them to the correct network address. If more than one possible route is available for the data to travel, the network layer figures out the best route. The network layer determines that data is sent to the right address via the most convenient (*need not be the shortest*) route.

### The Transport Layer

Though the Network Layer routes data to its destination, it cannot guarantee that the packets holding data will arrive in the correct order or that they won't have picked up any errors during transmission. It is the Transport Layer's job to make sure that the packets have no errors and they are also received in the correct order.

### The Application Layer

This is the layer, which contains the application that the user uses to send or receive data. Without this layer the computer and its user would never be able to send data and would not know what to do with data sent by another user.

### Internet Protocol

The Internet Protocol is responsible for basic network connectivity. When mapped to the TCP/IP layers, the Internet Protocol or IP works with *The Network Layer*. In networking there has to be a physical location to send data to or receive data from.

To make this happen, every physical location must have a unique network address. This address is called its IP address. Hence, every computer on a TCP/IP network must have an IP address, which is unique to that computer.

This IP address can be compared to a postal address that identifies the exact location of a residence or corporate house. Just as two residences cannot have the same postal address, so also no two computers on a TCP/IP network can have the same IP address.

### The Structure of an IP Address

The IP address is a set of numbers separated by periods. An IP address is a 32-bit number, divided into two sections, the network number and the host number. Addresses are written as four fields, eight bits each separated by a period. Each field can be a number ranging from 0 to 255. This method of addressing is called *dotted decimal notation*. An IP address looks like:

**field1.field2.field3.field4**

All hosts/network interfaces (NIC cards) on the same network use the same network number. Each host/network interface (NIC card) on the same network must have a unique host number.

The four fields of an IP address are clubbed together into two sections, the network number and the host number depending on the type of network the host or the network interface belongs to.

There are three types of networks on the Internet. Table 1.2 gives a classification of the network types and how the fields of an IP address are mapped to the Network section and the Host section of the address.

Network Class	Network Section of the IP Address	Host Section of the IP Address
A	Field1	Field2.Field3.Field4
B	Field1.Field2	Field3.Field4
C	Field1.Field2.Field3	Field4

Table 1.2

## TRANSMISSION CONTROL PROTOCOL

TCP/IP uses IP to deliver packets to the upper layer applications and provides a reliable stream of data among computers on the network. Once the packets arrive at the correct IP address, TCP goes to work. TCP's main task is error checking to make sure that the right numbers of packets are received and that they are in proper order. Thus, TCP guarantees that the information received by a computer on a TCP/IP network, is exactly the same information that was sent to it by another computer on the network.

TCP/IP consists of protocols, applications, and services. Protocols enable a server application to offer services, and the client application to use those services. It is possible to design a new protocol and add it to TCP/IP.

The Internet is a large worldwide network of computers, which uses TCP/IP as the underlying communication protocol. Since TCP/IP consists of various services, the Internet is in a position to offer those services to all computers connected to the Internet.

Following is a brief description of some of the commonly used services along with the protocols they use.

### World Wide Web

The World Wide Web is a worldwide information service on the Internet. The World Wide Web or the Web, as it is popularly known, uses special software called a Browser (client) and TCP/IP, HTTP and a Web Server to function.

TCP/IP is the communication protocol used by the Internet and is a must for the World Wide Web to function. HyperText Transfer Protocol or HTTP is the protocol used by the WWW service to make communication possible between a Web Server and a Web Browser. A Web Server is a special software, which runs on a computer and responds to requests made by other computers on the network. A Web Browser is simply an application program, which sends request to a Web Server and accepts a response to that request from the Web Server.

### FTP

File Transfer Protocol or FTP is not just a *protocol* but also a *service* and an *application*. FTP is especially useful for transferring files between different computers. FTP provides the facility to transfer files between two computers running on different operating systems such as UNIX, MS-DOS and Windows.

#### FTP As An Application

For two computers to actually make use of the FTP service, both computers require special application software, which understands this service. FTP is an application for copying files. A client application can be run on the local computer to contact the FTP server application on the remote computer. Depending upon what the user wants to do, instructions can be given to the client application, which works with the server application to execute those instructions.

CuteFTP and Reachout are two very popular FTP applications, which provide excellent user interfaces and a wide range of FTP commands and functions. CuteFTP permits the user to log onto a remote computer offering the FTP service, upload or download files between the client computer and the server computer, perform various operations on the files such as delete, rename, change file permissions, and so on and logout from the remote computer when done.

#### FTP As A Service

FTP is a service for copying files from one computer to another. A connection can be made from one computer (client) to another computer (server) offering this service and files can be sent or received.

#### FTP As A Protocol

FTP is a protocol for copying files between two computers. The client and the server applications both use it for communication to ensure that the new copy of the file is identical to the original.

## TELNET

Telnet is both a TCP/IP application and a protocol for connecting a local computer to a remote computer. The Telnet application acts as a terminal emulator. Whatever commands are typed into the local computer are sent across the network for execution by the remote computer.

To use Telnet, the Telnet application must be given the IP address of the computer to connect to. Once the connection is established with the remote computer, a username and a password must be supplied to access the resources of the remote computer.

### In Conclusion

The Internet is a worldwide collection of interconnected computer networks that use TCP/IP and its related services. TCP/IP provides the flexibility to add new protocols and services to the already existing ones. This led to the advent of protocols such as Hyper Text Transfer Protocol (HTTP) and its associated World Wide Web (WWW) service, the File Transfer Protocol (FTP) and its associated FTP services, and so on. The Internet gained popularity primarily due to the advent of the World Wide Web or the Web, as it is popularly known.

Once the Internet's addressing system is understood it is quite simple to link to an Internet Server anywhere on the Internet. Once linked to the appropriate Internet Server, a Client can access all that the Server has to offer with complete freedom.

The Internet really is a rich resource for information and data. Never ending, always changing and completely dynamic.

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. A permanent connection to the Internet is obtained by the ISP through its \_\_\_\_\_.
2. Servers run special software called \_\_\_\_\_ that allows them to respond to a client's request for information.
3. Each computer that has a permanent IP address, runs Server software and offers information to clients is considered to be a \_\_\_\_\_.
4. \_\_\_\_\_ a quasi government body in the U.S.A registers and issues Internet servers their IP addresses.
5. The protocol used to setup communications between a Client and Server on the Internet is \_\_\_\_\_.
6. \_\_\_\_\_ is the protocol used by the WWW service to establish a communication between a Web server and a Web Browser.
7. The default HTML page that is delivered to a client connected to a Web Server is \_\_\_\_\_.

### TRUE OR FALSE

8. It is possible for a 'Domain Name' be unknown to an ISP's Internet Server.
9. A single computer having a permanent IP address can host multiple virtual domains on its hard disk drive.
10. TCP/IP is not a connection independent protocol.

## 2. INTRODUCTION TO HTML

The term **Knowledge is Power** really takes on a special dimension in software. This is one area where information plays a vital and dynamic role. Tried and tested methods of providing information to those who need it, like printed documents bound together in the form of manuals, books and so on, does not really work for software information. This is because this information is very dynamic and it keeps changing and re-changing all the time.

A change in information requires the contents of books, manuals, and so on, to change. This will in turn require complete reprinting of the books and manuals. Reprinting always results in time and cost overheads. Reprinting also results in out dated books and manuals being held in inventory, unable to be used.

To resolve this problem, information needs be stored in such a manner that whenever the information changes, these changes can be incorporated with the least cost and time. A tried and tested method that allows this is to store the information in the form of computer based files. These files could be stored at a central location. Once stored at a central location the files can be accessed when required for reference.

Since these files are stored at a central location on a computer, their access will also require a computer and some sort of network that connects these two computers together.

Thus to provide information to the users:

- Files holding information must be created.
- These files must be stored at a central location on a computer.
- When required the users should be allowed to access these files using their desktop computers.
- A network link must be established between the desktop computer and the computer serving information at a central location.

This introduces **Client / Server** terminology wherein

- The desktop computer requesting for information is termed as the **Client**.
- The computer serving information from a central location is termed as the **Server**.

### INFORMATION FILES CREATION

If information has to be stored on a central computer, it must be created first. While being created, information can only be stored in the form of files on the computer. These files are created using special software programs or programming environments.

Files that travel across the largest network in the world, the Internet, and carry information from a **Server** to a **Client** that requested them are called **Web Pages**. The individual who develops these web pages is called **Web Developer**.

### WEB SERVER

Web Pages are created using HTML syntax. These pages must be organized and stored at a central computer.

The organization of web pages into directories and files stored on the HDD of a central computer is called **Web Site** creation.

Computers, which store web pages in the form of directories and files and provide these files to be read, are called **Servers**. They act like service providers that service the need for information.

The Server Computer runs special software called **Web Server** software that allows:

- Web Site Management.
- Accept a client's request for information.
- Respond to a client's request by providing the page with the required information.

Some of the most popular software, which Servers run to allow them to respond to client request for information, is Internet Information Server (IIS), Apache Web Server, Netscape Server, and Microsoft Personal Web Server.

Web Server Software stores and manages web pages. When required, the Web Server accepts requests for these Web Pages, retrieves these web pages from its HDD, and sends the page back to the client who requested for it.

## WEB CLIENT / BROWSER

To access information stored in the form of web pages, users must connect to a Web Server. Once connected, an interface that displays the contents of the web page is required.

Computers that offer the facility to read information stored in web pages are called **Web Clients**.

Web Clients run special software called a **Browser** that allows them to:

- Connect to an appropriate Server.
- Query the Server for the information to be read.
- Provides an interface to read the information returned by the Server.

Some of the most popular Browser software that clients run to allow them to query Web Servers for information is Netscape Communicator, Internet Explorer.

### Understanding How A Browser Communicates With A Web Server

As seen earlier, a Web Server is responsible for sending web pages to a Browser on a client. When a Browser communicates with a Web Server, it results into a four-step HTTP transaction.

As seen in diagram 2.1, a client's Browser retrieves a web page from the server and displays the web page in the Browser. The communication steps between the client and the server can be summarized as follows:

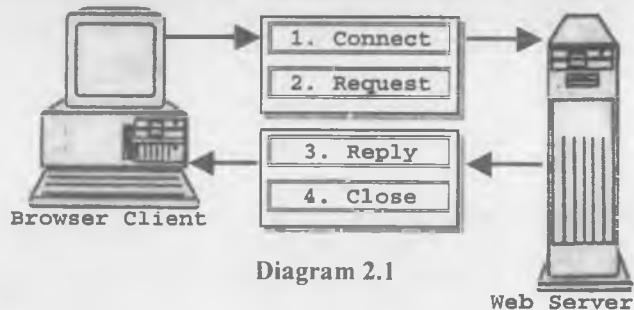


Diagram 2.1

### Establish Connection

Before a client and server can exchange information, they must first establish a connection. TCP/IP is used to let computers establish a link between a Web Server and a Web Browser over the Internet.

To communicate with the Web Server, the client machine must be given the IP address of the server along with the sub protocol that must be used i.e. HTTP, FTP etc. The client browser will attempt to locate the server based on the IP address supplied and establish a connection.

A Web Server supports multiple protocols. For example, a Web Server may support two protocols viz. HTTP, FTP.

In such cases, each of the protocols (like HTTP, FTP) can be accessed by specifying protocol name and a specific "port number". Common protocols like HTTP, FTP etc have "well known" port numbers. For example the HTTP protocol by default works on port number 80. Similarly the FTP protocol by default works on port number 21.

If the protocols are configured on default port numbers, the connection to a Web Server can be established by:

Protocol://IP address

For example, if the IP address for the SCT Server is 131.100.2.107 and communication must be established using HTTP then any client attempting to connect would have to pass the IP address as: - http://131.100.2.107 to the ISP whose gateway is being used to access the Internet.

Protocols can be configured to run on port numbers other than the default port number. Valid values are 0 to 9999. If a protocol is configured to run on any other port number then the client when trying to establish a communication link with the Web Server must specify the port number along with the protocol name and the IP address.

Thus the complete syntax to access and connect to any server would be:

Protocol://servername:port number.

### Client Issues A Request And Server Sends A Response

As seen earlier, each Web Server controls a web site. From amongst the collection of several web pages (i.e., the Web site), one page is treated as a 'Default Web Document'.

When a Browser connects to a Web Server using an appropriate protocol name, IP address and port number, and the Web Server treats this connection to be a request for the 'Default Web Document'. The Web Server then dispatches the 'Default Web Document' to the client who connected.

If the client requires viewing any other web page then the client can specify the web page name (*if known*) along with the connection information. Thus the complete connection and web page information will now be specified as:

Protocol://servername:port number/web page name.

A web page, apart from text and HTML tags, can also include references to external objects like GIF's, JPEG's, Audio files, Video files and so on.

Thus the request for a web page can be two-fold:

- The web page itself.
- The request for the objects referenced by the web page i.e. GIF's, JPEG's, Audio Files, Video Files, Executable programs and so on.

When an appropriately structured, HTTP request, is received from a Browser, the Web Server will try to locate the web page requested. If the Web Page exists, the server responds by providing the page to the browser.

If the Web Page is not found, an appropriate Error Message is sent as a response to the browser request. After sending the web page the Web Server resets the connection with the browser.

After receiving the Web page, the Browser will identify the external objects that are specified in the web page (*if any*) and request the Server to deliver the external objects.

Such a request will result in a connection being re-established with the Web Server and a request being sent by the browser. The Web Server responds to the request being sent by the browser.



## Server Terminates The Connection

It is the Server's responsibility to terminate the TCP/IP connection with the Browser after it responds to the Browser's request. However, both the Browser and the Web Server must manage an unexpected closing of a connection as well. In other words, if the user clicks on the Browser's STOP button, the Browser must close the connection.

Also a computer crash by either a Browser or a Web Server must be recognized by the surviving computer, which, in turn, will close the connection.

## HYPER TEXT MARKUP LANGUAGE (HTML)

The language used to develop web pages is called HyperText Markup Language (HTML). HTML is the language interpreted by a Browser. Web Pages are also called HTML documents. HTML is a set of special codes that can be embedded in text to add formatting and linking information. HTML is specified as TAGS in an HTML document (*i.e. the Web page*).

### HTML Tags

Tags are instructions that are embedded directly into the text of the document. An HTML tag is a signal to a browser that it should do something other than just throw text up on the screen. By convention all HTML tags begin with an open angle bracket (<) and end with a close angle bracket (>).

HTML tags can be of two types:

#### Paired Tags

A tag is said to be a *paired* tag if it, along with a companion tag, flanks the text. For example the <B> tag is a paired tag. The <B> tag with its companion tag </B> causes the text contained between them to be rendered in *bold*. The effect of other paired tags is applied only to the text they contain.

In paired tags, the *first* tag (<B>) is often called the *opening tag* and the *second* tag (</B>) is called the *closing tag*.

The opening tag activates the effect and the closing tag turns the effect off.

#### Singular Tags

The second type of tag is the *singular* or *stand-alone* tag. A *stand-alone* tag does not have a companion tag. For example <BR> tag will insert a line break. This tag does not require any companion tag.

### Note



---

Some HTML tags require additional information to be supplied to them. For instance, when a picture is placed on the screen, information like the height and width of the picture can be specified.

Additional information supplied to an HTML tag is known as *Attributes* of a tag. Attributes are written immediately following the tag, separated by a *space*. Multiple attributes can be associated with a tag, also separated by a *space*.

---

The skill in creating an HTML (web) page would lie in knowing the functionality of all HTML tags and tag attributes where applicable. This skill can then be used to format large quantities of textual information as required and have this ready to use on a web site.

The simplest and quickest way to know the functionality of HTML tags would be to learn a few tags (*and their attributes where applicable*) and immediately use these tags in formatting textual information to be used in a web page as part of a web site. Using this technique the HTML tag, its attributes where applicable, and its functionality will immediately become apparent.

Our approach to learning HTML is to create a series of web pages for a fictitious company to be used on its web site. Textual information will be viewed in a browser **without** the HTML tags in place and subsequently **with** the HTML tags in place. The difference in presentation quality of the textual information will be very apparent.

### FOCUS 1:

Delta Engineering Pvt. Ltd. is a Company, dealing with manufacture of products like Barbed Wire, Barbed Tapes and so on. This Company is interested in publicity with the widest possible reach and modest investment. The least expensive approach is through the Web! Delta is therefore looking at preparing a *Web Site* to let the World know about itself and main lines of business.

Consider the preparation of a web site for this engineering company. The information being presented on the site is categorized into:

- The Company Profile
- The Products manufactured in the Company

The Products manufactured by the Company primarily include:

- Barbed Wire
- Barbed Tape
- Animal Fencing

Information in all these areas needs to be provided to anyone who requires it. The information provided is segregated into four major heads:

- The Company Profile
- Barbed Wire
  - Line Wire
  - Barbed Wire
  - Carrying Handles
- Barbed Tape
  - Short Blade Barbed Tape (SBBT)
  - Medium Blade Barbed Tape (MBBT)
  - Long Blade Barbed Tape (LBBT)
- Animal Fencing

The creation of the textual content of the Web Site is done in a simple ASCII editor such as **Notepad** and saved as *filename.html* file. The file will contain the following textual material created as an ASCII file and saved as indicated above.

After creating this text file, view its contents in a browser, either Internet Explorer or Netscape navigator. The browser used for page testing purposes while creating this book has always been Netscape navigator.

After the text file created has been viewed in a browser a few HTML tags will be added to the material and the file will be re-viewed to register the changes in its presentation. This is the standard learning technique used with all the HTML tags and their attributes where applicable.

DELTA ENGINEERING PVT. LTD.

---

PROFILE

---

Delta Engineering Pvt. Ltd. is a specialist manufacturer of wire and wire products. DEPL established in Mumbai, India is ideally located for shipments to any part of the world. The main items of manufacture, in collaboration with Gulf Fencing Industry (GFI) include protector gabions and protector fencing systems.

Protector fencing systems offer a wide range of solutions for all security problems on any terrain and under extreme climatic conditions. The protector fencing system has been used extensively in Europe, America and the Far East, and this has helped formulate a package especially suited to Middle East requirements.

DEPL is equipped to offer comprehensive packages including design, material and installation.

Products strictly adhere to BS, ASTM and DIN international specifications. Quality control is guaranteed by independent laboratory test certificates from India.

Quality control is implemented without sacrificing economy and efficiency. DEPL is dedicated to technical services and problem solving.

DEPL can provide planning support by offering design, technical specifications, drawings, foundation plans and installation instructions, together with a personalized service. The company's technical sales engineers keep in constant touch with all clients. Utmost importance is given to optimum design, versatility, durability and economy backed by the DEPL guarantee for work undertaken.

Please forward any inquiries to [enq\\_depl@bom2.vsnl.net.in](mailto:enq_depl@bom2.vsnl.net.in)

DELTA ENGINEERING PVT. LTD.  
502, 5<sup>th</sup> Floor, Tejas Building  
Andheri (W), Mumbai  
INDIA  
Telephone : 91-022-8210050

---

BARBED WIRE

---

Concertina Barbed Wire in roll form is used in high security areas to deter trespassing men and animals. The effectiveness of this material is proved as it has been in use for more than 75 years during war and peace.

Toshkent Axborot Texnologiyalari Universitet  
11.2.765  
Axborot Resurs Markazi

**Line Wire:**

This is made up of 3.05 mm diameter high carbon steel wire, with a tensile strength of 170 to 180 kg/mm<sup>2</sup>. The wire is drawn and dressed in such a manner that the coils formed will fall naturally into the specified diameter without forming a figure eight. Line wire is heavy hot dipped galvanized with minimum zinc coating of 185 gm/mm<sup>2</sup>.

**Barbed Wire:**

This is 2.00mm diameter bright mild steel wire with a tensile strength of 38 to 55kg/mm<sup>2</sup> conforming to BS 1052. The wire is hot dipped galvanized with a minimum zinc coating of 20 to 50 gm/mm<sup>2</sup>. The barbs are formed with four points. Spacing between the centers of the barbs is every 70mm along the length of the wire. The barbs are firmly secured by indentations made on the wire.

**Carrying Handles:**

Handles are made of mild steel wire of 3.55mm diameter and are attached to the outer turn of the coil on each side.

-----  
-----  
**BARBED TAPES**  
-----  
-----

Barbed Tapes are used as psychological and physical deterrent against intrusion by personnel and animals. Barbed tape barrier systems are more vicious and difficult to tamper with, providing superior perimeter security. Some of the users are military installations, nuclear energy sites, maximum-security prisons, various petroleum installations, tank farms and other important industrial facilities.

**Short Blade Barbed Tape (SBBT):**

To give the maximum tensile strength at the time of the breach hard drawn steel cord is used (ES1). For camouflage purposes, a coaltar coating can be applied over the entire surface (ES-1), and to attain the high degree of rust resistance, the core wire is made of galvanized steel (ES-2). Any of the above three options are available to meet your specific requirements. There are several variations in spiral diameter in accordance with the needs of the customers.

**Medium Blade Barbed Tape (MBBT):**

The specification of the "ivory" tape is similar to those of ES types. However the blades are sharper and have a greater pricking capacity. This can be used for both temporary and permanent security. The life of this tape is more than three times longer than those of ES types. Therefore, "ivory" tape is a more effective choice in terrain, which has heavy rain or in coastal areas.

Long Blade Barbed Tape (LBBT):

This is the most effective psychological and physical deterrent, ever made as a barrier obstacle. It is available in stainless steel (SUS 430) and the core wire can be fabricated from either galvanized carbon steel or stainless steel.

---

## ANIMAL FENCING

---

DEPL's animal fencing system is mainly used as an anti-intrusion barrier against any farm and other animals.

The most common use is on highways where vast distances are covered at a very economical cost.

Animal fencing can also be used to enclose areas such as farms, forest areas and national parks, where security is not crucial.

This system is easy to install and can be erected in a comparatively short time.

The above textual material can be typed into a file using any ASCII text editor and saved as an HTML file. This HTML file can then be opened in a Web Browser. This file when opened in the Browser will display the output as shown in diagram 2.2.

### Output For Focus 1:

The output in diagram 2.2 lacks visual appeal. To increase the presentability of this textual material, it should be well formatted. HTML has a series of formatting tags. HTML tags along with syntax on how to use them are described in the following pages.

DELTA ENGINEERING PVT LTD -----

#### ----- PROFILE -----

Delta Engineering Pvt. Ltd is a specialist manufacturer of wire and wire products. DEPL established in Mumbai, India is ideally located for shipments to any part of the world. The main items of manufacture, in collaboration with Gulf Fencing Industry (GFI) include protector gabions and protector fencing systems. Protector fencing systems offer a wide range of solutions for all security problems on any terrain and under extreme climatic conditions. The protector fencing system has been used extensively in Europe, America and the Far East, and this has helped formulate a package especially suited to Middle East requirements. DEPL is equipped to offer comprehensive packages including design, material and installation. Products strictly adhere to BS, ASTM and DIN international specifications. Quality control is guaranteed by independent laboratory test certificates from India. Quality control is implemented without sacrificing economy and efficiency. DEPL is dedicated to technical services and problem solving. DEPL can provide planning support by offering design, technical specifications, drawings, foundation plans and installation instructions, together with a personalized service. The company's technical sales engineers keep in constant touch with all clients. Utmost importance is given to optimum design, versatility, durability and economy backed by the DEPL guarantee for work undertaken. Please forward any inquiries to enq\_depl@bom2.vsnl.net in DELTA ENGINEERING PVT LTD 502.

Diagram 2.2

## COMMONLY USED HTML COMMANDS

### The Structure Of An HTML program

Every HTML program has a rigid structure. The entire web page is enclosed within <HTML> </HTML> tags. Within these tags two distinct sections are created using the <HEAD> </HEAD> tags and the <BODY> </BODY> tags. These sections are described below.

### Document Head

Information placed in this section is essential to the inner workings of the document and has nothing to do with the content of the document. With the exception of information contained within the <TITLE> </TITLE> tags, all information placed within the <HEAD> </HEAD> tags is not displayed in the browser. The HTML tags used to indicate the start and end of the head section are:

```
<HEAD><TITLE> ... </TITLE></HEAD>
```

## Document Body

The tags used to indicate the start and end of the main body of textual information are:

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

Page defaults like background color, text color, font size, font weight and so on can be specified as *attributes* of the <BODY> tag. The attributes that the <BODY> tag takes are:

<b>BGCOLOR</b>	Changes the default background color to whatever color is specified with this tag. The user can specify a color by name or its equivalent hexadecimal number.
<b>BACKGROUND</b>	Specifies the name of the Gif file that will be used as the background of the document. This Gif tiles up across the page to give a background.
<b>TEXT</b>	Changes the body text color from its default value to the color specified with this attribute.

Table 2.1

### Example:

```
<BODY BACKGROUND = "starfield.gif" TEXT= red>
```

### Note



The .gif file "starfield.gif" should be present in the current working directory. If not, a Relative path should be specified to where the .gif file exists.

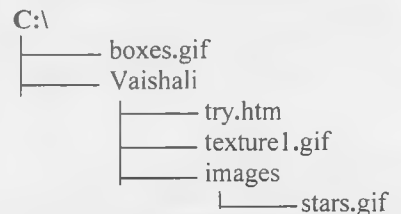
### Tip



#### Specifying a Relative File Path:

Wherever a file name needs to be specified (for instance, a .gif file needs to be specified with the *BACKGROUND* attribute), the specified file must be present in the current working directory. If the file is not in the current working directory, the specified file must include the file path. The file path can be specified relative to the current directory. This is because, by default, the Browser searches for the file only within the current directory.

For example, consider the following directory structure.



Consider working with the file *try.htm*. To use all the different .gif files as backgrounds, the tag specifications will change as follows:

#### Use of the file boxes.gif

```
<BODY BACKGROUND = "../boxes.gif">
```

#### Use of the file texture1.gif

```
<BODY BACKGROUND = "texture1.gif">
```

#### Use of the file stars.gif

```
<BODY BACKGROUND = "images/stars.gif">
```

## TITLES AND FOOTERS

### Title

A web page could have a title that describes what the page is about without being too wordy. This can be achieved by using the TITLE tag. Text included between the <TITLE>...</TITLE> tag shows up in the title bar of the browser window.

```
<TITLE> ... </TITLE>
```

### Footer

Just as a title can be placed in the title bar of the browser window, certain information is commonly placed at the foot of the web page. Copyright information, contact details of the creator of the Web page and so on are the type of information traditionally placed at the foot of the web page. The HTML tags are:

```
<ADDRESS> ... </ADDRESS>
```

This tag should ideally be placed *immediately after* the last line of the textual material of the web page. However, it could also be placed anywhere in the body of the document. The text typed within these tags always appears in *Italics*.

### **Example:**

```
<HEAD><TITLE>This is the title</TITLE></HEAD>
<BODY>
...
<ADDRESS>This is the footer </ADDRESS>
</BODY>
```

## TEXT FORMATTING

### Paragraph breaks

A blank line always separates paragraphs in textual material. The tag that provides this functionality is <P>. On encountering this tag the browser, moves onto a new line, skipping one line between the previous line and the new line.

### **Example:**

A channel is a dedicated path for data to flow along. It doesn't bother too much about error correction on its own, merely reporting to the processor that the transfer failed. So a channel can be very fast. <P> Fiber channel has the above attributes along with those of a network. It lets you transfer data from a source buffer to another buffer at journey's end. It does not care for the contents or the data format.

### **Output:**

A channel is a dedicated path for data to flow along. It doesn't bother too much about error correction on its own, merely reporting to the processor that the transfer failed. So a channel can be very fast.

Fiber channel has the above attributes along with those of a network. It lets you transfer data from a source buffer to another buffer at journey's end. It does not care for the contents or the data format.

## Line Breaks

When text needs to start from a new line and not continue on the same line (*without skipping a blank line*), the **<BR>** tag should be used. This tag simply jumps to the start of the next line.

### Example:

Silicon Chip Technologies, **<BR>** A/5, Jay Apartments, **<BR>**  
Vile Parle (East), **<BR>** Mumbai - 400057.

### Output:

Silicon Chip Technologies,  
A/5, Jay Apartments,  
Vile Parle (East),  
Mumbai - 400057.

## *Note*



Browsers **ignore** multiple consecutive **<P>** tags, but **recognize** multiple consecutive **<BR>** tags.

### FOCUS 2:

Using appropriate text formatting and break tags to improve the presentability of the DEPL Web Site.

### Code Listing:

```
<HTML>
```

```
<HEAD><TITLE>Delta Engineering Pvt. Ltd.</TITLE></HEAD>
```

```
<BODY Background="images/pinkwhit.gif">DELTA ENGINEERING PVT. LTD.<P>
```

```
-----<BR>  
PROFILE <BR>
```

```
-----<BR>
```

Delta Engineering Pvt. Ltd. is a specialist manufacturer of wire and wire products. DEPL established in Mumbai, India is ideally located for shipments to any part of the world. The main items of manufacture, in collaboration with Gulf Fencing Industry (GFI) include protector gabions and protector fencing systems. **<P>** Protector fencing systems offer a wide range of solutions to all security problems on any terrain and under extreme climatic conditions. Protector fencing system has been used extensively in Europe, America and the Far East, and this has helped formulate a package especially suited to Middle East requirements. **<P>** DEPL is equipped to offer comprehensive packages including design, material and installation. **<P>** Products strictly adhere to BS, ASTM and DIN international specifications. Quality control is guaranteed by independent laboratory test certificates from India. **<P>** Quality control is implemented without sacrificing economy and efficiency. DEPL is dedicated to technical services and problem solving. **<P>** DEPL can provide planning support by offering design, technical specifications, drawings, foundation plans and installation instructions, together with personalized service. The company's technical sales engineers keep in constant touch with all clients. Utmost importance is given to optimum design, versatility, durability and economy backed by the DEPL guarantee for work undertaken. **<P>** Please forward any inquiries to [enq\\_depl@bom2.vsnl.net.in](mailto:enq_depl@bom2.vsnl.net.in) **<P>**

```
DELTA ENGINEERING PVT. LTD. <BR>
```

```
502, 5th Floor, Tejas Building, <BR> Andheri (W), Mumbai <BR> India <BR>
```

```
Telephone : 91-022-8210050 <BR>
```



---

**BARBED WIRE**

Concertina Barbed Wire in roll form is used in high security areas to deter trespassing men and animals. The effectiveness of this material is proved as it has been in use for more than 75 years during war and peace.

**Line Wire:** This is made up of 3.05mm diameter high carbon steel wire, with a tensile strength of 170 to 180 kg/mm<sup>2</sup>. The wire is drawn and dressed in such a manner that the coils formed will fall naturally into the specified diameter without forming a figure eight. Line wire is heavy hot dipped galvanized with minimum zinc coating of 185 gm/mm<sup>2</sup>.

**Barbed Wire:** This is 2.00mm diameter bright mild steel wire with a tensile strength of 38 to 55kg/mm<sup>2</sup> conforming to BS 1052. The wire is hot dipped galvanized with a minimum zinc coating of 20 to 50 gm/mm<sup>2</sup>. The barbs are formed with four points. Spacing between the centers of the barbs is every 70mm along the length of the wire. The barbs are firmly secured by indentations made on the wire.

**Carrying Handles:** Handles are made of mild steel wire of 3.55mm diameter and are attached to the outer turn of the coil on each side.

---

**BARBED TAPES**

Barbed Tapes are used as psychological and physical deterrent against intrusion by personnel and animals. Barbed tape barrier systems are more vicious and difficult to tamper with, providing superior perimeter security. Some of the users are military installations, nuclear energy sites, maximum-security prisons, various petroleum installations, tank farms and other important industrial facilities.

**Short Blade Barbed Tape (SBBT):** To give the maximum tensile strength at the time of the breach hard drawn steel cord is used (ES1). For camouflage purposes, a coaltar coating can be applied over the entire surface (ES-1), and to attain the high degree of rust resistance, the core wire is made of galvanized steel (ES-2). Any of the above three options are available to meet your specific requirements.

**Medium Blade Barbed Tape (MBBT):** The specification of the "ivory" tape is similar to those of ES types. However the blades are sharper and have a greater pricking capacity. This can be used for both temporary and permanent security. The life of this tape is more than three times longer than those of ES types. Therefore, "ivory" tape is a more effective choice in terrain, which, has heavy rain or in coastal areas.

**Long Blade Barbed Tape (LBBT):** This is the most effective psychological and physical deterrent, ever made as a barrier obstacle. It is available in authentic stainless steel (SUS 430) and the core wire can be fabricated from either galvanized carbon steel or stainless steel.

---

**ANIMAL FENCING**

DEPL's animal fencing system is mainly used as an anti-intrusion barrier against any farm and other animals. The most common use is on highways where vast distances are covered at a very economical cost. Animal fencing can also be used to enclose areas such as farms, forest areas and national parks, where security is not crucial. This system is easy to install and can be erected in a comparatively short time.

</BODY>  
</HTML>

**Output For Focus 2:**

When the web page is viewed in the browser the output shows paragraphs distinctly, but still is not visually attractive. For instance, headings like Profile, Products, Barbed Wire and so on are emphasized by means of two dashed lines, one above and one below the Heading. This is not a visually pleasing way of laying emphasis on headers. Headers should preferably be displayed in Bold, Italics and so on. Also, simple dashed lines drawn do not have aesthetic appeal.

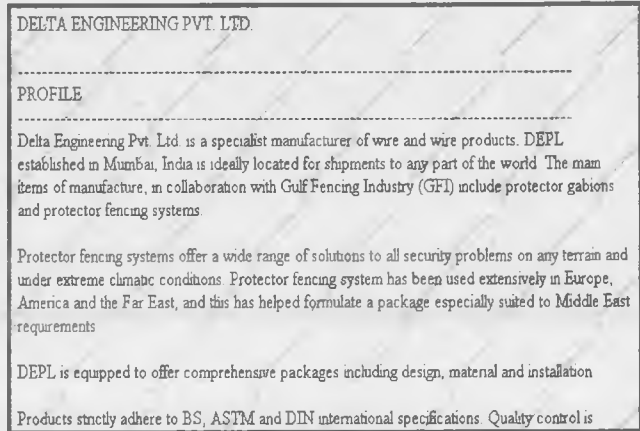


Diagram 2.3

**EMPHASIZING MATERIAL IN A WEB PAGE**

Document pages are usually divided into sections and subsections (i.e. *pages could have headings and sub headings*), which need to be emphasized. HTML provides certain HEADING STYLES and HORIZONTAL RULES, which helps break text into logical sections with visual appeal.

**Heading Styles**

HTML supports six different levels of headings. The highest-level header format is <H1> and the lowest level is <H6>. All the styles appear in BOLDFACE and the size of the heading depends on the level chosen, i.e. <H1> to <H6>

**Example:**

<H3>The early years</H3>

**Output:**

**The early years**

**Note**

As the number next to <H> (1, 2, ...) increases, the font size actually decreases.

**Drawing Lines**

The tag <HR> draws lines and horizontal rules. This tag draws a horizontal line across the whole page, wherever specified. The attributes to the <HR> tag are:

Attributes	Description
ALIGN	Aligns the line on the Browser screen, which is by default, aligned to the center of the screen. ALIGN = LEFT will align the line to the left of the screen ALIGN = RIGHT will align the line to the right of the screen ALIGN = CENTER will align the line to the center of the screen
SIZE	Changes the size of the rule.
WIDTH	Sets the width of the rule. It can be set to a fixed number of pixels, or to a percentage of the available screen width.

Table 2.2

**Example:**

Welcome to our Web site.  
 <HR ALIGN= LEFT WIDTH=10 SIZE=4>

**Output:****TEXT STYLES****Bold**

Displays text in BOLDFACE style. The tags used are <B>...</B>

**Example:**

<B>Welcome to our home page!</B>

**Output:**

**Welcome to our home page!**

**Italics**

Displays text in ITALICS. The tags used are between <I>...</I>

**Example:**

<I> Welcome to our home page! </I>

**Output:**

*Welcome to our home page!*

**Underline**

Displays text as UNDERLINED, The tags used are <U>...</U>

**Example:**

<U> Welcome to our home page! </U>

**Output:**

Welcome to our home page!

**FOCUS 3:**

Emphasize the headings and sub-headings displayed on the DEPL web page.

**Code Listing:**

<HTML>

<HEAD><TITLE>Delta Engineering Pvt. Ltd.</TITLE></HEAD>

<BODY Background="images/pinkwhit.gif"><H2><I>Delta Engineering Pvt. Ltd.</I></H2>

<H3>PROFILE</H3>Delta Engineering Pvt. Ltd. is a specialist manufacturer of wire and wire products. DEPL established in Mumbai, India is ideally located for shipments to any part of the world. The main items of manufacture, in collaboration with Gulf Fencing Industry (GFI) include protector gabions and protector fencing systems. <P>Protector fencing systems offer a wide range of solutions to all security problems on any terrain and under extreme climatic conditions. The protector fencing system has been used extensively in Europe, America and the Far East, and this has helped formulate a package especially suited to Middle East requirements. <P> DEPL is equipped to offer comprehensive packages including design, material and installation.

<P>Products strictly adhere to BS, ASTM and DIN international specifications. Quality control is guaranteed by independent laboratory test certificates from India. <P> Quality control is implemented without sacrificing economy and efficiency. DEPL is dedicated to technical services and problem solving. <P> DEPL is following quality methods and is accredited with ISO 9002. <P> DEPL can provide planning support by offering design, technical specifications, drawings, foundation plans and installation instructions, together with a personalized service. The company's technical sales engineers keep in constant touch with all clients. Utmost importance is given to optimum design, versatility, durability and economy backed by the DEPL guarantee for work undertaken. <P> Please forward any inquiries to enq\_depl@bom2.vsnl.net.in  
<P><I>DELTA ENGINEERING PVT. LTD. <BR> 502, 5<sup>th</sup> Floor, Tejas Building <BR> Andheri (W), Mumbai <BR> India <BR> Telephone: 91-022-8210050</I><P><HR>

<H3>BARBED WIRE</H3> Concertina Barbed Wire in roll form is used in high security areas to deter trespassing men and animals. The effectiveness of this material is proved as it has been in use for more than 75 years during war and peace.

<P><B><I> Line Wire: </I></B><BR> This is made up of 3.05mm diameter high carbon steel wire, with a tensile strength of 170 to 180 kg/mm<sup>2</sup>. The wire is drawn and dressed in such a manner that the coils formed will fall naturally into the specified diameter without forming a figure eight. Line wire is heavy hot dipped galvanized with minimum zinc coating of 185 gm/mm<sup>2</sup>.

<P><B><I> Barbed Wire: </I></B> <BR> This is 2.00mm diameter bright mild steel wire with a tensile strength of 38 to 55kg/mm<sup>2</sup> conforming to BS 1052. The wire is hot dipped galvanized with a minimum zinc coating of 20 to 50 gm/mm<sup>2</sup>. The barbs are formed with four points. Spacing between the centers of the barbs is every 70mm along the length of the wire. The barbs are firmly secured by indentations made on the wire, so that the barbs do not rotate or slide along the wire. The four points of the barbs are formed at the right angles to one another and project outwards approximately 12mm from the center of the wire.

<P><B><I> Carrying Handles: </I></B> <BR> Handles are made of mild steel wire of 3.55mm diameter and are attached to the outer turn of the coil on each side.<P><HR>

<H3>BARBED TAPES </H3> Barbed Tapes are used as psychological and physical deterrent against intrusion by personnel and animals. Barbed tape barrier systems are more vicious and difficult to tamper with, providing superior perimeter security. Some of the users are military installations, nuclear energy sites, maximum-security prisons, various petroleum installations, tank farms and other important industrial facilities.

<P><B><I> Short Blade Barbed Tape (SBBT): </I></B><BR> To give the maximum tensile strength at the time of the breach hard drawn steel cord is used (ES1). For camouflage purposes, a coaltar coating can be applied over the entire surface (ES-1), and to attain the high degree of rust resistance, the core wire is made of galvanized steel (ES-2). Any of the above three options are available to meet your specific requirements.

<P><B><I> Medium Blade Barbed Tape (MBBT): </I></B><BR> The specification of the "ivory" tape is similar to those of ES types. The blades are sharper and have a greater pricking capacity. The life of this tape is more than three times longer than those of ES types. Therefore, "ivory" tape is a more effective choice in terrain which has heavy rain or in coastal areas. The diameter will be similar to ES type.

<P><B><I> Long Blade Barbed Tape (LBBT): </I></B><BR> This is the most effective psychological and physical deterrent, ever made as a barrier obstacle. It is available in authentic stainless steel (SUS 430) and the core wire can be fabricated from either galvanized carbon steel or stainless steel.<P><HR>

**<H3>ANIMAL FENCING</H3>** DEPL's animal fencing system is mainly used as an anti-intrusion barrier against any farm and other animals. **<P>** The most common use is on highways where vast distances are covered at a very economical cost. **<P>** imal fencing can also be used to enclose areas such as farms, forest areas and national parks, where security is not crucial. **<P>** This system is easy to install and can be erected in a comparatively short time. **<HR>**

**</BODY>**

**</HTML>**

**Output For Focus 3:**

<p><i>Delta Engineering Pvt. Ltd.</i></p> <p><b>PROFILE</b></p> <p>Delta Engineering Pvt. Ltd. is a specialist manufacturer of wire and wire products. DEPL established in Mumbai, India is ideally located for shipments to any part of the world. The main items of manufacture, in collaboration with Gulf Fencing Industry (GFI) include protector gabions and protector fencing systems.</p> <p>Protector fencing systems offer a wide range of solutions to all security problems on any terrain and under extreme climatic conditions. The protector fencing system has been used extensively in Europe, America and the Far East, and this has helped formulate a package especially suited to Middle East requirements.</p> <p>DEPL is equipped to offer comprehensive packages including design, material and installation.</p> <p>Products strictly adhere to BS, ASTM and DIN international specifications. Quality control is</p>	<p><b>BARBED WIRE</b></p> <p>Concertina Barbed Wire in roll form is used in high security areas to deter trespassing men and animals. The effectiveness of this material is proved as it has been in use for more than 75 years during war and peace.</p> <p><i>Line Wire:</i></p> <p>This is made up of 3.05mm diameter high carbon steel wire, with a tensile strength of 170 to 180 kg/mm<sup>2</sup>. The wire is drawn and dressed in such a manner that the coils formed will fall naturally into the specified diameter without forming a figure eight. Line wire is heavy hot dipped galvanized with minimum zinc coating of 185 g/m<sup>2</sup>.</p> <p><i>Barbed Wire:</i></p> <p>This is 2.00mm diameter bright mild steel wire with a tensile strength of 38 to 55kg/mm<sup>2</sup> conforming to BS 1052. The wire is hot dipped galvanized with a minimum zinc coating of 20 to 50 g/m<sup>2</sup>. The barbs are formed with four points. Spacing between the centers of the barbs is every 70mm along the length of the wire. The barbs are firmly secured by indentations made on the wire, so that the barbs do not rotate or slide along the wire. The four points of the barbs are formed at the right</p>
--	--

**Diagram 2.4.1**

**Diagram 2.4.2**

When a text file is formatted as above is viewed in a browser what is seen is visually pleasing. The visual aesthetics of the web page can be further improved by introducing various Fonts, Colors, centering of Text and so on. The following sections illustrate how this Web page can be made visually, further attractive.

**OTHER TEXT EFFECTS**

**Centering (Text, Images etc.)**

**<CENTER>** . . . . . **</CENTER>** tags are used to center everything found between them – text, lists, images, rules, tables or any other page element.

**Example:**

**<CENTER>** Welcome to our home page! **</CENTER>**

**Output:**

Welcome to our home page!

**Spacing (Indenting Text)**

The tag used for inserting blank spaces in an HTML document is **<SPACER>**. Its attributes are:

<b>TYPE</b>	To specify whether space has to be left horizontally or vertically. <b>TYPE = "HORIZONTAL"</b> indicates that horizontal space has to be left. <b>TYPE = "VERTICAL"</b> indicates that vertical space has to be left.
<b>SIZE</b>	Indicates the amount of space to be left. Size accepts any integer.

**Table 2.3**

## Note



The SPACER command is understood **only** by the browser Netscape. There are several other techniques that can be used to introduce space in the document (understood by all Browsers). These will be covered in the later Chapters.

### Example:

```
Welcome to my site <BR>
<SPACER TYPE = "HORIZONTAL" SIZE = 90>Hope you enjoy it
```

### Output:

```
Welcome to my site
      Hope you enjoy it
```

### Controlling Font Size and Color

All text specified within the tags <FONT> and </FONT> will appear in the font, size and color as specified as attributes of the tag <FONT>. The attributes are:

<b>FACE</b>	Sets the font to the specified font name.
<b>SIZE</b>	Sets the size of the text. SIZE can take values between 1 and 7. The default size used is 3. SIZE can also be set relative to the default size. i.e. <b>SIZE = +x</b> , where x is any integer value and will add up to the default size. For instance, <b>SIZE = +3</b> will display a size of 6.
<b>COLOR</b>	Sets the color of the text. COLOR can be set to an English language color name or to a hexadecimal triplet.

Table 2.4

### Example:

```
<FONT Face="Comic Sans MS" Size = 6 Color = RED>Welcome to our home page!</FONT>
```

### Output:

```
Welcome to our home page!
```

## Caution



The FONTFACE attribute can take the name of any font supported by Windows i.e. the font should be present in the directory. C:\Windows\Font\.... . If the font used with this attribute is not available on the computer, the browser uses its own default font. Therefore, it is a good practice to use commonly used fonts, which have a high possibility of being present on every client computer.

### FOCUS 4:

One method of making a web site more readable and interesting, to be able to draw the viewer's attention to certain key areas in the page is by using different Fonts and Colors.

### Code Listing:

```
<HTML>
<HEAD><TITLE> Delta Engineering Pvt. Ltd. </TITLE></HEAD>
<BODY Background="images/pinkwhit.gif">
```

<FONT Face = "Brush Script MT" Size = 7 Color = "#008000"><I>Delta Engineering Pvt. Ltd. </I><HR></FONT>

<FONT Color = "#008000"><CENTER><H3>PROFILE</H3>Delta Engineering Pvt. Ltd. is a specialist manufacturer of wire and wire products. DEPL established in Mumbai, India is ideally located for shipments to any part of the world. The main items of manufacture, in collaboration with Gulf Fencing Industry (GFI) include protector gabions and protector fencing systems. <P> Protector fencing systems offer a wide range of solutions to all security problems on any terrain and under extreme climatic conditions. The protector fencing system has been used extensively in Europe, America and the Far East, and this has helped formulate a package especially suited to Middle East requirements. <P> DEPL is equipped to offer comprehensive packages including design, material and installation. <P> Products strictly adhere to BS, ASTM and DIN international specifications. Quality control is guaranteed by independent laboratory test certificates from India. <P> Quality control is implemented without sacrificing economy and efficiency. DEPL is dedicated to technical services and problem solving. <P> DEPL is following quality methods and is accredited with ISO 9002. <P> DEPL can provide planning support by offering design, technical specifications, drawings, foundation plans and installation instructions, together with a personalized service. The company's technical sales engineers keep in constant touch with all clients. Utmost importance is given to optimum design, versatility, durability and economy backed by the DEPL guarantee for work undertaken.</CENTER>

<P>Please forward any enquiries to enq\_depl@bom2.vsnl.net.in

<P><I><B>DELTA ENGINEERING PVT. LTD.</B><BR>502, 5th Floor, Tejas Building <BR>Andheri (W), Mumbai <BR>India <BR>Telephone : 91-022-8210050</I><P><HR>

<CENTER><H3>BARBED WIRE</H3></CENTER> Concertina Barbed Wire in roll form is used in high security areas to deter trespassing men and animals. The effectiveness of this material is proved as it has been in use for more than 75 years during war and peace.

<P><B><I>Line Wire:</I></B><BR>This is made up of 3.05mm diameter high carbon steel wire, with a tensile strength of 170 to 180 kg/mm<sup>2</sup>. The wire is drawn and dressed in such a manner that the coils formed will fall naturally into the specified diameter without forming a figure eight. Line wire is heavy hot dipped galvanized with minimum inc coating of 185 gm/mm<sup>2</sup>.

<P><B><I>Barbed Wire:</I></B><BR>This is 2.00mm diameter bright mild steel wire with a tensile strength of 38 to 55kg/mm<sup>2</sup> conforming to BS 1052. The wire is hot dipped galvanized with a minimum zinc coating of 20 to 50 gm/mm<sup>2</sup>. The barbs are formed with four points. Spacing between the centers of the barbs of every 70mm along the length of the wire. The barbs are firmly secured by indentations made on the wire, so that the barbs do not rotate or slide along the wire. The four points of the barbs are formed at the right angles to one another and project outwards approximately 12mm from the center of the wire.

<P><B><I>Carrying Handles:</I></B><BR>Handles are made of mild steel wire of 3.55mm diameter and are attached to the outer turn of the coil on each side. <P><HR>

<CENTER><H3>BARBED TAPES</H3></CENTER> Barbed Tapes are used as psychological and physical deterrent against intrusion by personnel and animals. Barbed tape barrier systems are more vicious and difficult to tamper with, providing superior perimeter security. Some of the users are military installations, nuclear energy sites, maximum-security prisons, various petroleum installations, tank farms and other important industrial facilities.

<P><B><I>Short Blade Barbed Tape (SBBT):</I></B><BR> To give the maximum tensile strength at the time of the breach hard drawn steel cord is used (ES1). For camouflage purposes, a coaltar coating can be applied over the entire surface (ES-1), and to attain the high degree of rust resistance, the core wire is made of galvanized steel (ES-2). Any of the above three options are available to meet your specific requirements.

<P><B><I>Medium Blade Barbed Tape (MBBT):</I></B><BR> The specification of the "ivory" tape is similar to those of ES types. The blades are sharper and have a greater pricking capacity. The life of this tape is more than three times longer than those of ES types. Therefore, "ivory" tape is a more effective choice in terrain which has heavy rain or in coastal areas. The diameter will be similar to ES type.

<P><B><I>Long Blade Barbed Tape (LBBT):</I></B><BR> This is the most effective psychological and physical deterrent, ever made as a barrier obstacle. It is available in authentic stainless steel (SUS 430) and the core wire can be fabricated from either galvanized carbon steel or stainless steel. <P><HR>

<CENTER><H3>ANIMAL FENCING</H3></CENTER> DEPL's animal fencing system is mainly used as an anti-intrusion barrier against any farm and other animals. <P> The most common use is on highways where vast distances are covered at a very economical cost. <P> Animal fencing can also be used to enclose areas such as farms, forest areas and national parks, where security is not crucial. <P> This system is easy to install and can be erected in a comparatively short time.<HR>

</FONT>

</BODY>

</HTML>

#### Output:

The visual aesthetics of this page may be improved upon by using common techniques like HTML Lists, Bulleted, Numbered, Unordered and Ordered and so on. The technique (syntax) of using HTML Lists is given in the following chapters.

*Delta Engineering Pvt. Ltd.*

---

**PROFILE**

Delta Engineering Pvt. Ltd. is a specialist manufacturer of wire and wire products. DEPL established in Mumbai, India is ideally located for shipments to any part of the world. The main items of manufacture, in collaboration with Gulf Fencing Industry (GFI) include protector gabions and protector fencing systems.

Protector fencing systems offer a wide range of solutions to all security problems on any terrain and under extreme climatic conditions. The protector fencing system has been used extensively in Europe, America and the Far East, and this has helped formulate a package especially suited to Middle East requirements.

DEPL is equipped to offer comprehensive packages including design, material and installation.

**Diagram 2.5**

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. HTML stands for \_\_\_\_\_.
2. Files that travel across the largest network in the world, the Internet, and carry information from \_\_\_\_\_ to a \_\_\_\_\_ that request them are called \_\_\_\_\_. The person who develops them is called a \_\_\_\_\_.
3. TCP/IP stands for \_\_\_\_\_.
4. An HTML document is divided into \_\_\_\_\_ and the \_\_\_\_\_ sections.
5. \_\_\_\_\_ tag starts text from a new line skipping one line in between.

### TRUE OR FALSE

6. Some of the most popular software, which Servers run to allow them to respond to client request for information, are Microsoft Internet Explorer, Netscape Navigator, Neo Planet etc.
7. HTML supports 6 different levels of headings.



8. HTML tags are of two types are Paired Tags and Singular Tags.
9. SIZE can take values between 1 & 7.
10. The .gif files are specified with BGCOLOR attribute.
11. The tag used for inserting spaces in HTML documents is <MARKER>.

## HANDS ON EXERCISES

1. For the screen shown in diagram 2.7 write the HTML code which makes use of:
  - The Italics tag
  - The Center tag
  - The Paragraph tag
  - The Break tag
  - The Font tag and its attributes

### Text Content

Beware of password hackers and crackers!

The threat of hackers and crackers is for real, and is alike for everything for everyone. If the security breach at the Babha Atomic Research Center last year by the hacker group milworm didn't convince you of their capabilities, then consider this: James Davis aka jdavis\_4, a resident of [www.geocities.com/SiliconValley/Bit/2483](http://www.geocities.com/SiliconValley/Bit/2483) homestead community on the Internet, might just be reading your e-mail in your mailbox right now, or using your account for some benign Internet surfing, or exchanging it with someone else on the Net for more such accounts!

Surprised? Or scared?

Whatever it is, jdavis\_4 claims in his home page Free VSNL passwords that he has got exactly 108 working passwords (and growing daily), 36 Satyam, 31 MTNL, 11 WimiNet and 23 online passwords.

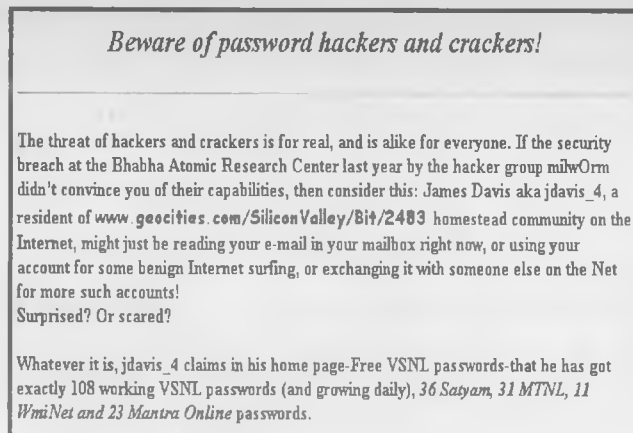


Diagram 2.6: Output of Hands on Exercise

### 3. LISTS

#### TYPES OF LISTS

##### Unordered List (Bullets)

An unordered list starts with the tag `<UL>` and ends with `</UL>`. Each list item starts with the tag `<LI>`. The attributes that can be specified with `<LI>` are

<b>TYPE</b>	Specifies the type of the bullet. <b>TYPE= FILLROUND</b> will give a solid round black bullet <b>TYPE= SQUARE</b> will give a solid square black bullet
-------------	---

Table 3.1

##### Example:

```
Some of these products include:  
<UL TYPE = FILLROUND>  
  <LI> Floppies  
  <LI> Hard Disks  
  <LI> Monitors  
</UL>
```

##### Output:

```
Some of these products include:  
• Floppies  
• Hard Disks  
• Monitors
```

##### Ordered Lists (Numbering)

An ordered list start with the tag `<OL>` and ends with `</OL>`. Each list items start with the tag `<LI>`. The attributes that can be specified with `<LI>` are:

<b>TYPE</b>	Controls the numbering scheme to be used. <b>TYPE = "1"</b> will give counting numbers ( 1, 2, . . . . . ) <b>TYPE = "A"</b> will give Uppercase letters ( A, B, . . . . . ) <b>TYPE = "a"</b> will give Lowercase letters ( a, b, . . . . . ) <b>TYPE = "I"</b> will give Uppercase Roman Numerals ( I, II, . . . . . ) <b>TYPE = "i"</b> will give Lowercase Roman Numerals ( i, ii, . . . . . )
<b>START</b>	Alters the numbering sequence. Can be set to any numeric value.
<b>VALUE</b>	Changes the numbering sequence in the middle of an ordered list. It is to be specified with the <code>&lt;LI&gt;</code> tag.

Table 3.2

##### Example:

```
Some of these products include:  
<OL TYPE = "1" START="3">  
  <LI> Floppies
```

```

<LI> Hard Disks
<LI> Monitors
</OL>

```

**Output:**

```

Some of these products include:
3. Floppies
4. Hard Disks
5. Monitors

```

**Definition Lists**

Definition list values appear within tags `<DL>` and `</DL>`. Definition lists consists of two parts:

<b>Definition term</b>	appears after the tag <code>&lt;DT&gt;</code>
<b>Definition description</b>	appears after the tag <code>&lt;DD&gt;</code>

Table 3.3

**Example:**

```

<DL>
  <DT> Keyboard
  <DD> An input device
  <DT> Printer
  <DD> An output device
</DL>

```

**Output:**

```

Keyboard
  An input device
Printer
  An output device

```

**FOCUS**

When a manufacturing company publishes its information on the Web it seeks to draw user attention to its (*list of*) products. Creating such a list can be achieved through the use of Bullets or Numbering. These techniques will be well applied to the *Barbed Wires* and *Barbed Tapes* lists on DEPL's web page to illustrate their use.

**Code Listing:**

```

<HTML>
  <HEAD><TITLE>Delta Engineering Pvt. Ltd.</TITLE></HEAD>
  <BODY Background="images/pinkwhit.gif">
    <FONT Face = "Brush Script MT" Size = 7 Color = "#008000"><I>Delta Engineering Pvt.
    Ltd.</I><HR></FONT>
    <FONT Color = "#008000"><CENTER><H3>PROFILE</H3>Delta Engineering Pvt. Ltd. is a
    specialist manufacturer of wire and wire products. DEPL established in Mumbai, India is ideally
    located for shipments to any part of the world. The main items of manufacture, in collaboration
    with Gulf Fencing Industry (GFI) include protector gabions and protector fencing
    systems.<P>Protector fencing systems offer a wide range of solutions to all security problems on
    any terrain and under extreme climatic conditions. The protector fencing system has been used
    extensively in Europe, America and the Far East, and this has helped formulate a package
    especially suited to Middle East requirements.

```

DEPL is equipped to offer comprehensive packages including design, material and installation. Products strictly adhere to BS, ASTM and DIN international specifications. Quality control is guaranteed by independent laboratory test certificates from India. Quality control is implemented without sacrificing economy and efficiency. DEPL is dedicated to technical services and problem solving. DEPL is following quality methods and is accredited with ISO 9002. DEPL can provide planning support by offering design, technical specifications, drawings, foundation plans and installation instructions, together with a personalized service. The company's technical sales engineers keep in constant touch with all clients. Utmost importance is given to optimum design, versatility, durability and economy backed by the DEPL guarantee for work undertaken.

Please forward any enquiries to [enq\\_depl@bom2.vsnl.net.in](mailto:enq_depl@bom2.vsnl.net.in)

**DELTA ENGINEERING PVT. LTD.**  
502, 5th Floor, Tejas Building  
Andheri (W), Mumbai India  
Telephone : 91-022-8210050

**BARBED WIRE** Concertina Barbed Wire in roll form is used in high security areas to deter trespassing men and animals. The effectiveness of this material is proved as it has been in use for more than 75 years during war and peace.

**Line Wire:** This is made up of 3.05mm diameter high carbon steel wire, with a tensile strength of 170 to 180 kg/mm<sup>2</sup>. The wire is drawn and dressed in such a manner that the coils formed will fall naturally into the specified diameter without forming a figure eight. Line wire is heavy hot dipped galvanized with minimum inc coating of 185 gm/mm<sup>2</sup>.

**Barbed Wire:** This is 2.00mm diameter bright mild steel wire with a tensile strength of 38 to 55kg/mm<sup>2</sup> conforming to BS 1052. The wire is hot dipped galvanized with a minimum zinc coating of 20 to 50 gm/mm<sup>2</sup>. The barbs are formed with four points. Spacing between the centers of the barbs of every 70mm along the length of the wire. The barbs are firmly secured by indentations made on the wire, so that the barbs do not rotate or slide along the wire. The four points of the barbs are formed at the right angles to one another and project outwards approximately 12mm from the center of the wire.

**Carrying Handles:** Handles are made of mild steel wire of 3.55mm diameter and are attached to the outer turn of the coil on each side.

**BARBED TAPES** Barbed Tapes are used as psychological and physical deterrent against intrusion by personnel and animals. Barbed tape barrier systems are more vicious and difficult to tamper with, providing superior perimeter security. Some of the users are military installations, nuclear energy sites, maximum-security prisons, various petroleum installations, tank farms and other important industrial facilities.

**Short Blade Barbed Tape (SBBT):**

To give the maximum tensile strength at the time of the breach hard drawn steel cord is used (ES1). For camouflage purposes, a coaltar coating can be applied over the entire surface (ES-1), and to attain the high degree of rust resistance, the core wire is made of galvanized steel (ES-2). Any of the above three options are available to meet your specific requirements.

**Medium Blade Barbed Tape (MBBT):**

The specification of the "ivory" tape is similar to those of ES types. The blades are sharper and have a greater pricking capacity. The life of this tape is more than three times longer than those of ES types. Therefore, "ivory" tape is a more effective choice in terrain which has heavy rain or in coastal areas. The diameter will be similar to ES type.

**Long Blade Barbed Tape (LBBT):**

This is the most effective psychological and physical deterrent, ever made as a barrier obstacle. It is available in authentic stainless steel (SUS 430) and the core wire can be fabricated from either galvanized carbon steel or stainless steel.

<CENTER><H3>ANIMAL FENCING</H3></CENTER>DEPL's animal fencing system is mainly used as an anti-intrusion barrier against any farm and other animals. <P> The most common use is on highways where vast distances are covered at a very economical cost. <P> Animal fencing can also be used to enclose areas such as farms, forest areas and national parks, where security is not crucial. <P> This system is easy to install and can be erected in a comparatively short time.<HR>  
</FONT>

</BODY>

</HTML>

### Output For Focus:

The HTML pages of DEPL's site are fairly well laid out, visually. However, it is a well established fact that a large document with vast textual content is not very interesting to the viewer. On the Web, such documents fail to attract viewers. Pictures or images added to web pages definitely serve to attract users.

At the same time, an image can be used very effectively to give information to a user, without having to read lengthy descriptions.

Fortunately, HTML provides techniques by which not only static, but also animated pictures can be placed on a web page. In fact the power of HTML lies in the fact that HTML supports adding Multimedia to the Web Pages.

How Pictures can be added to the Web Page is described in the following chapters.

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. TYPE = \_\_\_\_\_ is the attribute specified with the <LI> tag which will give a solid round black bullet.
2. The \_\_\_\_\_ attribute specified with the <LI> tag alters the numbering sequence for ordered lists.
3. \_\_\_\_\_ appears after the <DD> tag of the definition lists.

### TRUE OR FALSE

4. Lists are of three types; Ordered lists, Unordered lists and Definition lists .
5. Ordered lists are used for Bullets.
6. START alters the numbering sequence in the middle of an Ordered list.
7. Definition lists consists of two parts; Definition Term and Definition Description

#### BARBED WIRE

Concertina Barbed Wire in roll form is used in high security areas to deter trespassing men and animals. The effectiveness of this material is proved as it has been in use for more than 75 years during war and peace.

##### 1. Line Wire:

This is made up of 3.05mm diameter high carbon steel wire, with a tensile strength of 170 to 180 kg/mm<sup>2</sup>. The wire is drawn and dressed in such a manner that the coils formed will fall naturally into the specified diameter without forming a figure eight. Line wire is heavy hot dipped galvanized with minimum zinc coating of 185 gm/m<sup>2</sup>.

##### 2. Barbed Wire:

This is 2.00mm diameter bright mild steel wire with a tensile strength of 38 to 55 kg/mm<sup>2</sup> conforming to BS 1052. The wire is hot dipped galvanized with a minimum zinc coating of 50 to 50 gm/m<sup>2</sup>. The barbs are formed with four points. Spacing between the centers of the barbs of every 70mm along the length of the wire. The barbs are firmly secured by indentations made in the wire, so that the barbs do not rotate or slide along the wire. The

Diagram 3.1

## HANDS ON EXERCISE

1. Looking at the Screen given below write the HTML code making use of following tags:

- Unordered Lists
- Ordered Lists
- Definition Lists

### Text Content:

#### //Example on Unordered List//

- Sportstar
- Business Week
- Time

#### //Example on Ordered List//

- iv. Sportstar
- v. Business Week
- vi. Time

#### //Example on Definition List//

Sports Magazine  
Sportstar  
Business Magazine  
Business Week

```
// Example on Unordered List //
```

- Sportstar
- Business Week
- Time

```
// Example on Ordered List //
```

- iv Sportstar
- v Business Week
- vi. Time

```
// Example on Definition List //
```

```
Sports Magazine  
Sportstar  
Business Magazine  
Business Week
```

**Diagram 3.2:** Output of Hands on Exercise

## 4. ADDING GRAPHICS TO HTML DOCUMENTS

Other than text, HTML allows placing of static and/or animated images in an HTML page. HTML accepts two picture file formats **.gif** and **.jpg**. Using tools such as Gif Constructor or Adobe PhotoShop, images can be created to suit the requirements of a web page and saved in these file formats.

Once an image is ready and stored in the above-mentioned formats, it can be inserted into a web page using the tag `<IMG>`, which takes the name of the image file (*filename.gif filename.jpg or filename.jpeg*) as an attribute. In addition, HTML also allows control of the height, width, border and so on, of every image placed on the web Page. The `<IMG>` tag takes the following attributes:

<b>ALIGN</b>	Controls alignment of the text following the image <b>ALIGN = TOP</b> indicates the text after the image to be written at the top, next to the image. <b>ALIGN = MIDDLE</b> indicates the text after the image to be written at the middle, next to the image. <b>ALIGN = BOTTOM</b> indicates the text after the image to be written at the bottom, next to the image. Controls alignment of the image with respect to the VDU screen. <b>ALIGN = LEFT</b> indicates the image is aligned to the left with respect to the screen. <b>ALIGN = CENTER</b> indicates the image is aligned to the center with respect to the screen. <b>ALIGN = RIGHT</b> indicates the image is aligned to the right with respect to the screen.
<b>BORDER</b>	Specifies the size of the border to place around the image.
<b>WIDTH</b>	Specifies the width of the image in pixels.
<b>HEIGHT</b>	Specifies the height of the image in pixels.
<b>HSPACE</b>	Indicates the amount of space to the left and right of the image.
<b>VSPACE</b>	Indicates the amount of space to the top and bottom of the image.
<b>ALT</b>	Indicates the text to be displayed in case the Browser is unable to display the image specified in the SRC attribute.
<b>SRC</b>	Specifies the location and name of the image file.

Table 4.1

### Example:

```
<IMG Align=CENTER Border=0 Height=57 HSpace=0 Src="Image1.GIF" Width=447 >
```

The attributes taken by the `<IMG ... >` tag are explained in the following examples.

## USING THE BORDER ATTRIBUTE

### Example 1: (Refer to diagram 4.1)

```
<HTML>
  <HEAD><TITLE> Working with Images </TITLE></HEAD>
  <BODY Background="images/texture1.gif">
    <B>Controlling Image Borders!</B><CENTER>
      <I>Image Without a BORDER </I><BR><BR>
      <IMG Src = "images/corp.gif"><BR><BR>
      <I>Image With BORDER = 3</I><BR><BR>
      <IMG Border=3 Src="images/corp.gif"><BR>
    </CENTER>
  </BODY>
</HTML>
```

Output For Example 1:

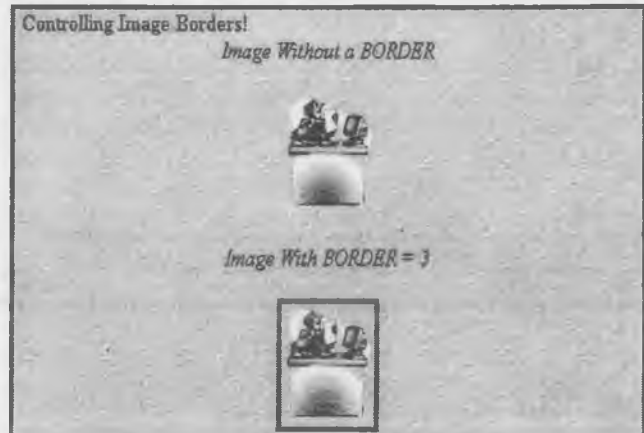


Diagram 4.1

## USING THE WIDTH AND HEIGHT ATTRIBUTE

Example 2: (Refer to diagram 4.2)

```
<HTML>
  <HEAD><TITLE> Working with Images </TITLE></HEAD>
  <BODY Background="images/texture1.gif">
    <B>Controlling Image Sizes!</B><CENTER>
      <I>Normal Image Size</I><BR><BR>
      <IMG Src="images/computer.gif"><BR>
      <I>Image With Size (Height And Width) Set To 200</I><BR><BR>
      <IMG Height=200 Src="images/computer.gif" Width=200><BR>
    </CENTER>
  </BODY>
</HTML>
```

Output For Example 2:

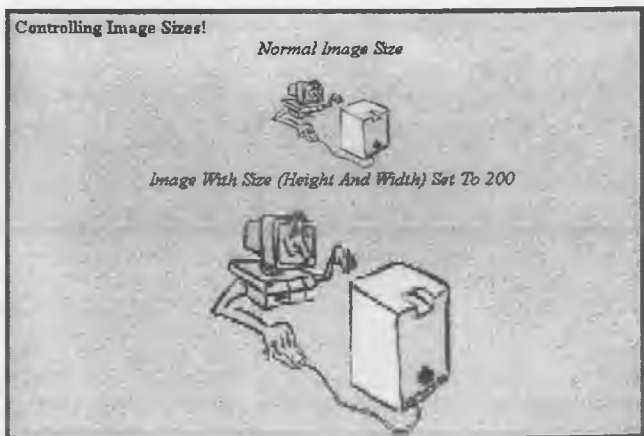


Diagram 4.2

## USING THE ALIGN ATTRIBUTE

Example 3: (Refer to diagram 4.3)

```
<HTML>
  <HEAD><TITLE> Working with Images </TITLE></HEAD>
  <BODY Background="images/texture1.gif">
    <B><I> Image Aligned Left </I></B>
    <IMG Align=Left Src="images/sctonly2.gif" ><BR><BR>Silicon Chip Technologies, <BR>
    <I> We Specialize In Corporate Training, Software Development And Placements. Our Training
    Programs and Software Development include Web Based Commercial Applications and
    Commercial Application Development using Databases. </I><BR><BR><BR><BR>
    <B><I>Image Aligned Right </I></B>
    <IMG Align=Right Src="images/sctonly2.gif" ><BR><BR>Silicon Chip Technologies, <BR>
```



<I>We Specialize In Corporate Training, Software Development And Placements. Our Training Programs and Software Development include Web Based Commercial Applications and Commercial Application Development using Databases</I>

</BODY>

</HTML>

Output For Example 3:



Diagram 4.3

## USING THE ALT ATTRIBUTE

Example 4: (Refer to diagram 4.4)

<HTML>

<HEAD><TITLE> Working with Images </TITLE></HEAD>

<BODY Background="images/texture1.gif">

<B>Use of ALT attribute</B><BR><CENTER>

<I>Avaliable Image: javacup.gif</I><BR><BR>

<IMG Src="images/javacup.gif"><BR><BR>

<I>Unavaliable Image: javac.gif - Without the ALT Attribute </I><BR><BR>

<IMG Src="images/javac.gif"><BR><BR>

<I>Unavaliable Image: javac.gif - With the ALT Attribute set to "Java"</I><BR><BR>

<IMG Alt="The Java Cup" Src="images/javac.gif"><BR>

</CENTER>

</BODY>

</HTML>

### Note



The <IMG> tag specifies a file javac.gif as the source file. This file is *not present in the current work directory*.

Therefore, the browser will display an icon indicating the unavailability of the file. If the Alt attribute is present along with the <IMG> tag then the text that is specified in the Alt attribute is displayed in place of the missing the icon.

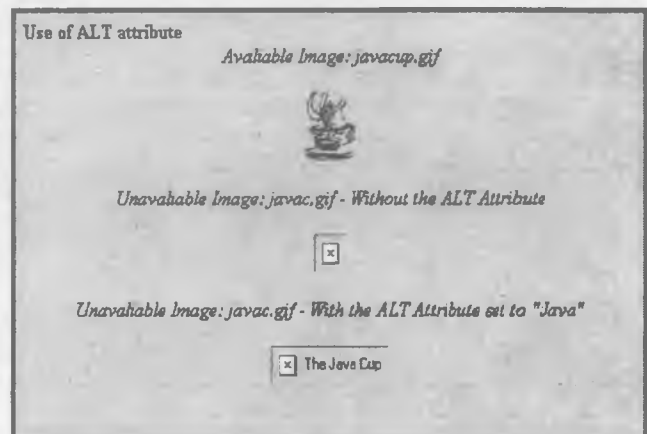


Diagram 4.4

Output For Example 4:

**FOCUS:** Increase the visual appeal of the web site by replacing or complementing the text with pictures wherever possible. For instance, the Company's Logo can be used at the start of the web page. Each product description can be supplemented with an image.

**Code Listing:**

<HTML>

```
<HEAD><TITLE> Delta Engineering Pvt. Ltd. </TITLE></HEAD>
<BODY Background="images/pinkwhit.gif">
  <FONT Face="Brush Script MT" Size=7 Color="#008000"><IMG Align=Bottom
  Src="images/logo.gif"><I>Delta Engineering Pvt. Ltd.</I><HR></FONT>
  <FONT Color="#008000"><CENTER><H3>PROFILE</H3><IMG Height="57"
  Src="images/intro.gif" Width="185"><BR><BR>Delta Engineering Pvt. Ltd. is a specialist
  manufacturer of wire and wire products. DEPL established in Mumbai, India is ideally located for
  shipments to any part of the world. The main items of manufacture, in collaboration with Gulf
  Fencing Industry (GFI) include protector gabions and protector fencing systems.<P>Protector
  fencing systems offer a wide range of solutions to all security problems on any terrain and under
  extreme climatic conditions. The protector fencing system has been used extensively in Europe,
  America and the Far East, and this has helped formulate a package especially suited to Middle
  East requirements.<P>DEPL is equipped to offer comprehensive packages including design,
  material and installation.<P>Products strictly adhere to BS, ASTM and DIN international
  specifications. Quality control is guaranteed by independent laboratory test certificates from
  India.<P>Quality control is implemented without sacrificing economy and efficiency. DEPL is
  dedicated to technical services and problem solving.<P>DEPL is following quality methods and is
  accredited with ISO 9002.<P><IMG Height="57" Src="images/planning.gif"
  Width="184"><BR>DEPL can provide planning support by offering design, technical
  specifications, drawings, foundation plans and installation instructions, together with a
  personalized service. The company's technical sales engineers keep in constant touch with all
  clients. Utmost importance is given to optimum design, versatility, durability and economy backed
  by the DEPL guarantee for work undertaken.</CENTER>
  <P>Please forward any enquiries to enq_depl@bom2.vsnl.net.in
  <P><I><B>DELTA ENGINEERING PVT. LTD.</B></I><BR>502, 5th Floor, Tejas Building <BR>
  Andheri (W), Mumbai <BR> India <BR> Telephone : 91-022-8210050</I><P><HR>
  <CENTER><H3>BARBED WIRE</H3></CENTER><IMG Align=Bottom Border=2
  Height="130" Src="images/barbed1.jpg" Width=150">Concertina Barbed Wire in roll form is
  used in high security areas to deter trespassing men and animals. The effectiveness of this material
  is proved as it has been in use for more than 75 years during war and peace.
  <OL><P><B><I><LI>Line Wire:</I></B><BR>This is made up of 3.05mm diameter high
  carbon steel wire, with a tensile strength of 170 to 180 kg/mm2. The wire is drawn and dressed in
  such a manner that the coils formed will fall naturally into the specified diameter without forming
  a figure eight. Line wire is heavy hot dipped galvanized with minimum inc coating of 185
  gm/mm2.
  <P><B><I><LI>Barbed Wire:</B></I><BR>This is 2.00mm diameter bright mild steel wire
  with a tensile strength of 38 to 55kg/mm2 conforming to BS 1052. The wire is hot dipped
  galvanized with a minimum zinc coating of 20 to 50 gm/mm2. The barbs are formed with four
  points. Spacing between the centers of the barbs of every 70mm along the length of the wire. The
  barbs are firmly secured by indentations made on the wire, so that the barbs do not rotate or slide
  along the wire. The four points of the barbs are formed at the right angles to one another and
  project outwards approximately 12mm from the center of the wire.
  <P><B><I><LI>Carrying Handles:</I></B><BR>Handles are made of mild steel wire of
  3.55mm diameter and are attached to the outer turn of the coil on each side.</OL><P><HR>
```

**BARBED TAPES**

Barbed Tapes are used as psychological and physical deterrent against intrusion by personnel and animals. Barbed tape barrier systems are more vicious and difficult to tamper with, providing superior perimeter security. Some of the users are military installations, nuclear energy sites, maximum-security prisons, various petroleum installations, tank farms and other important industrial facilities.

**Short Blade Barbed Tape (SBBT):**  
 To give the maximum tensile strength at the time of the breach hard drawn steel cord is used (ES1). For camouflage purposes, a coaltar coating can be applied over the entire surface (ES-1), and to attain the high degree of rust resistance, the core wire is made of galvanized steel (ES-2). Any of the above three options are available to meet your specific requirements.

**Medium Blade Barbed Tape (MBBT):**  
 The specification of the "ivory" tape is similar to those of ES types. The blades are sharper and have a greater pricking capacity. The life of this tape is more than three times longer than those of ES types. Therefore, "ivory" tape is a more effective choice in terrain which has heavy rain or in coastal areas. The diameter will be similar to ES type.

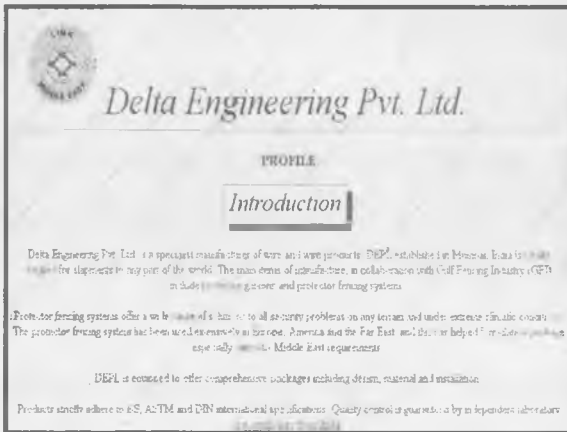
**Long Blade Barbed Tape (LBBT):**  
 This is the most effective psychological and physical deterrent, ever made as a barrier obstacle. It is available in authentic stainless steel (SUS 430) and the core wire can be fabricated from either galvanized carbon steel or stainless steel.

**ANIMAL FENCING**

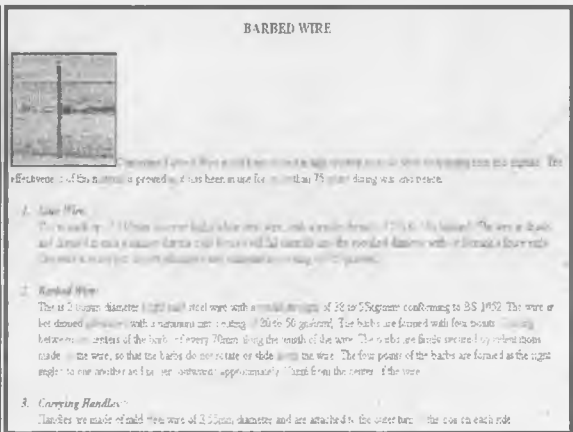
DEPL's animal fencing system is mainly used as an anti-intrusion barrier against any farm and other animals. The most common use is on highways where vast distances are covered at a very economical cost. Animal fencing can also be used to enclose areas such as farms, forest areas and national parks, where security is not crucial. This system is easy to install and can be erected in a comparatively short time.

</BODY>  
 </HTML>

**Output:**



**Diagram 4.5.1**



**Diagram 4.5.2**

The output in diagram 4.5.1 and diagram 4.5.2 displays the same web site, but now with pictures inserted wherever possible. The first page displays the Company Logo and another small picture, which displays the word *Introduction*. The second page shows the picture of a *Barbed Wire*. Similarly, there are pictures for Barbed Tape, Animal Fencing and so on.

Displayed in the second page, the text following the picture starts at the bottom of the picture.

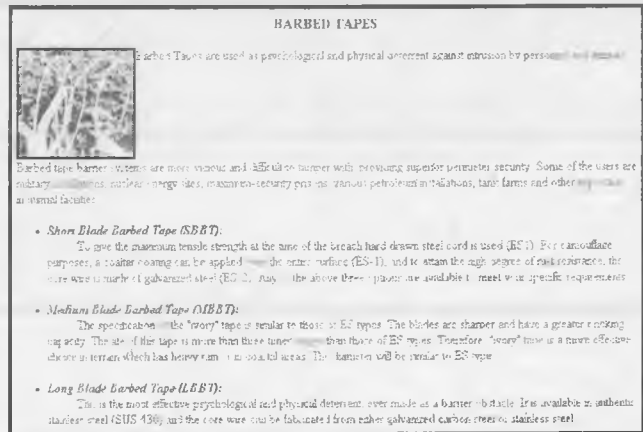
It would be nice to have the whole text about barbed tape appearing exactly besides the picture of barbed tape. To have text aligned starting from the top of the image. The IMG tag attributes can be rewritten as:

```
<IMG Align=Top Border=3 Height="130" Src="images/barbed.jpg" Width="150">
```

This would display in the browser as seen in diagram 4.6.

This is because the image and the first line of text are considered to be placed in *one line*. To overcome this problem, HTML allows the placement of tables on a web page. Then place the image in one cell of the table, and the text in the adjacent cell. This will make the entire text to appear aligned next to the image.

The width of the table, the number of columns in a table and so on, can be controlled according to requirements. The following chapter deals with how to create tables and place them on a web page together with all the attributes that the <TABLE> tag takes.



**Diagram 4.6**

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. HTML accepts \_\_\_\_\_ and \_\_\_\_\_ picture formats.
2. An Image can be inserted into the HTML page using \_\_\_\_\_ tag that takes the name of the image file as an attribute.
3. \_\_\_\_\_ and \_\_\_\_\_ attributes are used to control the size of the image on the web page.
4. \_\_\_\_\_ attribute is used to set a border around the image.
5. The amount of space to the top and bottom of the image is indicated by \_\_\_\_\_.
6. The \_\_\_\_\_ attribute takes text to be displayed in case the Browser is unable to display the image specified in the SRC attribute.

### TRUE OR FALSE

7. HTML allows only static text to be displayed on the page.
8. GIF Constructors or Adobe Photoshop can be used to create .GIF and .JPG picture formats.

9. ALIGN=LEFT indicates the image is aligned to the left with respect to the screen.
10. SRC takes the name of an image file to be displayed as a parameter.

## HANDS ON EXERCISE

1. Design a web page using the image files **House.gif**, **Javacup.gif**, and **Computer.gif** according to the following specifications.
  - Use a Border for **House.gif**.
  - Resize the **Height** and **Width** of **Javacup.gif** and **Computer.gif** to 100 pixels each.
  - Align** the text with respect to the images so as to obtain the output displayed in the diagrams 4.7.1 and 4.7.2.

### Text Content

#### The World Wide Web

The World Wide Web or simply the Web has been a 'Killer App' of the Internet. Certainly its capabilities to display text and graphics provide access to other pages and information has made the fastest growing component of the Internet. Major online services are American Online, CompuServe and Prodigy.

#### Get Connected

A way to access is to get an account with an Internet Service Provider, or ISP. These accounts include access to the World Wide Web and other Internet resources, and often provide space to store Web pages one will create. A very helpful feature of the web is the capability to move from page to page by selecting specific highlighted words and phrases or images, which are called as links.

#### Output For Hands On Exercise:

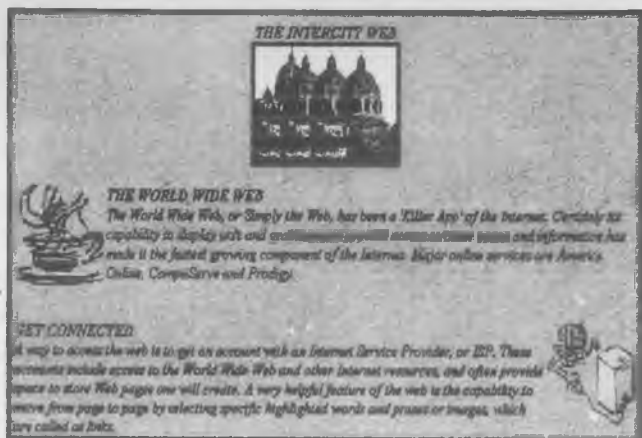


Diagram 4.7: Output for Hands on Exercise.

## 5. TABLES

### INTRODUCTION

A table is a two dimensional matrix, consisting of rows and columns. Tables are intended for displaying data in columns on a web page. All table related tags are included between the `<TABLE>...</TABLE>` tags. Each row of a table is described between the `<TR>...</TR>` tags. Each column of a table is described between the `<TD>...</TD>` tags.

Table rows can be of two types:

- ❑ **Header Rows:** A row that spans across columns of a table is called the Header Row. A table header row is defined using `<TH>...</TH>` tags. The content of a table header row is automatically centered and appears in boldface.
- ❑ **Data Rows:** Individual data cells placed in the horizontal plane creates a data row. There could be a single data cell (i.e. a single column table) or multiple data cells (i.e. a multi column table)

Data cells hold data that must be displayed in the table. A data row is defined using `<TR>...</TR>` tags. Text matter displayed in a data row is left justified by default. Any special formatting like boldface or italics is done by including appropriate formatting tags inside the `<TR>...</TR>` tags. An image can also be displayed in a data cell.

The attributes that can be included in the `<TABLE>` tag are:

<b>ALIGN</b>	Horizontal alignment is controlled by the ALIGN attribute. It can be set to LEFT, CENTER, or RIGHT.
<b>VALIGN</b>	Controls the vertical alignment of cell contents. It accepts the values TOP, MIDDLE or BOTTOM.
<b>WIDTH</b>	Sets the WIDTH to a specific number of pixels or to a percentage of the available screen width. If width is not specified, the data cell is adjusted based on the cell data value.
<b>BORDER</b>	Controls the border to be placed around the table. The border thickness is specified in pixels.
<b>CELLPADDING</b>	This attribute controls the distance between the data in a cell and the boundaries of the cell.
<b>CELLSPACING</b>	Controls the spacing between adjacent cells.
<b>COLSPAN</b>	The COLSPAN attribute inside a <code>&lt;TH&gt;</code> or <code>&lt;TD&gt;</code> tag instructs the browser to make the cell defined by the tag to take up more than one column. The COLSPAN attribute can be set equal to the number of columns the cell is to occupy. This attribute is useful when one row of the table needs to be a certain number of columns wide.
<b>ROWSPAN</b>	The ROWSPAN attribute works in the same way as the COLSPAN attribute except that it allows a cell to take up more than one row. The attribute can be set by giving a numeric value. For example ROWSPAN = 3.

Table 5.1

### The Caption Tag

Often tables need to be given a heading, which gives the reader a context for the information in the tables. Table Headings are called Captions. Captions can be provided to a table by using the `<CAPTION>...</CAPTION>` tags. This paired tag appears within the `<TABLE>...</TABLE>` tags. The table caption can be made to appear above or below the table structure with the help of the attribute ALIGN, as explained in table 5.2.

<b>ALIGN</b>	<p>It controls placing of the caption with respect to the table.</p> <ul style="list-style-type: none"> <li>• <b>ALIGN = BOTTOM</b> will place the caption immediately below the table</li> <li>• <b>ALIGN =TOP</b> will place the caption immediately above the table.</li> </ul>
--------------	--

**Table 5.2**

By passing a row's <TR> tag the VALIGN and ALIGN attributes, vertical or the horizontal alignment can be made identical for every cell in a given row.

By passing the <TH> and/or <TD> tags, VALIGN or ALIGN attributes, vertical or horizontal alignments in both header and data cells can be done. *Any alignment specified at the cell level overrides any default alignments and any alignments specified in a <TR> tag.*

### Note



- Alignments specified in <TD> or <TH> apply only to the cell being defined.
- Alignments specified in a <TR> tag apply to all cells in a row, unless overridden by an alignment specification in a <TD> or <TH> tag.

## USING THE WIDTH AND BORDER ATTRIBUTE

**Example 1:** (Refer to diagram 5.1)

```
<HTML>
<HEAD><TITLE>Table Attributes</TITLE></HEAD>
<BODY BgColor=LIGHTGREY>
  <B>Specifying the BORDER and WIDTH of the Table!</B><BR><BR><BR><BR>
  <CENTER><TABLE Border=5 Width=50%>
    <CAPTION Align=Bottom><B>Personal Information</B></CAPTION>
    <TR><TH>NAME</TH><TH>AGE</TH></TR>
    <TR Align=CENTER><TD>Shilpa</TD><TD>21</TD></TR>
    <TR Align=CENTER><TD>Vaishali</TD><TD>22</TD></TR>
  </TABLE></CENTER>
</BODY>
</HTML>
```

**Output For Example 1:**

**Specifying the BORDER and WIDTH of the Table!**

NAME	AGE
Shilpa	21
Vaishali	22

**Personal Information**

### Note



If the **WIDTH** attribute is associated with the <TH> tag then the width of an individual column can be adjusted.

**Diagram 5.1**

## USING THE CELLPADDING ATTRIBUTE

**Example 2:** (Refer to diagram 5.2)

```
<HTML>
  <HEAD><TITLE>Working With Table</TITLE></HEAD>
  <BODY BgColor=LIGHTGREY>
    <B>Specifying CELLPADDING!</B><BR><HR>
    <I>Without Cellpadding</I>
    <CENTER><TABLE Border=1 Align=CENTER Width=25%>
      <TR><TH>NAME</TH><TH>AGE</TH></TR>
      <TR Align=CENTER><TD>Shilpa</TD><TD>21</TD></TR>
      <TR ALIGN=CENTER><TD>Vaishali</TD><TD>22</TD></TR>
    </TABLE></CENTER><HR>
    <I>With Cellpadding of 10</I>
    <CENTER><TABLE Align=CENTER Border=1 Cellpadding=10 Width=25%>
      <TR><TH>NAME</TH><TH>AGE</TH></TR>
      <TR Align=CENTER><TD>Shilpa</TD><TD>21</TD></TR>
      <TR Align=CENTER><TD>Vaishali</TD><TD>22</TD></TR>
    </TABLE></CENTER>
  </BODY>
</HTML>
```

**Output For Example 2:**

## USING THE CELLSPACING ATTRIBUTE

**Example 3:** (Refer to diagram 5.3)

```
<HTML>
  <HEAD><TITLE>Working With
    Table</TITLE></HEAD>
  <BODY BGCOLOR=LIGHTGREY>
    <B>Controlling the space between Adjacent Cells!</B><BR><BR><HR>
    <I>Without Cellspacing</I>
    <CENTER><TABLE Align=CENTER Border=1 Width=25%>
      <TR><TH>NAME</TH><TH>AGE</TH></TR>
      <TR Align=CENTER><TD>Shilpa</TD><TD>21</TD></TR>
      <TR Align=CENTER><TD>Vaishali</TD><TD>22</TD></TR>
    </TABLE></CENTER><HR>
    <I>With Cellspacing of 10</I>
    <CENTER><TABLE Align=CENTER Border=1 Cellspacing = 10 Width=25%>
      <TR><TH>NAME</TH><TH>AGE</TH></TR>
      <TR Align=CENTER><TD>Shilpa</TD><TD>21</TD></TR>
      <TR Align=CENTER><TD>Vaishali</TD><TD>22</TD></TR>
    </TABLE></CENTER>
  </BODY>
</HTML>
```

Specifying CELLPADDING!	
<i>Without Cellpadding</i>	
NAME	AGE
Shilpa	21
Vaishali	22
<i>With Cellpadding of 10</i>	
NAME	AGE
Shilpa	21
vaishali	22

**Diagram 5.2**



Output For Example 3:

USING THE BGCOLOR ATTRIBUTE

Example 4: (Refer to diagram 5.4)

```
<HTML>
  <HEAD><TITLE>Working With Tables</TITLE></HEAD>
  <BODY BgColor=LIGHTGREY>
    <B>Specifying Coloured Table Cells!</B><BR><BR><BR><BR>
    <CENTER><TABLE Align=CENTER Border=1 Width=50%>
      <TR><TH BgColor=GRAY>NAME</TH><TH BgColor=GRAY>AGE</TH></TR>
      <TR Align=CENTER>
        <TD BgColor=BLACK><FONT Color=White>Shilpa</FONT></TD>
        <TD BgColor=VIOLET><FONT Color=RED>21</FONT></TD>
      </TR><TR Align=CENTER>
        <TD BgColor=BLUE><FONT Color=WHITE>Vaishali</FONT></TD>
        <TD BgColor=RED><FONT Color=BLUE>22</FONT></TD>
      </TR><CAPTION Align=Bottom><B><BR>Personal Information</B></CAPTION>
    </TABLE></CENTER>
  </BODY>
</HTML>
```

Output For Example 4:

*Tip*



If the entire table needs to be displayed using the same Background Color, the attribute BgColor can be passed to the <TABLE> tag itself.

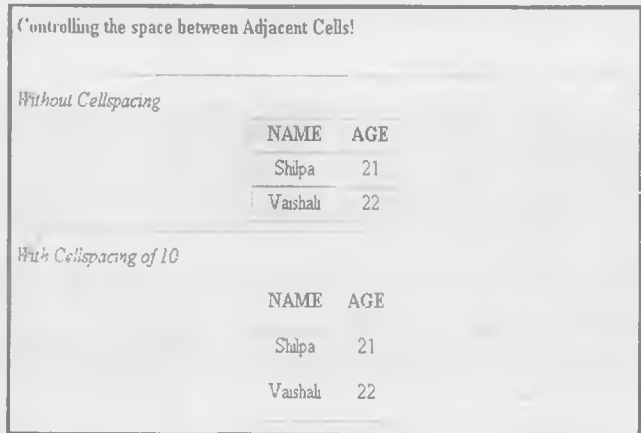


Diagram 5.3

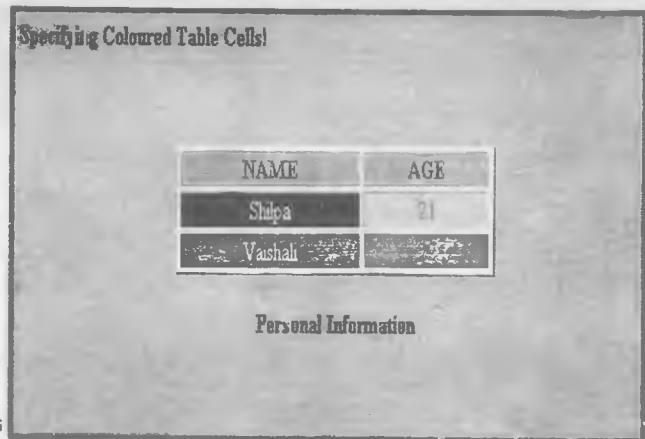


Diagram 5.4

USING THE COLSPAN AND ROWSPAN ATTRIBUTES

Example 5: (Refer to diagram 5.5)

```
<HTML>
  <HEAD><TITLE>Working With Table</TITLE></HEAD>
  <BODY BgColor=LIGHTGREY>
    <B>Specifying ROWSPAN and COLSPAN Attributes!</B><BR><BR><BR><BR>
    <CENTER><TABLE Align=CENTER Border=1 Width=50%><TR>
```

```

<TH RowSpan=2>NAME</TH><TH ColSpan=3>MARKS</TH>
</TR><TR>
  <TH>PowerBuilder</TH><TH>VisualBasic</TH><TH>Developer2000</TH>
</TR><TR ALIGN=CENTER>
  <TD>Shilpa</TD><TD>21</TD><TD>45</TD><TD>30</TD>
</TR><TR ALIGN=CENTER>
  <TD>Vaishali</TD><TD>26</TD><TD>30</TD><TD>40</TD>
</TR><CAPTION ALIGN=bottom><B><BR>Mark Sheet</B></CAPTION>
</TABLE></CENTER>
</BODY>
</HTML>

```

**Output For Example 5:**

**FOCUS:** The Management at DEPL has decided that all their product images should be displayed on the Web Site in appropriate web pages. The textual description of each product should come up alongside the image and not below it.

This can be done by the use of tables on the web page.

**Specifying ROWSPAN and COLSPAN Attributes!**

		Column Span		
Row Span	NAME	MARKS		
		PowerBuilder	VisualBasic	Developer2000
	Shilpa	21	45	30
	Vaishali	26	30	40

**Mark Sheet**

**Diagram 5.5****Code Listing:**

```

<HTML>
<HEAD><TITLE> Delta Engineering Pvt. Ltd. </TITLE></HEAD>
<BODY Background="images/pinkwhit.gif">
  <FONT Face="Brush Script MT" Size=7 Color="#008000"><CENTER><IMG Align=Bottom
  Src="images/logo.gif"><I>Delta Engineering Pvt. Ltd.</I></CENTER><HR></FONT>
  <FONT Color="#008000"><CENTER><H3>PROFILE</H3><IMG Height="57"
  Src="images/intro.gif" Width="185"><BR><BR>Delta Engineering Pvt. Ltd. is a specialist
  manufacturer of wire and wire products. DEPL established in Mumbai, India is ideally located for
  shipments to any part of the world. The main items of manufacture, in collaboration with Gulf
  Fencing Industry (GFI) include protector gabions and protector fencing systems.<P>Protector
  fencing systems offer a wide range of solutions to all security problems on any terrain and under
  extreme climatic conditions. The protector fencing system has been used extensively in Europe,
  America and the Far East, and this has helped formulate a package especially suited to Middle
  East requirements.<P>DEPL is equipped to offer comprehensive packages including design,
  material and installation.<P>Products strictly adhere to BS, ASTM and DIN international
  specifications. Quality control is guaranteed by independent laboratory test certificates from
  India.<P>Quality control is implemented without sacrificing economy and efficiency. DEPL is
  dedicated to technical services and problem solving.<P>DEPL is following quality methods and is
  accredited with ISO 9002.<P><IMG Height="57" Src="images/planning.gif"
  Width="184"><BR>DEPL can provide planning support by offering design, technical
  specifications, drawings, foundation plans and installation instructions, together with a
  personalized service. The company's technical sales engineers keep in constant touch with all
  clients. Utmost importance is given to optimum design, versatility, durability and economy backed
  by the DEPL guarantee for work undertaken.</CENTER>
  <P>Please forward any enquiries to enq_depl@bom2.vsnl.net.in

```

<P><I><B>DELTA ENGINEERING PVT. LTD.</B><BR>502, 5th Floor, Tejas Building <BR>Andheri (W), Mumbai <BR> India <BR> Telephone : 91-022-8210050</I><P><HR>

<CENTER><H3>BARBED WIRE</H3></CENTER>

<TABLE Border=0><TR><TD>

<IMG Align=Bottom Border=2 Height="130" Src="images/barbed1.jpg" Width=150">

</TD><TD>

<FONT Color="#008000">Concertina Barbed Wire in roll form is used in high security areas to deter trespassing men and animals. The effectiveness of this material is proved as it has been in use for more than 75 years during war and peace. </FONT>

</TD></TR></TABLE>

<OL><P><B><I><LI>Line Wire:</I></B><BR>This is made up of 3.05mm diameter high carbon steel wire, with a tensile strength of 170 to 180 kg/mm<sup>2</sup>. The wire is drawn and dressed in such a manner that the coils formed will fall naturally into the specified diameter without forming a figure eight. Line wire is heavy hot dipped galvanized with minimum inc coating of 185 gm/mm<sup>2</sup>.

<P><B><I><LI>Barbed Wire:</B></I><BR>This is 2.00mm diameter bright mild steel wire with a tensile strength of 38 to 55kg/mm<sup>2</sup> conforming to BS 1052. The wire is hot dipped galvanized with a minimum zinc coating of 20 to 50 gm/mm<sup>2</sup>. The barbs are formed with four points. Spacing between the centers of the barbs of every 70mm along the length of the wire. The barbs are firmly secured by indentations made on the wire, so that the barbs do not rotate or slide along the wire. The four points of the barbs are formed at the right angles to one another and project outwards approximately 12mm from the center of the wire.

<P><B><I><LI>Carrying Handles:</I></B><BR>Handles are made of mild steel wire of 3.55mm diameter and are attached to the outer turn of the coil on each side.</OL><P><HR>

<CENTER><H3>BARBED TAPES</H3></CENTER>

<TABLE Border=0><TR><TD>

<IMG Align=Bottom Border=3 Height="130" Src="images/barbed.jpg" Width="150">

</TD><TD>

<FONT Color="#008000">Barbed Tapes are used as psychological and physical deterrent against intrusion by personnel and animals. Barbed tape barrier systems are more vicious and difficult to tamper with, providing superior perimeter security. Some of the users are military installations, nuclear energy sites, maximum-security prisons, various petroleum installations, tank farms and other important industrial facilities.</FONT>

</TD></TR></TABLE>

<UL><P><LI><DT><B><I>Short Blade Barbed Tape (SBBT):</I></B><BR>

<DD>To give the maximum tensile strength at the time of the breach hard drawn steel cord is used (ES1). For camouflage purposes, a coaltar coating can be applied over the entire surface (ES-1), and to attain the high degree of rust resistance, the core wire is made of galvanized steel (ES-2). Any of the above three options are available to meet your specific requirements.

<P><LI><DT><B><I>Medium Blade Barbed Tape (MBBT):</I></B><BR>

<DD>The specification of the "ivory" tape is similar to those of ES types. The blades are sharper and have a greater pricking capacity. The life of this tape is more than three times longer than those of ES types. Therefore, "ivory" tape is a more effective choice in terrain which has heavy rain or in coastal areas. The diameter will be similar to ES type.

<P><LI><DT><B><I>Long Blade Barbed Tape (LBBT):</I></B><BR>

<DD>This is the most effective psychological and physical deterrent, ever made as a barrier obstacle. It is available in authentic stainless steel (SUS 430) and the core wire can be fabricated from either galvanized carbon steel or stainless steel.

```

</UL><P><HR>
<CENTER><H3>ANIMAL FENCING</H3></CENTER>
<TABLE Border=0><TR><TD>
  <IMG Align=Bottom Border=3 Height="130" Src="images/fence1.jpg" Width="150">
</TD><TD>
  <FONT Color="#008000">DEPL's animal fencing system is mainly used as an anti-intrusion
  barrier against any farm and other animals. <P> The most common use is on highways where
  vast distances are covered at a very economical cost. <P> Animal fencing can also be used
  to enclose areas such as farms, forest areas and national parks, where security is not crucial.
  <P> This system is easy to install and can be erected in a comparatively short time.</FONT>
</TD></TR></TABLE><HR>
</FONT>
</BODY>
</HTML>

```

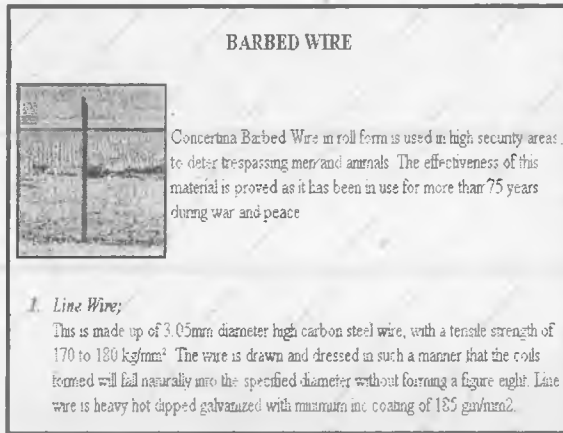
**Output:**

Diagram 5.6

<b>Short Blade Barbed Tape (SBBT):</b>	To give the maximum tensile strength at the time of the breach hard drawn steel cord is used (ES1). For camouflage purposes, a coaltar coating can be applied over the entire surface (ES-1), and to attain the high degree of rust resistance, the core wire is made of galvanized steel (ES-2). Any of the above three options are available to meet your specific requirements.
<b>Medium Blade Barbed Tape (MBBT):</b>	The specification of the "Ivory" tape is similar to those of ES types. The blades are sharper and have a greater pricking capacity. The life of this tape is more than three times longer than those of ES types. Therefore, "Ivory" tape is a more effective choice in terrain which has heavy rain or in coastal areas. The diameter will be similar to ES type.
<b>Long Blade Barbed Tape (LBBT):</b>	This is the most effective psychological and physical deterrent ever made as a barrier obstacle. It is available in authentic stainless steel (SUS 430) and the core wire can be fabricated from either galvanized carbon steel or stainless steel.

Diagram 5.7

Detailed information about the types of barbed tapes currently represented by means of bullets would now be represented in a tabular format as shown in diagram 5.7.

The output in diagram 5.7 can be achieved by writing the following code.

**Code Listing:**

```

<P><TABLE Border="4"><TR>
  <TD><FONT Color="#008000"><B><I>Short Blade Barbed Tape (SBBT):
  </I></B></FONT></TD>
  <TD><FONT Color="#008000">To give the maximum tensile strength at the time of the
  breach hard drawn steel cord is used (ES1). For camouflage purposes, a coaltar coating
  can be applied over the entire surface (ES-1), and to attain the high degree of rust
  resistance, the core wire is made of galvanized steel (ES-2). Any of the above three
  options are available to meet your specific requirements. </FONT></TD>
</TR><TR>
  <TD><FONT Color="#008000"><B><I>Medium Blade Barbed Tape
  (MBBT):</I></B></FONT></TD>

```

```
<TD><FONT Color="#008000">The specification of the "ivory" tape is similar to those of
ES types. The blades are sharper and have a greater pricking capacity. The life of this
tape is more than three times longer than those of ES types. Therefore, "ivory" tape is a
more effective choice in terrain which has heavy rain or in coastal areas. The diameter
will be similar to ES type.</FONT></TD>
```

```
</TR><TR>
```

```
<TD><FONT Color="#008000"><B><I>Long Blade Barbed Tape
(LBBT):</I></B></FONT></TD>
```

```
<TD><FONT Color="#008000">This is the most effective psychological and physical
deterrent, ever made as a barrier obstacle. It is available in authentic stainless steel
(SUS 430) and the core wire can be fabricated from either galvanized carbon steel or
stainless steel.</FONT></TD>
```

```
</TR></TABLE><P><HR>
```

The above HTML code display text in a much more presentable and attractive format. However, it still suffers from a drawback. Consider a user wanting information only about barbed tapes. To reach the required information, the user needs to scroll down the entire text and search for information concerned with barbed tapes.

In other words, the user will have to read the introduction to the company, information about barbed wire and then reach information about barbed tapes. The site is still not very user friendly. To increase user-friendliness, an index would be preferred. The entries in this index would serve as shortcuts to specific information, and would provide the option to access whatever information is relevant.

Such shortcuts are called Hyperlinks. Creation and Use of Hyperlinks are explained in detail in the following chapter.

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. A Table is a two dimensional matrix consisting of \_\_\_\_\_ and \_\_\_\_\_.
2. Table related tags are included between the \_\_\_\_\_ and \_\_\_\_\_ tags.
3. \_\_\_\_\_ controls the spacing between adjacent cells in the table.
4. Table rows can be of \_\_\_\_\_ and \_\_\_\_\_ types.
5. The horizontal alignment of the table on the page is controlled by the \_\_\_\_\_ attribute.
6. The \_\_\_\_\_ attribute is used to assign the width of the table.
7. The distance between the data in a cell and the boundaries of the cell is controlled by \_\_\_\_\_ attribute.
8. The \_\_\_\_\_ tag indicates a new row of a table.

### TRUE OR FALSE

9. The vertical or the horizontal alignment for every cell in a given row is controlled by using the VALIGN and ALIGN attributes in the row's <TR> tag.
10. A table header row is defined with the <TD> and </TD> tag pair.
11. CELSPACING controls the distance between the data in a cell and the boundaries of the cell.
12. ALIGN=TOP will place the caption immediately above the table.

## HANDS ON EXERCISES

1. Create a Web page giving the following train details.
  - Train name
  - Starting Place
  - Destination
  - Arrival and Departure time
  - Fare

Place a border for the table and use cell padding to present the cell data with clarity. Align the table in the center of the screen. Use a caption saying 'Time Table and Fare list'. Let the output be as shown in the diagram 5.8 below.

## Text Content

Name of Train	Place	Destination	Time		Fare
			Arrival	Departure	
Rajdhani Express	Bombay	Delhi	07.30	08.45	Rs. 989.00
Madras Mail	Bombay	Madras	09.00	10.15	Rs. 450.00
Konya Express	Bombay	Bangalore	11.30	12.25	Rs. 645.05
Konkan Express	Bombay	Mangalore	13.30	14.45	Rs. 756.00
Deccan Express	Bombay	Madras	13.30	14.45	Rs. 756.00

Table 5.3: Time Table And Fare List

Time Table And Fare List					
Name of Train	Place	Destination	Time		Fare
			Arrival	Departure	
Rajdhani Express	Bombay	Delhi	07.30	08.45	Rs 989.00
Madras Mail	Bombay	Madras	09.00	10.15	Rs 450.00
Konya Express	Bombay	Banglore	11.30	12.25	Rs 645.00
Konkan Express	Bombay	Manglore	13.30	14.45	Rs 756.00
Deccan Express	Bombay	Pune	16.00	17.30	Rs 345.00

Diagram 5.8: Output of Hands on Exercise

## 6. LINKING DOCUMENTS

### LINKS

HTML allows *linking* to other HTML documents as well as images. Clicking on a section of text or an image in one web page will open an entire web page or an image. The text or an image that provides such linkages is called *Hypertext*, a *Hyperlink*, or a *Hotspot*.

The browser distinguishes Hyperlinks from normal text. Every Hyperlink,

- Appears blue in color
  - *The default color setting in a browser for Hyperlink text or image*
  - *The color can be set dynamically via an HTML program if required*
- The hyperlink text / image is underlined
- When the mouse cursor is placed over it, the standard arrow shaped mouse cursor changes to the shape of a hand

The blue color, which appears by default, can be over-ridden. To change these link colors, there are three attributes that can be specified with the **<BODY>** tag. These are:

<b>LINK</b>	Changes the default color of a Hyperlink to whatever color is specified with this tag. The user can specify the color name or an equivalent hexadecimal number.
<b>ALINK</b>	Changes the default color of a Hyperlink that is activated to whatever color is specified with this tag. The user can specify the color name or an equivalent hexadecimal number.
<b>VLINK</b>	Changes the default color of a Hyperlink that is already visited to whatever color is specified with this tag. The user can specify the color name or an equivalent hexadecimal number.

Table 6.1

Links are created in a web page by using the **<A> . . . </A>** tags. Anything written between the **<A>** **</A>** tags becomes a hyperlink/hotspot. By clicking on the hyperlink navigation to a different web page or image takes place.

The document to be navigated needs to be specified. By using the HREF attribute of the **<A>** tag the next navigable web page or image can be specified.

#### Syntax:

```
<A HRef = "filename.htm">
```

Hyperlinks can be of two types:

- Links to an external document
- Links (jumps) to a specific place within the same document  
*Generally done in case of a web page containing a large amount of text*

#### External Document References

##### Example:

```
<A HRef = "details.htm">Visit my Home Page</A>
```

Here, *Visit my Home Page* becomes a *hyperlink*, and links to another document, **details.htm**, which is present in the current working directory. If the file is not present in the current directory, *a relative or absolute path can be specified*.

By default, a hyperlink takes a user to the beginning of the new web page. At times, it might be necessary to jump to a *particular location* within the new web page. To enable a jump to a specific location on a web page, *named anchors* can be set up. Anchors target hyperlinks to a specific location point on a web page.

Jumping to a particular location on a web page can be summarized in two steps:

□ **Step One:**

Mark the location to be jumped to i.e. Identify the location in a web page to jump to by giving the location a name.

Using the NAME attribute of the <A> tag does this.

**Syntax:**

```
<A Name = "location_name">
```

**Example:**

```
<A Name = "point1">
```

This identifies a location to be jumped to as *point1*.

□ **Step Two:**

While jumping to a specific web page and a specific location on the web page, in addition to the name of the web page to be jumped to, the name of the location on the web page to go to is required.

Hence the web page to jump to, requires a *filename.htm*, together with the name of the location to jump to in the HTML file.

This is done as follows:

**Syntax:**

```
<A HRef = "file_name.htm # location_name"> ... </A>
```

**Example:**

```
<A HRef = "details.htm# point1">Visit My Home Page</A>
```

*Visit my Home Page* becomes a Hotspot and leads to a location named *point1* in the file *details.htm*.

## Internal document references

Sometimes, a jump is required to a different location in the same document. Since the jump has to be targeted to a specific location, the same two steps need to be performed as before, i.e. identify a location with a name and then jump to that location using the name. The only difference is that the *filename.htm* now will be the **current filename.htm**.

**Syntax:**

```
<A Name="location_name">
```

```
<A HRef="# location_name"> ... </A>
```

## Note



The absence of the *filename.htm* before the # symbol indicates that a jump is required within the same document.

**Example:**

```
<A Name="point1">
```

```
<A HREF = "# point1"> Visit My Home Page </A>
```



Visit my Home Page becomes a Hotspot and leads to a location named *point1* in the same document.

## Note



Ensure that the named location is specified in the HTML file where a jump is being made.

## Hyper Linking To A HTML File (Starting At The Beginning Of The Document)

### Example 1:

The Web Page shown in diagram 6.1 presents SCT as an institute for Corporate Training. The subjects, undertaken for Corporate Training are listed, each of which is a Hyperlink, and clicking on the hyperlink will lead to a different document, showing a syllabus for the corresponding subject.

Clicking on *HTML* will show the syllabus of HTML, clicking on *Javascript* will show the syllabus of Javascript, and so on.

The syllabus for each subject is coded within separate files. The HTML syllabus is within a file called *HTMLSyl.html*. Thus, clicking on the *HTML* hyperlink opens a file, *HTMLSyl.html* that gives a listing of the syllabus of HTML.

The file *Index.htm* is the one that displays the hyperlinks to the syllabi of various subjects.

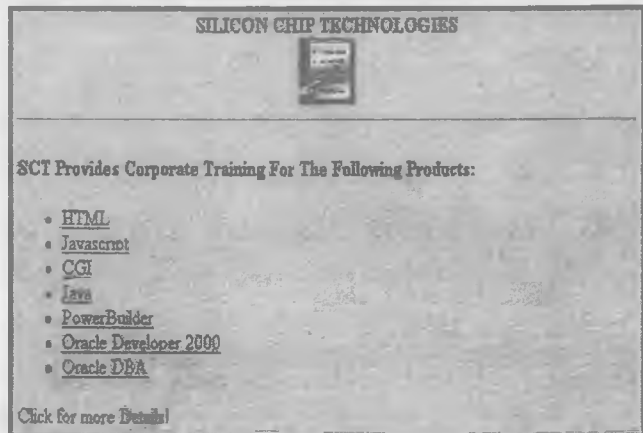


Diagram 6.1

### Code for Index.html

```
<HTML>
  <BODY Background="../images/texture1.gif">
    <CENTER><FONT Face="LatinoPalSH">
      <B>SILICON CHIP TECHNOLOGIES</B><BR>
      <IMG Width=50 Height=50 Src="../images/traing31.gif">
    </FONT></CENTER><HR>
    <H4>SCT Provides Corporate Training For The Following Products:</H4>
    <UL>
      <LI><A HRef="HTMLSyl.html">HTML</A>
      <LI><A HRef="JsSyl.html">Javascript</A>
      <LI><A HRef="CGISyl.html">CGI</A>
      <LI><A HRef="JvSyl.html">Java</A>
      <LI><A HRef="PBSyl.html">PowerBuilder</A>
      <LI><A HRef="OraSyl.html">Oracle Developer 2000</A>
      <LI><A HRef="DbaSyl.html">Oracle DBA</A>
    </UL><SPACER Size=275>Click for more Details!
  </BODY>
</HTML>
```

Subsequently, when the 'HTML' hyperlink is clicked on the *Htmlsyl.htm* is navigated to.

**Code for HTMLSyl.html**

```

<HTML>
  <BODY Background=" ../images/texture1.gif"><BR><B>
    <CENTER><H3>We Cover The Following Topics . . . </H3><HR><BR></CENTER>
    <OL>
      <SPACER Size = 200><LI>Text Formatting<BR>
      <SPACER Size = 200><LI>Creation Of Lists<BR>
      <SPACER Size = 200><LI>Creation Of Tables<BR>
      <SPACER Size = 200><LI>Creation Of Graphics<BR>
      <SPACER Size = 200><LI>Creation Of Hyperlinks<BR>
      <SPACER Size = 200><LI>Creation Of Imagemaps<BR>
      <SPACER Size = 200><LI>Creation Of Forms<BR>
    </OL>
  </B></BODY>
</HTML>

```

Similarly, different files can be created for each syllabus and called from the file Index.html. The HTML file names will have to be the same as the ones specified in Index.html.

**Linking To A Particular Location In A Separate Document****Example 2:**

The Web page shown in diagram 6.2.1 informs a reader about guidelines should be followed while developing web sites. This information is grouped into sections. Clicking on an appropriate section will display specific information.

Information about individual sections is held in one file called as **sections.htm**. Depending upon the section that a reader user clicks on, information about *only that particular section* will be seen on the reader's VDU.

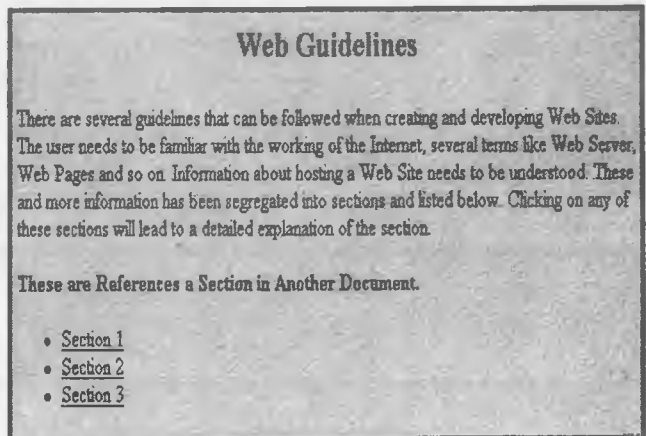
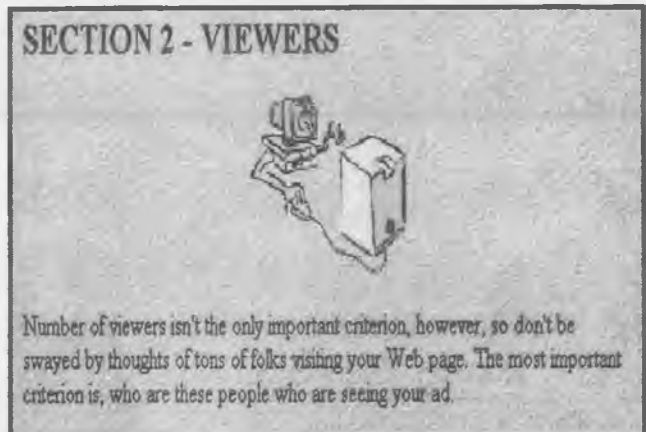
For instance, clicking on **Section 2** will display to the user, the output as shown in diagram 6.2.2.

**Code for Index.html**

```

<HTML>
  <HEAD><TITLE>Links To A Particular Location In A Separate Document</TITLE></HEAD>
  <BODY Background = " ../images/texture1.gif">
    <CENTER><FONT Face = "LatinoPalSH" Size=+2>
      <B>Web Guidelines</B><BR><BR>
    </FONT></CENTER><FONT FACE = "LatinoPalSH">

```

**Diagram 6.2.1****Diagram 6.2.2**

There are several guidelines that can be followed when creating and developing Web Sites. The user needs to be familiar with the working of the Internet, several terms like Web Server, Web Pages and so on. Information about hosting a Web Site needs to be understood. These and more information has been segregated into sections and listed below. Clicking on any of these sections will lead to a detailed explanation of the section. <BR><BR><B>These are References a Section in Another Document.</B><BR><BR><UL>

<LI><A HREF = "Sections.html#SECTION1">Section 1</A>

<LI><A HREF = "Sections.html#SECTION2">Section 2</A>

<LI><A HREF = "Sections.html#SECTION3">Section 3</A>

</UL></FONT>

<BODY>

</HTML>

### Code for Sections.html

<HTML>

<HEAD><TITLE>Document With Targets For Links To Particular Locations</TITLE></HEAD>

<BODY Background = "../images/texture1.gif">

<A Name="SECTION1"><H2>SECTION 1 - INTRODUCTION</H2></A>

<CENTER><IMG.Src="../images/intro.gif"></CENTER><BR>

Internet has a growing importance in today's life. It provides us with a vast variety of information including educational stuff, Political comments, Current affairs, Technological Advancements, Social and Cultural information etc. Besides it offers mailing facilities and also provides, with facilities wherein a user can demand for anything and be sure that his requirements will be met.<P>With so much to give, its importance is flashing and more and more people are attracted towards this giant network.<P>Thus, developing information and hosting it on the Web is of prime importance in the current century.<BR><BR>

<A Name="SECTION2"><H2>SECTION 2 - VIEWERS</H2></A>

<CENTER><IMG Height=100 Src="../images/computer.gif" Widht=100></CENTER><BR>

Number of viewers isn't the only important criterion, however, so don't be swayed by thoughts of tons of folks visiting your Web page. The most important criterion is, who are these people who are seeing your ad.<BR><BR>

<A Name="SECTION3"><H2>SECTION 3 - LANGUAGES</H2></A>

<CENTER><IMG Height =150 Src="../images/corp.gif" Width =150></CENTER><BR>

The Language That Is Used To Develop Web Pages Is Called HTML Which Stands For <I><B>HyperText Markup Language</B></I> Which Is A Linking And Formatting Language, which Is The Only Language That Is Understood By a Browser.

</BODY>

</HTML>

## IMAGES AS HYPERLINKS

Just as text can act as a hyperlink, so also images can act as hyperlinks. Anything included within <A>...</A> tags becomes a Hotspot. Thus, an image can be made a Hotspot by enclosing an <IMG> tag within the <A>...</A> tags. The <IMG> tag places the image on the screen, and because the <IMG> tag is enclosed within the <A>...</A> tags, it becomes a Hotspot.

### Example:

<A HRef="details.htm"><IMG Src="mickey.gif"></A>

Here, the picture mickey.gif acts as a Hotspot, and navigates to a file **details.htm**.

## Image Maps

When a hyperlink is created on an image, clicking on any part of the image will lead to opening of the document specified in the `<A HRef ...>` tag. If the image is a large image and there is a need to link multiple documents to the same image, there has to be a technique that divides the image into multiple sections and allows linking of each section to a different document.

The technique that is implemented to achieve this is an **Image Map**. Image maps can be created and applied to an image so specific portions of the image can be linked to a different file/image.

Linked regions of an image map are called **hot regions** and each hot region is associated with a *filename.htm* document that will be loaded into the browser (*navigated to*) when the hot region is clicked.

Creating an image map is a two-step process:

### Step One:

Create an image map, i.e. divide the image into various areas. This is done using the `<MAP>` `</MAP>` tags. The `<MAP>` tag takes an attribute, **Name**, via which the map can be referenced in an HTML file.

### Syntax:

```
<MAP Name="map name">
```

Within the `<MAP>`. . . `</MAP>` tags the `<AREA>` tag is specified. This tag defines the specific region within the image. The `<AREA>` tag takes certain attributes. The attributes are:

<b>Shape</b>	The shape of a region can be one of the following: Rect, Circle, Polygon, Default
<b>Coords</b>	Each of the above shapes takes different coordinate parameters. A <b>Rectangle</b> will take four coordinates: <b>x1, y1, x2, y2</b> A <b>Circle</b> will take three coordinates: <b>centerx, centery</b> and <b>radius</b> . A <b>Polygon</b> will take three or more pairs of coordinates denoting a polygonal region. A <b>Default</b> shape will not take any parameter and it indicates the portion of the image not specified under any Area tag.
<b>HRef</b>	Takes the name of the .htm file that is linked to the particular area on the image.

Table 6.2

If specific areas within an image have to be linked to different documents, these areas will have to be identified on the image and linked to different documents. These areas can be in the shape of a rectangle, circle or a polygon. For each, coordinates need to be specific to mark an appropriate area on the image. For instance, a rectangle needs 2 points of specification, the upper left corner and the lower right corner. Consider the following image, which offers more knowledge about a company called SCT and provides two options.

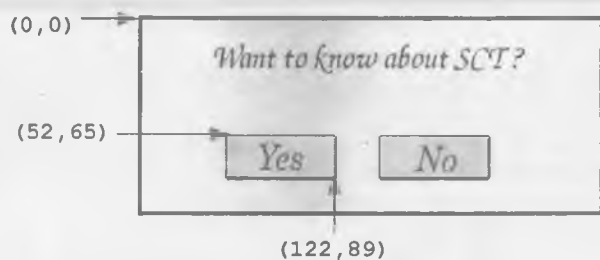


Diagram 6.3

Clicking on **Yes** will display information about SCT and clicking on **No** could lead back to a start page. Thus, area on the image that displays the buttons **Yes** and **No** need to be mapped to two different HTML files. These areas are in the shape of a rectangle and will be identified by means of top/left and bottom/right **coordinates**, i.e. **Yes** can be marked as (52,65,122,89).

Similarly, the coordinates of **No** can be obtained and marked.

**Example:**

```
<MAP Name="Sct_map">
  <AREA Shape="rect" Coords="52,65,122,89" HRef="sct.htm">
  <AREA Shape="rect" Coords="148,65,217,89" HRef="no.htm">
</MAP>
```

**Step Two:**

Deals with applying the image map to a particular image. For this purpose, the <IMG> tag takes an attribute called **UseMap** that takes the name of the image map as a value, and applies the map specifications to the respective image. The value is always preceded with the # sign.

**Syntax:**

```
<IMG UseMap = "#map_name">
```

**Example:**

```
<IMG Src="question.gif" UseMap="#Sct_map">
```

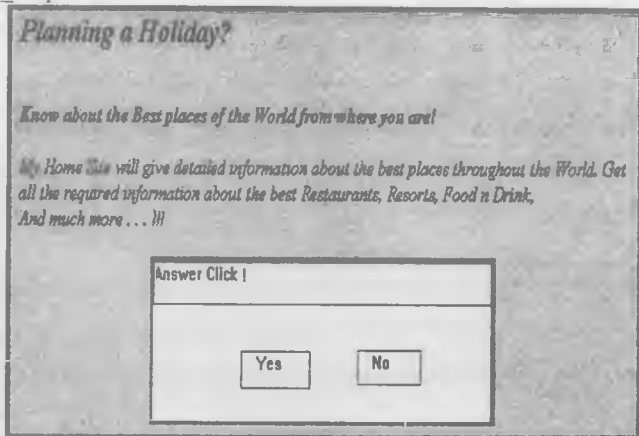
**Example 3:**

To create an HTML web page that offers an opportunity to get information about Travel & Tourism. Clicking on  will display the required information. Clicking on  displays another HTML file.

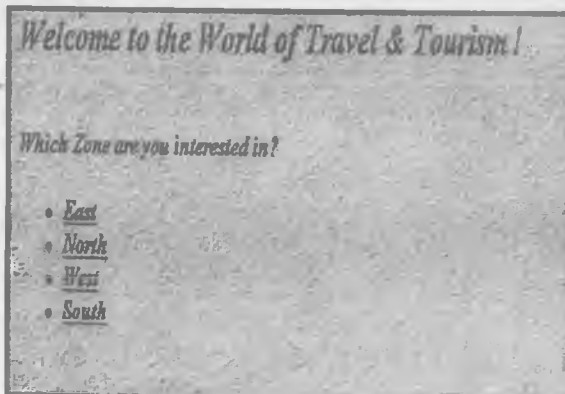
The startup HTML Page should be as shown in diagram 6.4.1.

Clicking on  will displays the output as shown in diagram 6.4.2.

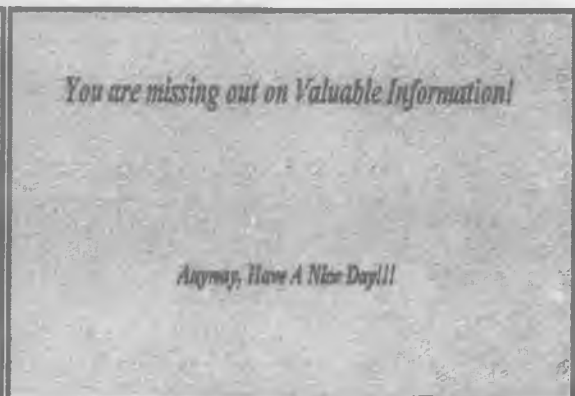
Clicking on  will display to the user the output as shown in diagram 6.4.3.



**Diagram 6.4.1**



**Diagram 6.4.2**



**Diagram 6.4.3**

**Code for ImgMap.html**

```
<HTML>
  <HEAD><TITLE>Using Image Maps!!</TITLE></HEAD>
  <BODY Background=" ../images/texture1.gif">
```

```

<MAP Name="Alert_Map">
  <AREA Shape="Rect" Coords="102,74,164,96" HRef="Travel.html">
  <AREA Shape="Rect" Coords="209,74,272,96" HRef="Missing.html">
</MAP><BR>
<H2><I>Planning a Holiday?</I></H2><BR><I><B>Know about the Best places of the World
from where you are!</I></B><P><I>My Home Site will give detailed information about the best
places throughout the World. Get all the required information about the best Restaurants, Resorts,
Food n Drink,<BR>And much more . . . !!!</I><BR><BR>
<CENTER><IMG Src=" ../images/alert.gif" UseMap="#Alert_Map"></CENTER>
</BODY>
</HTML>

```

**Code for Travel.html**

```

<HTML>
<HEAD><TITLE>Using Image Maps!!</Title></HEAD>
<BODY BackGround = "../images/texture1.gif"><BR>
  <H2><I> Welcome to the World of Travel & Tourism ! </I></H2><BR>
  <I><B>Which Zone are you interested in?<UL>
    <LI><A HRef="East.html"> East </A>
    <LI><A HRef="North.html"> North </A>
    <LI><A HRef="West.html"> West </A>
    <LI><A HRef="South.html"> South </A>
  </UL></B></I>
</BODY>
</HTML>

```

**Code for Missing.html**

```

<HTML>
<HEAD><TITLE>Using Image Maps!!</TITLE></HEAD>
<BODY Background=" ../images/texture1.gif"><CENTER><BR><BR>
  <H2><I>You are missing out on Valuable Information!</I></H2><BR><BR><BR><BR>
  <H3><I>Anyway, Have A Nice Day!!!</I></H3>
</CENTER></BODY>
</HTML>

```

**FOCUS**

Make use of Hyperlinks to provide navigation through the DEPL Web Site. Provide an index, each entry of the index leads to different blocks of information.

This means that the DEPL site information needs to be separated into multiple *filename.html* files. A startup file has to serve as the index. The DEPL is being broken up into five different HTML files described as follows:

File Name	Functionality
Index.html	<p>The first (startup) file. This file has following <b>4 Hyperlinks</b> which leads to specific information:</p> <ul style="list-style-type: none"> <li><input type="checkbox"/> Profile</li> <li><input type="checkbox"/> Barbed Wire</li> <li><input type="checkbox"/> Barbed Tape</li> <li><input type="checkbox"/> Animal Fencing</li> </ul> <p>Additionally, these hyperlinks are <b>images</b>, not text.</p>

Table 6.3

File Name	Functionality
Profile.html	Holds the company profile.
BWire.html	Holds a detailed description about the barbed wire products of the company.
BTape.html	Holds a detailed description about one the barbed tape products of the company.
AniFenc.html	Holds a detailed description about the animal fencing products of the company.

Table 6.3 (Continued)

Code for Index.html

```
<HTML>
<HEAD><TITLE>Delta Engineering Pvt. Ltd.</TITLE></HEAD>
<BODY Background=" ../images/pinkwhit.gif"><CENTER>
  <FONT Color="#008000" Face="Brush Script MT" Size=7>
    <IMG Align="Bottom" Src=" ../images/logo.gif">
    <I>Delta Engineering Pvt. Ltd.</I><HR>
  </FONT><FONT Color="#008000" Size=3><BR>
    <B><I>Delta Engineering Pvt. Ltd. is a specialist manufacturer of wire and wire products.
    DEPL established in Mumbai, India is ideally located for shipments to any part of the world.
    The main items of manufacture, in collaboration with Gulf Fencing Industry (GFI) include
    protector gabions and protector fencing systems.</I></B><P></FONT>
  <TABLE Width = 90%><TR>
    <TD Align=CENTER><A HRef="Profile.html"><IMG Alt="Profile" Border=0
    Height="35" Src=" ../images/profile.gif" Width="101"></A></TD>
    <TD Align=CENTER><A HRef="BWire.html"><IMG Alt="Barbed Wire" Border=0
    Height="35" Src=" ../images/conwire.gif" Width="101"></A></TD>
    <TD Align=CENTER><A HRef="BTape.html"><IMG Alt="Barbed Tapes" Border=0
    Height="35" Src=" ../images/contape.gif" Width="101"></A></TD>
    <TD ALIGN=CENTER><A Href="AniFenc.html"><IMG Alt="Fencing" Border=0
    Height="35" Src=" ../images/anifenc.gif" Width="101"></A></TD>
  </TR></TABLE>
</CENTER></BODY>
</HTML>
```

Output For Index.html:

As seen in the output shown in diagram 6.5.1, there are four pictures at the end of the screen. Each of these is actually a Hyperlink.

- ❑ Clicking on **Profile** opens a file called **Profile.html**.
- ❑ Clicking on **Concertina Barbed Wire** opens a file called **BWire.html**.
- ❑ Clicking on **Concertina Barbed Tape** opens a file called **BTape.html**.
- ❑ Clicking on **Animal Fencing** opens a file called **Anifenc.html**.



Diagram 6.5.1

The code for each of these HTML files along with their what these files should look like when viewed in a browser is given below.

## Code For Profile.html

```

<HTML>
<HEAD><TITLE> Delta Engineering Pvt. Ltd. </TITLE></HEAD>
<BODY Background=" ../images/pinkwhit.gif">
  <FONT Face="Brush Script MT" Size=7 Color="#008000"><CENTER><IMG Align=Bottom
  Src=" ../images/logo.gif"><I>Our Profile</I></CENTER></FONT><BR><HR Size=3>
  <FONT Color="#008000" Face="Times Roman" Size=3><CENTER><IMG Height="57"
  Src=" ../images/intro.gif" Width="185"><BR>
  <P>Delta Engineering Pvt. Ltd. is a specialist manufacturer of wire and wire products. DEPL
  established in Mumbai, India is ideally located for shipments to any part of the world. The main
  items of manufacture, in collaboration with Gulf Fencing Industry (GFI) include protector gabions
  and protector fencing systems.<P>Protector fencing systems offer a wide range of solutions to all
  security problems on any terrain and under extreme climatic conditions. The protector fencing
  system has been used extensively in Europe, America and the Far East, and this has helped
  formulate a package especially suited to Middle East requirements.<P>DEPL is equipped to offer
  comprehensive packages including design, material and installation.<P>Products strictly adhere to
  BS, ASTM and DIN international specifications. Quality control is guaranteed by independent
  laboratory test certificates from India.<P>Quality control is implemented without sacrificing
  economy and efficiency. DEPL is dedicated to technical services and problem solving.<P>DEPL
  is following quality methods and is accredited with ISO 9002.<P><IMG Height="57"
  Src=" ../images/planning.gif" Width="184"><BR>DEPL can provide planning support by offering
  design, technical specifications, drawings, foundation plans and installation instructions, together
  with a personalized service. The company's technical sales engineers keep in constant touch with
  all clients. Utmost importance is given to optimum design, versatility, durability and economy
  backed by the DEPL guarantee for work undertaken.</CENTER><FONT><BR><HR>
  <FONT Color="Green" Face="ZappedChancellorSH" Size="4">
    <P><SPACER Type="Horizontal" Size="20">
      Please forward any enquiries to enq_depl@bom2.vsnl.net.in
    <P><I>
      <SPACER Type="Horizontal" Size="120">
        <B>DELTA ENGINEERING PVT. LTD.</B><BR>
        <SPACER Type="Horizontal" Size="120">502, 5th Floor, Tejas Building,<BR>
        <SPACER Type="Horizontal" Size="120">Andheri (W), Mumbai<BR>
        <SPACER Type="Horizontal" Size="120">INDIA<BR>
        <SPACER Type="Horizontal" Size="120">Telephone : 91-022-8210050
      </I><P>
    </FONT><HR>
  </BODY>
</HTML>

```

## Output For Profile.html:

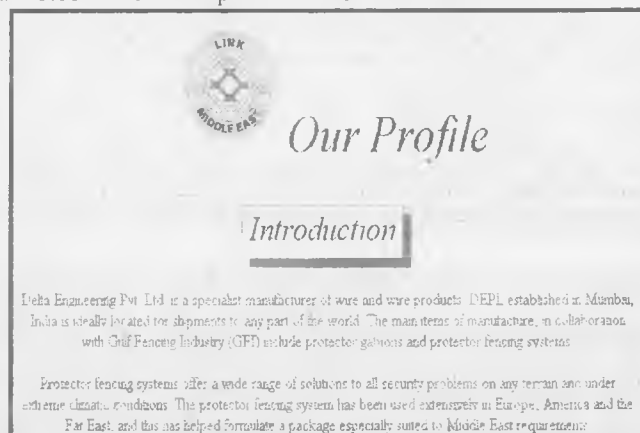


Diagram 6.5.2



## Code for BWire.htm

&lt;HTML&gt;

&lt;HEAD&gt;&lt;TITLE&gt; Delta Engineering Pvt. Ltd. &lt;/TITLE&gt;&lt;/HEAD&gt;

&lt;BODY Background=" ../images/pinkwhit.gif"&gt;

&lt;FONT Face="Brush Script MT" Size=7 Color="#008000"&gt;&lt;CENTER&gt;&lt;I&gt;Barbed Wires&lt;/I&gt;&lt;/CENTER&gt;&lt;/FONT&gt;&lt;BR&gt;&lt;HR Size=3&gt;

&lt;TABLE CellPadding=20&gt;&lt;TR&gt;

&lt;TD&gt;&lt;IMG Align=Bottom Border=2 Height="300" Src=" ../images/barbed1.jpg" Width="250"&gt;&lt;/TD&gt;

&lt;TD&gt;&lt;FONT Face="Comic Sans MS" Size=2 Color="#008000"&gt;&lt;B&gt;Concertina Barbed Wire in roll form is used in high security areas to deter trespassing men and animals.&lt;P&gt;The effectiveness of this material is proved as it has been in use for more than 75 years during war and peace.&lt;/B&gt;&lt;/FONT&gt;&lt;/TD&gt;

&lt;/TR&gt;&lt;/TABLE&gt;

&lt;FONT Face="Comic Sans MS" Size=2 Color="#008000"&gt;&lt;OL&gt;

<P><B><I><LI>Line Wire:</I></B><BR>This is made up of 3.05mm diameter high carbon steel wire, with a tensile strength of 170 to 180 kg/mm<sup>2</sup>. The wire is drawn and dressed in such a manner that the coils formed will fall naturally into the specified diameter without forming a figure eight. Line wire is heavy hot dipped galvanized with minimum inc coating of 185 gm/mm<sup>2</sup>.<P><B><I><LI>Barbed Wire:</I></B><BR>This is 2.00mm diameter bright mild steel wire with a tensile strength of 38 to 55kg/mm<sup>2</sup> conforming to BS 1052. The wire is hot dipped galvanized with a minimum zinc coating of 20 to 50 gm/mm<sup>2</sup>. The barbs are formed with four points. Spacing between the centers of the barbs of every 70mm along the length of the wire. The barbs are firmly secured by indentations made on the wire, so that the barbs do not rotate or slide along the wire. The four points of the barbs are formed at the right angles to one another and project outwards approximately 12mm from the center of the wire.

&lt;P&gt;&lt;B&gt;&lt;I&gt;&lt;LI&gt; Carrying Handles:&lt;/I&gt;&lt;/B&gt;&lt;BR&gt;Handles are made of mild steel wire of 3.55mm diameter and are attached to the outer turn of the coil on each side.

&lt;/OL&gt;&lt;/FONT&gt;&lt;P&gt;&lt;HR&gt;

&lt;/BODY&gt;

&lt;/HTML&gt;



Diagram 6.5.3

## Output For BWire.html:

## Code for BTape.html

&lt;HTML&gt;

&lt;HEAD&gt;&lt;TITLE&gt; Delta Engineering Pvt. Ltd. &lt;/TITLE&gt;&lt;/HEAD&gt;

&lt;BODY Background=" ../images/pinkwhit.gif"&gt;

&lt;FONT Face="Brush Script MT" Size=7 Color="#008000"&gt;&lt;CENTER&gt;&lt;I&gt;Barbed Tapes&lt;/I&gt;&lt;/CENTER&gt;&lt;/FONT&gt;&lt;BR&gt;&lt;HR Size=3&gt;

```

<TABLE CellPadding=20><TR>
  <TD><IMG Align=Bottom Border=2 Height="300" Src="../images/barbed.jpg"
    Width="280"></TD>
  <TD><FONT Face="Comic Sans MS" Size=2 Color="#008000"><B>Barbed Tapes are used
    as psychological and physical deterrent against intrusion by personnel and animals. Barbed
    tape barrier systems are more vicious and difficult to tamper with, providing superior
    perimeter security. Some of the users are military installations, nuclear energy sites,
    maximum security prisons, various petroleum installations, tank farms and other important
    industrial facilities.</B></FONT></TD>
</TR></TABLE>
<FONT Face="Comic Sans MS" Size=2 Color="#008000"><UL>
  <P><LI><DT><B><I>Short Blade Barbed Tape (SBBT):</I></B><BR>
  <DD>To give the maximum tensile strength at the time of the breach hard drawn steel cord is
    used (ES1). For camouflage purposes, a coaltar coating can be applied over the entire surface
    (ES-1), and to attain the high degree of rust resistance, the core wire is made of galvanized
    steel (ES-2). Any of the above three options are available to meet your specific requirements.
  <P><LI><DT><B><I>Medium Blade Barbed Tape (MBBT):</I></B><BR>
  <DD>The specification of the "ivory" tape is similar to those of ES types. The blades are
    sharper and have a greater pricking capacity. The life of this tape is more than three times
    longer than those of ES types. Therefore, "ivory" tape is a more effective choice in terrain
    which has heavy rain or in coastal areas. The diameter will be similar to ES type.
  <P><LI><DT><B><I>Long Blade Barbed Tape (LBBT):</I></B><BR>
  <DD>This is the most effective psychological and physical deterrent, ever made as a barrier
    obstacle. It is available in authentic stainless steel (SUS 430) and the core wire can be
    fabricated from either galvanized carbon steel or stainless steel.
</UL></FONT><P><HR>

```

&lt;/BODY&gt;

&lt;/HTML&gt;

Output For Btape.html:

Code for AniFenc.html

&lt;HTML&gt;

```

<HEAD><TITLE> Delta Engineering
Pvt. Ltd. </TITLE></HEAD>

```

```

<BODY Background="../images/pinkwhit.gif">

```

```

  <FONT Face="Brush Script MT" Size=7 Color="#008000"><CENTER><I>Animal
  Fencing</I></CENTER></FONT><BR><HR Size=3>

```

```

  <TABLE CellPadding=20><TR>

```

```

    <TD><IMG Align=Bottom Border=2 Height="240" Src="../images/fence1.jpg"
      Width="200"></TD>

```

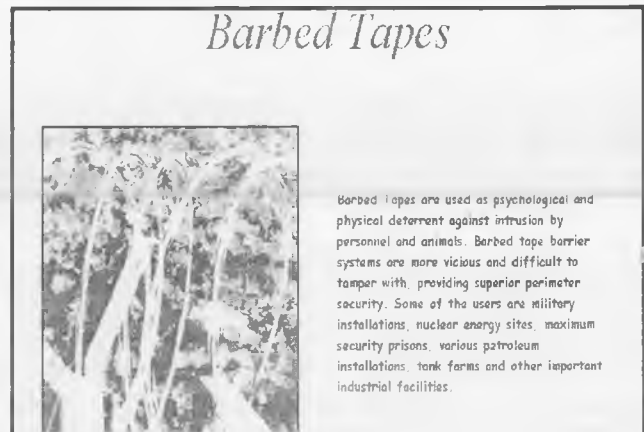


Diagram 6.5.4

```
<TD><FONT Face="Comic Sans MS" Size=2 Color="#008000"><B>DEPL's animal fencing
system is mainly used as an anti-intrusion barrier against any farm and other animals.<P>The
most common use is on highways where vast distances are covered at a very economical
cost.<P>Animal fencing can also be used to enclose areas such as farms, forest areas and
national parks, where security is not crucial.<P>This system is easy to install and can be
erected in a comparatively short time.</B></FONT></TD>
```

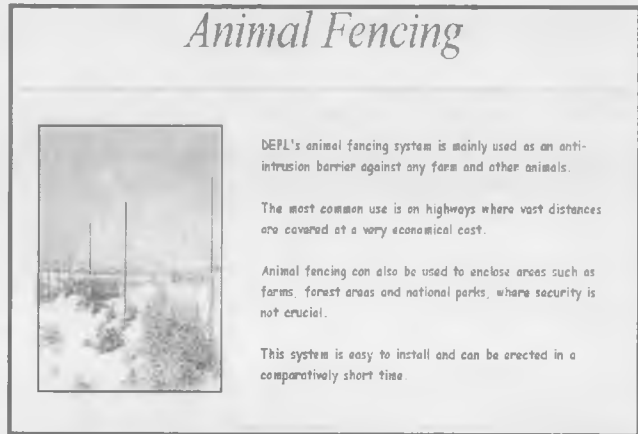
```
</TR></TABLE><HR>
```

```
</BODY>
```

```
</HTML>
```

### Output For AniFenc.html:

The technique of using hyperlinks allows navigation to specific information in a web site. If this technique is incorporated into a web site the facility of navigating to specific information in the site is available. This provides some 'Reader Control' over the site. It is now possible for a reader to view only selective information, without scanning through a large amount of textual matter.



**Diagram 6.5.5**

There is still one obvious drawback. Every time a reader clicks on a hyperlink, a new document is opened over the current document. That is, the document containing the Index is hidden, and the reader has to compulsorily click on the browser's **Back** button to reach the Index page.

For a single one level of navigation, it might be feasible to click on the **Back** button. But consider a scenario where the index, opens **File1** (via a hyperlink). Further, File1 consists of a hyperlink, which leads to **File2**. Now, to go back to the Index, the **Back** button will have to be clicked twice. This complexity will increase with the number 'drill down' nested hyperlinks, thereby the reader may lose sight of the index.

To overcome this problem, one approach would be to let the index file always be visible on one side of the screen, no matter what hyperlink is clicked on. In such a case, the remaining part of the screen will display the document asked for.

In other words, the browser screen itself needs to be divided two unique, recognizable sections, one always showing the index, and the other changing its content dynamically, depending on the hyperlink clicked on.

When the browser screen is divided into more than one unique, (HTML) recognizable, section each such section is called a **Frame**.

The following chapter explains how the browser screen can be divided into frames. How hyperlinks in one frame can open different HTML documents in another frame.

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. \_\_\_\_\_ allows linking to other documents.
2. The \_\_\_\_\_ color is the default color of a hyperlink.
3. Hyperlinks can be of two types: Links to an \_\_\_\_\_ document or an \_\_\_\_\_ document.
4. \_\_\_\_\_ enable a jump to a particular location in a document.
5. The name of the location has to be preceded by a \_\_\_\_\_ - \_\_\_\_\_ symbol.
6. Linked regions of image map are called \_\_\_\_\_.
7. An image map can take any of the following four shapes : \_\_\_\_\_, \_\_\_\_\_, \_\_\_\_\_ & \_\_\_\_\_.

### TRUE OR FALSE

8. Anchors target links to the beginning of the document.
9. The color of the links can be changed in the <Body> tag.
10. A filename always has to be mentioned before the # symbol in the HREF attribute of a link.

## HANDS ON EXERCISES

1. Create a document with two links to an external document. The first link should lead to the beginning of the external document. The second link should lead to a particular section in the external document.

In the external document specify a link that will lead to a particular section within it.

### **Text Content:**

Welcome to our homepage

This page has links to the website of **ABC Lever Inc.**

For further information click on any of the following:

- About ABC Lever Inc.
- Contact information

About us

---

ABC Lever inc. is a conglomerate that has interests ranging from bodycare products to toilet soaps.

A couple of years ago we entered the frozen food industry through mergers and acquisitions.

Last year we started our plant to manufacture salt and this year it is wheat flour.

Our current turnover is about Rs. 7500 cr and by the next decade we are looking at a target of 15000 cr.

---

Contact Us

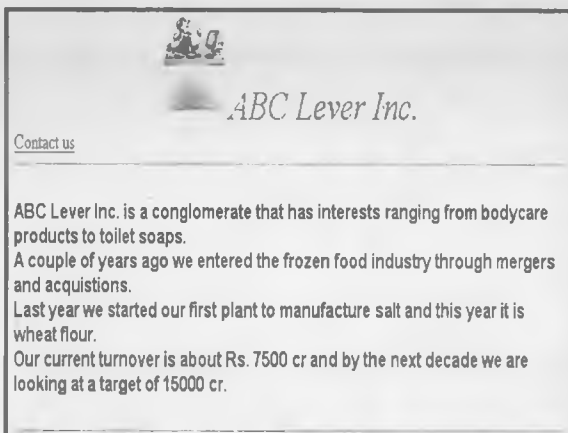
You can contact us at the following address:

ABC Lever Inc.  
 101 Maker Chambers III,  
 Nariman Point,  
 Mumbai - 21  
 Tel. 2102011

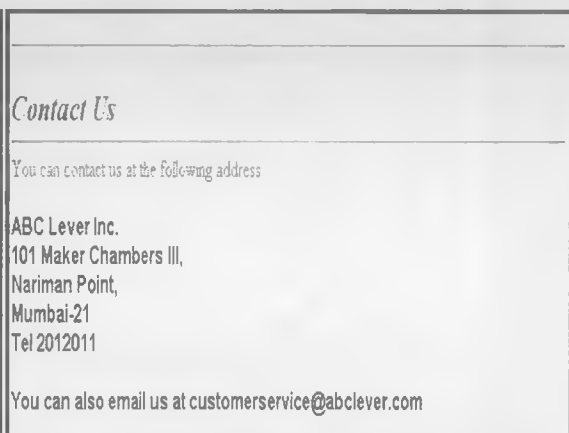
You can also email us at [customersservice@abclever.com](mailto:customersservice@abclever.com)



**Diagram 6.6.1:** Screen 1 of Hands on Exercise.



**Diagram 6.6.2:** Screen 2 of the Hands on Exercise.



**Diagram 6.6.3:** Screen 3 of the Hands on Exercise.

## 7. FRAMES

### INTRODUCTION TO FRAMES

Until now each web page when opened takes over the entire browser screen. The browser screen could not be split into separate (unique) sections, showing different but related information.

The HTML tags that divide a browser screen into two or more HTML recognizable unique regions is the `<FRAMESET>` `</FRAMESET>` tags. Each unique region is called a frame. Each frame can be loaded with a different document and hence, allow multiple HTML documents to be seen concurrently.

The HTML frame is a powerful feature that enables a web page to be broken into different unique sections that, although related, operate independently of each other.

#### The `<FRAMESET>` Tag

The splitting of a browser screen into frames is accomplished with the `<FRAMESET>` and `</FRAMESET>` tags embedded into the HTML document. The `<FRAMESET>` . . . `</FRAMESET>` tags require one of the following two attributes depending on whether the screen has to be divided into rows or columns.

<b>Rows</b>	This attribute is used to divide the screen into multiple rows. It can be set equal to a list of values. Depending on the required size of each row. The values can be: <ul style="list-style-type: none"><li><input type="checkbox"/> A number of pixels</li><li><input type="checkbox"/> Expressed as a percentage of the screen resolution</li><li><input type="checkbox"/> The symbol *, which indicates <i>the remaining space</i>.</li></ul>
<b>Cols</b>	This attribute is used to divide the screen into multiple columns. It can be set equal to a list of values. Depending on the required size of each column. The values can be: <ul style="list-style-type: none"><li><input type="checkbox"/> A number of pixels</li><li><input type="checkbox"/> Expressed as a percentage of the screen resolution</li><li><input type="checkbox"/> The symbol *, which indicates <i>the remaining space</i>.</li></ul>

Table 7.1

#### Example:

```
<FRAMESET Rows="33%,33%,33%"> -- Divides the browser screen into 3 equal Horizontal
    sections.
  <FRAMESET Cols="50%,50%"> -- Splits the 1st Horizontal Section into 2 equal Vertical
    sections.
  </FRAMESET>
  <FRAMESET Cols="50%,50%"> -- Splits the 2nd Horizontal section into 2 equal Vertical
    sections
  </FRAMESET>
</FRAMESET>
```

#### The `<FRAME>` Tag

Once the browser screen is divided into rows (Horizontal Sections) and columns (Vertical Sections), each unique section defined can be loaded with different HTML documents. This is achieved by using the `<FRAME>` tag, which takes in the following attributes:

<b>SRC="url"</b>	Indicates the URL of the document to be loaded into the frame.
<b>MarginHeight="n"</b>	Specifies the amount of white space to be left at the top and bottom of the frame.
<b>MarginWidth="n"</b>	Specifies the amount of white space to be left along the sides of the frame
<b>Name="name"</b>	Gives the frame a unique name so it can be targeted by other documents. The name given must begin with an Alphanumeric character
<b>Noresize</b>	Disables the frames resizing capability.
<b>Scrolling</b>	Controls the appearance of horizontal and vertical scrollbars in a frame. This takes the values YES / NO /AUTO.

Table 7.2

**Example 1:**

<HTML>

<FRAMESET Rows = "30%. \*">

-- Divides the screen into 2 rows, one occupying 30% of the screen, and the other occupying the remaining space, i.e. 70% of the screen.

<FRAMESET Cols = "50%, 50%">

-- Divides the 1st row into 2 equal columns, each 50% of the screen.

<FRAME Src="File1.html">

-- Loads the 1st frame with **file1.html**.

<FRAME Src="File2.html">

-- Loads the 2nd frame with **file2.html**.

</FRAMESET>

<FRAMESET Cols="50%, 50%">

-- Divides the 2nd row into 2 equal columns, 50% of the screen

<FRAME Src="File3.html">

-- Loads the 1st frame with **file3.html**.

<FRAME Src="File4.html">

-- Loads the 2nd frame with **file4.html**.

</FRAMESET>

</FRAMESET>

</HTML>

**Output For Example 1:**

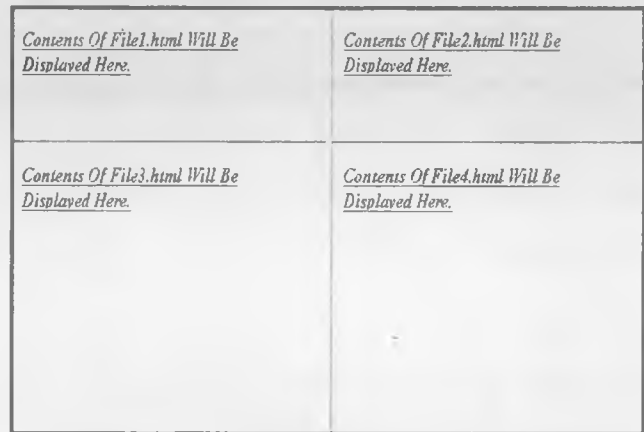


Diagram 7.1

**Targeting Named Frames**

Whenever a hyperlink, which loads a document in a frame is created, the file referenced in the hyperlink will be opened and will replace the current document that is in the frame.

In a situation where the new document needs to be opened in a different frame while keeping the document from which the new document was navigated open in a different frame, a simple HTML coding technique must be used.

Since the hyperlink must open an HTML file in another frame, the frame in which the HTML file is to be opened needs to be named. This is done by using the NAME attribute of the <FRAME> ... </FRAME> tags. The NAME takes one parameter, which is its frame name.

The hyperlink tag will have to be supplied with the following information

1. The *filename.htm* file that has to be opened (*navigated to*).
2. The name of the frame where the *filename.htm* file has to be opened.

The attribute, via which the frame name is specified is the **Target** attribute, which is a part of the `<A>...</A>` tag. This information is given as:

```
Target = "<Frame_Name>"
```

The attribute, via which the HTML file name is specified is the **HRef** attribute which is a part of the `<A>...</A>` tag. This information is given as:

```
<A HRef="index.html" Target="Main">Visit us</A>
```

#### Example:

##### Frame Identification:

```
<FRAMESET Cols = 30%, 70%>
  <FRAME Name="Part">
  <FRAME Name="Main">
</FRAMESET>
```

The above command will divide the browser screen into two vertical frames the first frame called **Part** that will occupy 30% of the browser area and the second frame called **Main** will occupy 70% of the browser area.

##### Hyperlink Specification:

```
<A HRef="Index.html" Target="Main">Visit us</A>
```

Here, an HTML file called **Index.html** is loaded into the frame named **Main** when the hyperlink Visit us is clicked.

### Note



While specifying the name of the target frame in the TARGET attribute, the case must be same as specified in the NAME attribute of the `<A>` `</A>` tag.

#### Example 2:

The following example divides the browser screen into 3 frames. The need is to give information about SCT staff. The startup file is **frames.html**, which loads three different documents in the three different frames. These documents are **header.html**, **sctfamil.html** and **desc.html**.

##### Code Listing For frames.html

```
<HTML>
  <FRAMESET ROWS = "70, *">
    <FRAME Src="header.html" MarginHeight=0 MarginWidth=0 Name="FRAME1">
    <FRAMESET Cols="35%, *">
      <FRAME Src="sctfamil.html" Name="FRAME2">
      <FRAME Src="desc.html" Name="FRAME3">
    </FRAMESET>
  </FRAMESET>
</HTML>
```

### Note



When `<FRAMESET>` is being coded within an HTML document, the `<BODY>` `</BODY>` tags are not used.



**Code Listing For header.html**

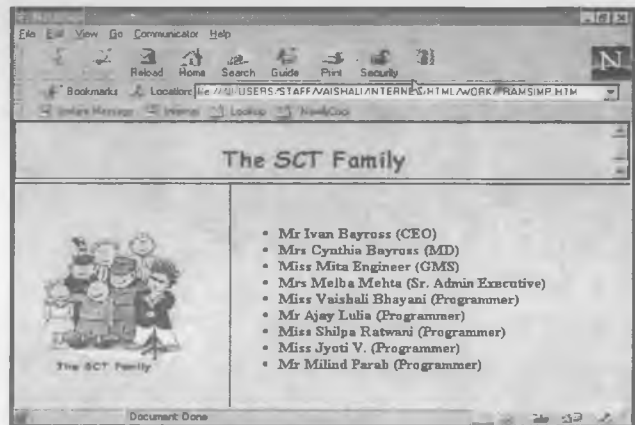
```
<HTML>
  <BODY Background = "../images/texture1.gif">
    <FONT Face = "Comic Sans MS"><BR>
    <CENTER><H2>The SCT Family</H2></CENTER>
  </BODY>
</HTML>
```

**Code Listing for sctfamil.html**

```
<HTML>
  <BODY Background="../images/texture1.gif"><BR><BR>
  <CENTER><IMG Height=175 Src="../images/sctfamil.gif" Width=150></CENTER>
</BODY>
</HTML>
```

**Code Listing for desc.html**

```
<HTML>
  <BODY Background="../images/texture1.gif"><BR><BR>
  <B><UL>
    <LI>Mr Ivan Bayross (CEO)
    <LI>Mrs Cynthia Bayross (MD)
    <LI>Miss Mita Engineer (GMS)
    <LI>Mrs Melba Mehta (Sr. Admin Executive)
    <LI>Miss Vaishali Bhayani (Programmer)
    <LI>Mr Ajay Lulia (Programmer)
    <LI>Miss Shilpa Ratwani (Programmer)
    <LI>Miss Jyoti V. (HR Manager)
    <LI>Mr Milind Parab (Marketing Executive)
  </UL></B>
</BODY>
</HTML>
```

**Output For Example 2:****Diagram 7.2****FOCUS**

The DEPL site will now have a starting page as a brief introduction of the company, consisting of Logo, Name and what the company does in a few words.

The starting page should provide hyperlinks, which give more detailed information about a topic. The Company Logo, Name and the site index must always be visible when the site is navigated through.

This is achieved by the use of frames. Various HTML files required are as listed in table 7.3:

File Name	Functionality
Frames.htm	This file divides the screen into 3 frames as follows: <ul style="list-style-type: none"> <li><input type="checkbox"/> One frame on top, loaded with the file - Header.htm</li> <li><input type="checkbox"/> Second frame on the left, loaded with the file - <i>Index.htm</i></li> <li><input type="checkbox"/> Third Frame occupying the remaining area on the screen, initially loaded with the file - <i>Intro.htm</i></li> </ul>
Header.htm	Displays Header Information like company name and logo.
Index.htm	Provides an Index, consisting of the following 4 <i>Image Hyperlinks</i> : <ul style="list-style-type: none"> <li><input type="checkbox"/> Profile</li> <li><input type="checkbox"/> Barbed Wire</li> <li><input type="checkbox"/> Barbed Tape</li> <li><input type="checkbox"/> Animal Fencing</li> </ul> These Hyperlinks, when clicked, open different files in the third frame on the screen.
Profile.htm	Gives the company profile
BWire.htm	Gives a description about the product - Barbed Wires
BTape.htm	Gives a description about the product - Barbed Tapes
AniFenc.htm	Gives a description about the product - Animal Fencing

Table 7.3

#### Code for Frames.htm

```
<HTML><FRAMESET Rows="30%,*" Framespacing="0">
  <FRAME Name="Header" Src="Header.html" FrameBorder="0" Scrolling=NO>
  <FRAMESET Cols="25%,*">
    <FRAME Name="Index" Src="Index.html" FrameBorder="0" Scrolling= NO>
    <FRAME Name="Details" Src="Intro.html" FrameBorder="0">
  </FRAMESET>
</FRAMESET></HTML>
```

#### Code listing for Header.html:

```
<HTML><BODY Background="../images/pinkwhit.gif">
  <CENTER><FONT Color="#008000" Face="Brush Script MT" Size=6>
    <IMG Align=Middle Src="../images/logo.gif"><SPACER Size=30>
    <I>Delta Engineering Pvt. Ltd.</I>
  </FONT><CENTER>
</BODY></HTML>
```

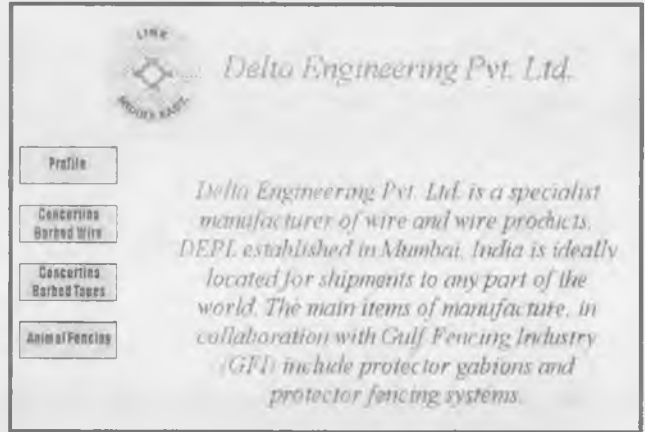
#### Code listing for Index.html:

```
<HTML>
  <BODY Background="../images/pinkwhit.gif">
    <A HRef="Profile.html" Target="Details"><IMG Alt="Profile" Border=0 Height="35"
      Src="../images/profile.gif" Width="101"></A><BR><BR>
    <A HRef="BWire.html" Target="Details"><IMG Alt="Barbed Wire" Border=0 Height="35"
      Src="../images/conwire.gif" Width="101"></A><BR><BR>
    <A HRef="BTape.html" Target="Details"><IMG Alt="Barbed Tapes" Border=0 Height="35"
      Src="../images/contape.gif" Width="101"></A><BR><BR>
    <A Href="AniFenc.html" Target="Details"><IMG Alt="Fencing" Border=0 Height="35"
      Src="../images/anifenc.gif" Width="101"></A>
  </BODY>
</HTML>
```

**Code listing for Intro.html:**

```
<HTML>
  <BODY Background="../images/pinkwhit.gif">
    <FONT Color="#008000" Face="Brush Script MT" Size=5><BR><CENTER><I>
      Delta Engineering Pvt. Ltd. is a specialist manufacturer of wire and wire products. DEPL
      established in Mumbai, India is ideally located for shipments to any part of the world. The
      main items of manufacture, in collaboration with Gulf Fencing Industry (GFI) include
      protector gabions and protector fencing systems.
    </I></CENTER></FONT>
  </BODY>
</HTML>
```

**Output For Frame.html:**



**Diagram 7.3.1**

**Code For Profile.html:**

```
<HTML>
  <HEAD><TITLE> Delta Engineering
    Pvt. Ltd. </TITLE></HEAD>
  <BODY Background="../images/pinkwhit.gif">
    <FONT Face="Brush Script MT" Size=7 Color="#008000"><I>Our Profile</I></FONT>
    <BR><HR Size=3>
    <FONT Color="#008000" Face="Times Roman" Size=3><CENTER><IMG Height="40"
      Src="../images/intro.gif" Width="120"><BR>
    <P>Delta Engineering Pvt. Ltd. is a specialist manufacturer of wire and wire products. DEPL
    established in Mumbai, India is ideally located for shipments to any part of the world. The main
    items of manufacture, in collaboration with Gulf Fencing Industry (GFI) include protector gabions
    and protector fencing systems.<P>Protector fencing systems offer a wide range of solutions to all
    security problems on any terrain and under extreme climatic conditions. The protector fencing
    system has been used extensively in Europe, America and the Far East, and this has helped
    formulate a package especially suited to Middle East requirements.<P>DEPL is equipped to offer
    comprehensive packages including design, material and installation.<P>Products strictly adhere to
    BS, ASTM and DIN international specifications. Quality control is guaranteed by independent
    laboratory test certificates from India.<P>Quality control is implemented without sacrificing
    economy and efficiency. DEPL is dedicated to technical services and problem solving.<P>DEPL
    is following quality methods and is accredited with ISO 9002.<P><IMG Height="40"
    Src="../images/planning.gif" Width="120"><BR>DEPL can provide planning support by offering
    design, technical specifications, drawings, foundation plans and installation instructions, together
    with a personalized service. The company's technical sales engineers keep in constant touch with
    all clients. Utmost importance is given to optimum design, versatility, durability and economy
    backed by the DEPL guarantee for work undertaken.</CENTER><FONT><BR><HR>
    <FONT Color="Green" Face="ZappedChancellorSH" Size="4">
      <P><SPACER Type="Horizontal" Size="20">
        Please forward any enquiries to enq\_depl@bom2.vsnl.net.in
```

```

<P><I>
  <SPACER Type="Horizontal" Size="120">
    <B>DELTA ENGINEERING PVT. LTD.</B><BR>
    <SPACER Type="Horizontal" Size="120">502, 5th Floor, Tejas Building,<BR>
    <SPACER Type="Horizontal" Size="120">Andheri (W), Mumbai<BR>
    <SPACER Type="Horizontal" Size="120">INDIA<BR>
    <SPACER Type="Horizontal" Size="120">Telephone : 91-022-8210050
  </I></P>
</FONT><HR>
</BODY>
</HTML>

```

**Output For Profile.html:****Diagram 7.3.2****Code for BWire.htm**

```

<HTML>
<HEAD><TITLE> Delta Engineering
  Pvt. Ltd. </TITLE></HEAD>
<BODY Background="../images/pinkwhit.gif">
  <FONT Face="Brush Script MT" Size=7 Color="#008000"><I>Barbed Wires</I></CENTER>
  <FONT><BR><HR Size=3>
  <TABLE CellPadding=10><TR>
    <TD><IMG Align=Bottom Border=2 Height="200" Src="../images/barbed1.jpg"
      Width="160"></TD>
    <TD><FONT Face="Comic Sans MS" Size=2 Color="#008000"><B>Concertina Barbed
      Wire in roll form is used in high security areas to deter trespassing men and animals.<P>The
      effectiveness of this material is proved as it has been in use for more than 75 years during war
      and peace.</B></FONT></TD>
  </TR></TABLE>
  <FONT Face="Comic Sans MS" Size=2 Color="#008000"><OL>
    <P><B><I><LI>Line Wire:</I></B><BR>This is made up of 3.05mm diameter high carbon
    steel wire, with a tensile strength of 170 to 180 kg/mm2. The wire is drawn and dressed in
    such a manner that the coils formed will fall naturally into the specified diameter without
    forming a figure eight. Line wire is heavy hot dipped galvanized with minimum inc coating of
    185 gm/mm2.
    <P><B><I><LI>Barbed Wire:</B></I><BR>This is 2.00mm diameter bright mild steel wire
    with a tensile strength of 38 to 55kg/mm2 conforming to BS 1052. The wire is hot dipped
    galvanized with a minimum zinc coating of 20 to 50 gm/mm2. The barbs are formed with four
    points. Spacing between the centers of the barbs of every 70mm along the length of the wire.
    The barbs are firmly secured by indentations made on the wire, so that the barbs do not rotate
    or slide along the wire. The four points of the barbs are formed at the right angles to one
    another and project outwards approximately 12mm from the center of the wire.
    <P><B><I><LI> Carrying Handles:</I></B><BR>Handles are made of mild steel wire of
    3.55mm diameter and are attached to the outer turn of the coil on each side.

```

```
</OL></FONT><P><HR>
</BODY>
</HTML>
```

Output For BWire.html:

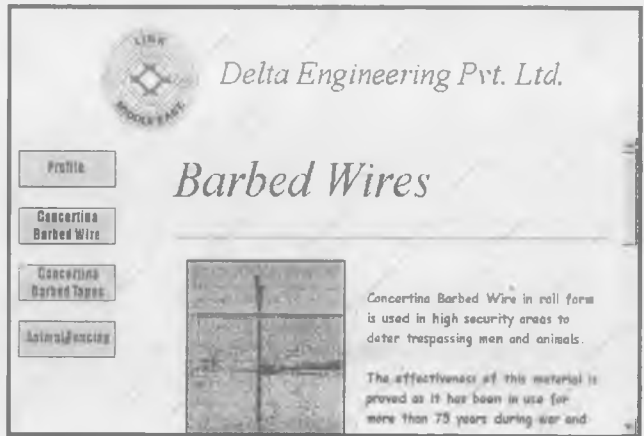


Diagram 7.3.3

Code for BTape.htm

```
<HTML>
<HEAD><TITLE>Delta Engineering Pvt. Ltd.</TITLE></HEAD>
<BODY Background=" ../images/pinkwhit.gif">
  <FONT Face="Brush Script MT" Size=7 Color="#008000"><I>Barbed Tapes</I></FONT>
  <BR><HR Size=3>
  <TABLE CellPadding=20><TR>
    <TD><IMG Align=Bottom Border=2 Height="350" Src=" ../images/barbed.jpg"
      Width="200"></TD>
    <TD><FONT Face="Comic Sans MS" Size=2 Color="#008000"><B>Barbed Tapes are used
      as psychological and physical deterrent against intrusion by personnel and animals. Barbed
      tape barrier systems are more vicious and difficult to tamper with, providing superior
      perimeter security. Some of the users are military installations, nuclear energy sites,
      maximum security prisons, various petroleum installations, tank farms and other important
      industrial facilities.</B></FONT></TD>
  </TR></TABLE>
  <FONT Face="Comic Sans MS" Size=2 Color="#008000"><UL>
    <P><LI><DT><B><I>Short Blade Barbed Tape (SBBT):</I></B><BR>
    <DD>To give the maximum tensile strength at the time of the breach hard drawn steel cord is
    used (ES1). For camouflage purposes, a coaltar coating can be applied over the entire surface
    (ES-1), and to attain the high degree of rust resistance, the core wire is made of galvanized
    steel (ES-2). Any of the above three options are available to meet your specific requirements.
    <P><LI><DT><B><I>Medium Blade Barbed Tape (MBBT):</I></B><BR>
    <DD>The specification of the "ivory" tape is similar to those of ES types. The blades are
    sharper and have a greater pricking capacity. The life of this tape is more than three times
    longer than those of ES types. Therefore, "ivory" tape is a more effective choice in terrain
    which has heavy rain or in coastal areas. The diameter will be similar to ES type.
    <P><LI><DT><B><I>Long Blade Barbed Tape (LBBT):</I></B><BR>
    <DD>This is the most effective psychological and physical deterrent, ever made as a barrier
    obstacle. It is available in authentic stainless steel (SUS 430) and the core wire can be
    fabricated from either galvanized carbon steel or stainless steel.
  </UL></FONT><P><HR>
</BODY>
</HTML>
```

Output For BTape.html:



Diagram 7.3.4

Code for Anifenc.htm

```

<HTML>
<HEAD><TITLE> Delta Engineering
Pvt. Ltd. </TITLE></HEAD>
<BODY Background=" ../images/pinkwhit.gif">
  <FONT Face="Brush Script MT" Size=7 Color="#008000">
    <I>Animal Fencing</I></FONT><BR><HR Size=3>
  <TABLE CellPadding=20><TR>
    <TD><IMG Align=Bottom Border=2 Height="240" Src=" ../images/fence1.jpg"
    Width="200"></TD>
    <TD><FONT Face="Comic Sans MS" Size=2 Color="#008000"><B>DEPL's animal fencing
    system is mainly used as an anti-intrusion barrier against any farm and other animals.<P>The
    most common use is on highways where vast distances are covered at a very economical
    cost.<P>Animal fencing can also be used to enclose areas such as farms, forest areas and
    national parks, where security is not crucial.<P>This system is easy to install and can be
    erected in a comparatively short time.</B></FONT></TD>
  </TR></TABLE><HR>
</BODY>
</HTML>

```

Output:

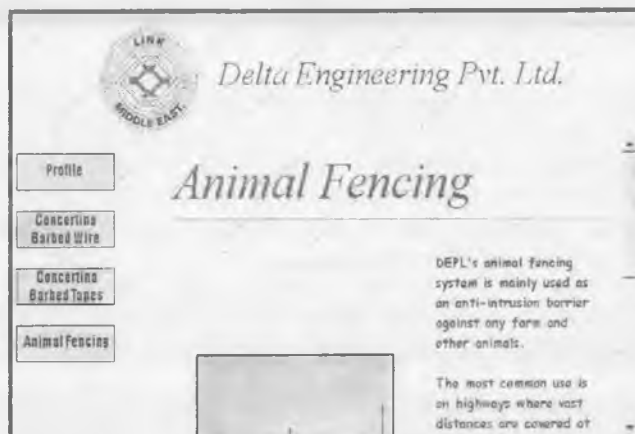


Diagram 7.3.5

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. \_\_\_\_\_ enable a Web page to be broken into different sections.
2. The splitting of a page into frames can be accomplished by using the \_\_\_\_\_ tag.
3. Once the screen is divided into different sections, each section can be loaded with a different HTML document using the \_\_\_\_\_ tag.
4. The \_\_\_\_\_ attribute of the <FRAME> tag contains the URL of the document to be loaded into the frame.
5. The \_\_\_\_\_ attribute of the <FRAME> tag disables the user's ability to resize the frame.
6. The \_\_\_\_\_ attribute controls the appearance of horizontal and vertical scrollbars in a frame.
7. A frame is identified using the \_\_\_\_\_ attribute of the <FRAME> tag.

### TRUE OR FALSE

8. The browser window cannot be split into separate little sections, showing different but related information.
9. By default, a user cannot resize a frame.
10. The TARGET attribute in the <A>...</A> tag specifies the name of the Target frame.

## HANDS ON EXERCISES

1. Create a specimen of a corporate web page. Divide the browser screen into two frames. The frame on the left will be a menu consisting of hyperlinks. Clicking on any one of these links will lead to a new page, which must open in the target frame, which is on the right hand side.
2. Create two links, the first link that will open a page that displays the company profile, its business and its products. The second link will display the contact address of the company.

### **Text Content:**

Welcome to our homepage

This page has links to the website of **ABC Lever Inc.**

For further information click on any of the following:

- About ABC Lever Inc.
- Contact information

About us

---

ABC Lever inc. is a conglomerate that has interests ranging from bodycare products to toilet soaps.

A couple of years ago we entered the frozen food industry through mergers and acquisitions.

Last year we started our plant to manufacture salt and this year it is wheat flour.

Our current turnover is about Rs. 7500 cr and by the next decade we are looking at a target of 15000 cr.

---

### Contact Us

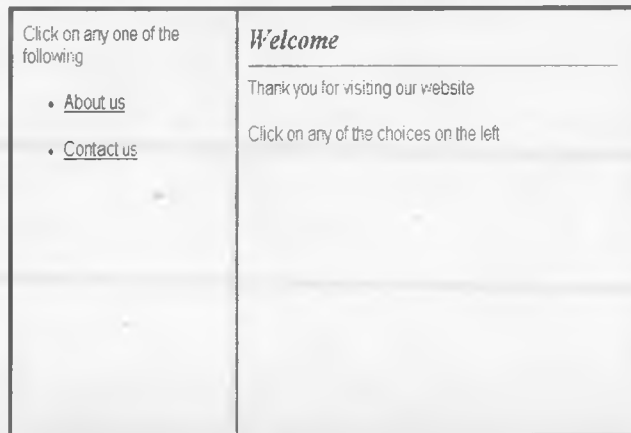
---

You can contact us at the following address:

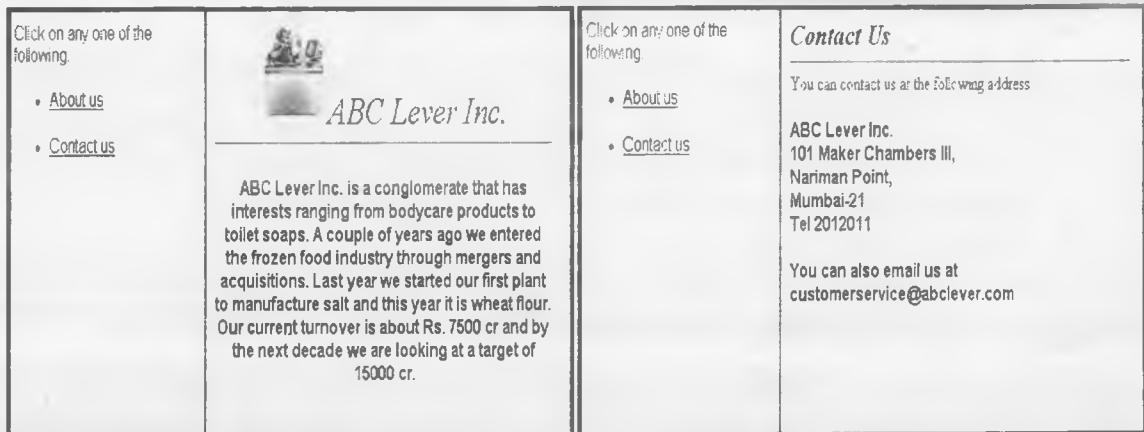
ABC Lever Inc.  
101 Maker Chambers III,  
Nariman Point,  
Mumbai - 21  
Tel. 2102011

You can also email us at [customersservice@abclever.com](mailto:customersservice@abclever.com)

### Output:



**Diagram 7.4.1:** Screen 1 of the Hands on Exercise.



**Diagram 7.4.2:** Screen 2 of the Hands on Exercise.      **Diagram 7.4.3:** Screen 3 of the Hands on Exercise.



# A - PROJECTS IN HTML

## Project Specifications For The First Project In HTML

Since the learning of HTML is completed, it is time to consolidate this learning by building a small web site. The structure of web site is given in the following pages. This is a description of how the pages of this web site will be navigated through starting from the traditional first page, index.html.

Each web page called from index.html is described in complete detail.

- Textual content
- Simple visuals in the form of .gif or .jpeg files
- The look and feel of each page

The files required to construct these pages is available on the accompanying CD-ROM for immediate use.

There are two unique sets of files:

- One set of files, is the raw text files that can be used for formatting using appropriate HTML tags. In addition to the text files, there are .gif and .jpg (*where necessary*) that will help build the web site.
- The other set of files, are HTML files, which are a solution.

If required run the HTML files first in a Web Browser and get a look and feel of what the project could be like. Once this is done code the web pages according to what you believe is appropriate.

Each HTML file is really just a simple guideline to what the web page could look like.

Feel free to improvise and get a look and feel that satisfies you.

### For your guidance:

A broad idea of the HTML pages of the project is given in the diagrams below.

The images and links of each page have been explicitly indicated. Below each diagram is a table that indicates the names of both the **Text** files and the **Images** used.

- The image files that are used in the project are in the sub directory named **GIFJPG**.

- The Text files, that have to be formatted, are in the sub directory named **TEXTFILES**.

### Output For The Project's Home Page

The Home page objects and their corresponding file names:

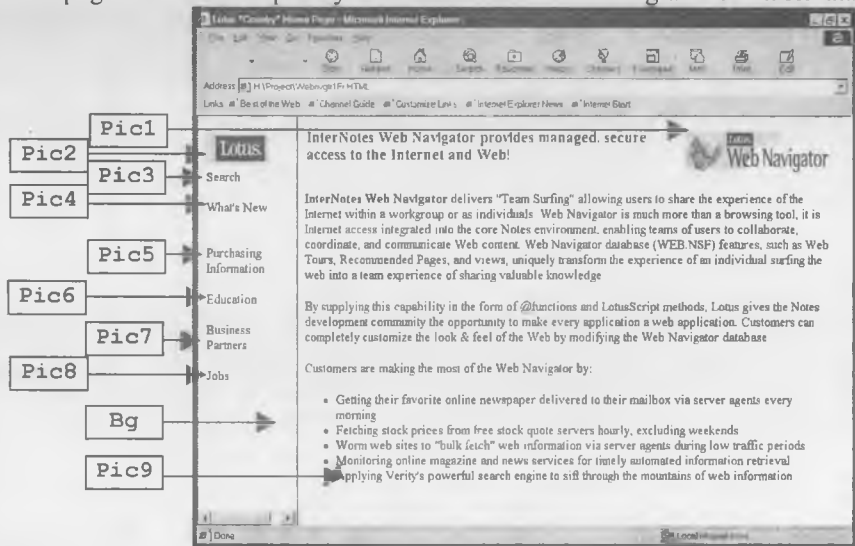


Diagram 1.1a: Pictures of the Home page Webnvgtr1.html.

Bg	yellban1.gif	Pic1	le2 0.gif	Pic2	ylotus1.gif	Pic3	ysearch.gif	Pic8	yjobs.gif
Pic4	ywhatnew.gif	Pic5	ybuy.gif	Pic6	yeducat.gif	Pic7	ypartner.gif	Pic9	white2.gif

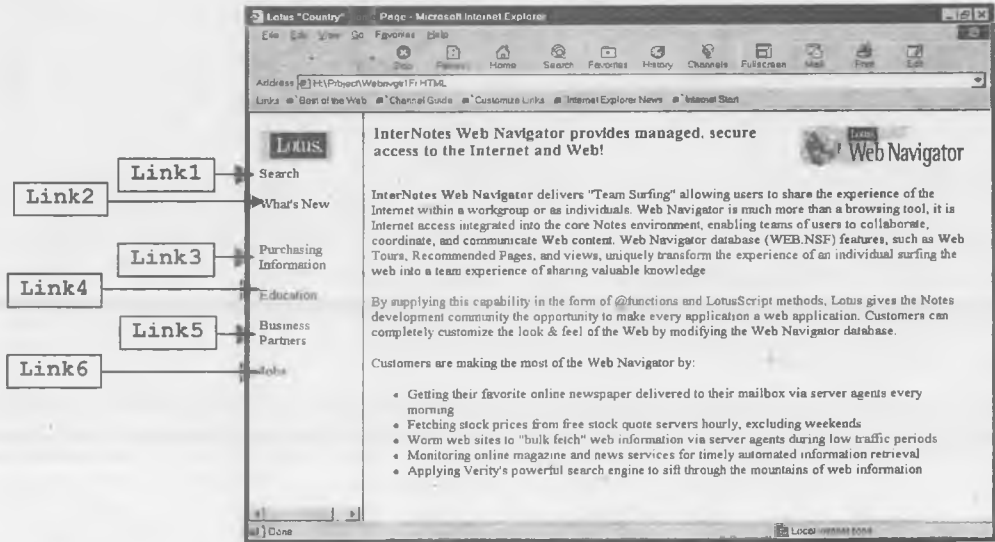


Diagram 1.1b: Links of the Home page Webvngtr1.html.

The Home page links and their corresponding file names:

Link1	search1.html	Link2	whatsnew1.html	Link3	buylots1.html
Link4	laec1.html	Link5	partners1.html	Link6	strategy1.html

Output For search1.html called from the Home page[Search]

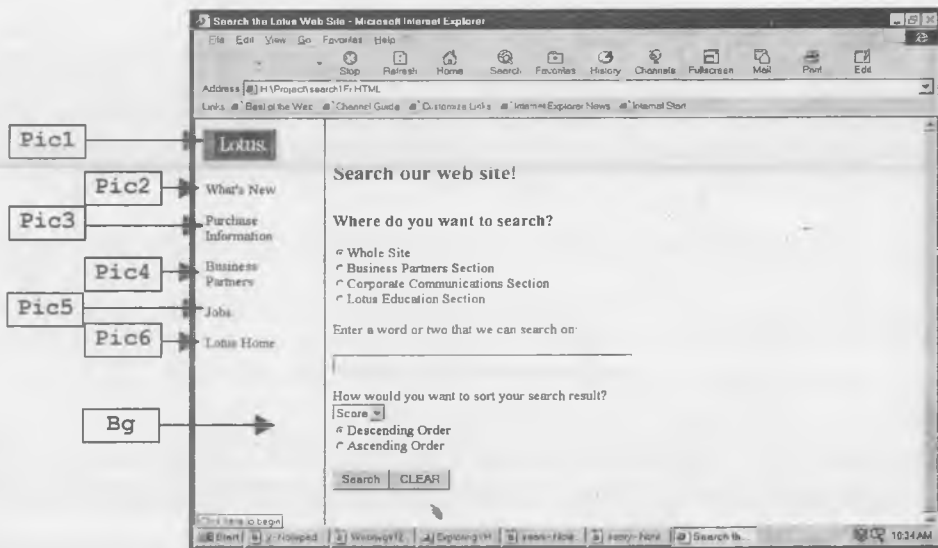


Diagram 1.1.1a: Pictures of the page search1.html.

The search1.html page objects and their corresponding file names:

Bg	purpband.gif	Pic1	plotus1.gif	Pic2	pwhatnew.gif	Pic3	pbuy.gif
Pic4	ppartner.gif	Pic5	pjobs.gif	Pic6	phome.gif		

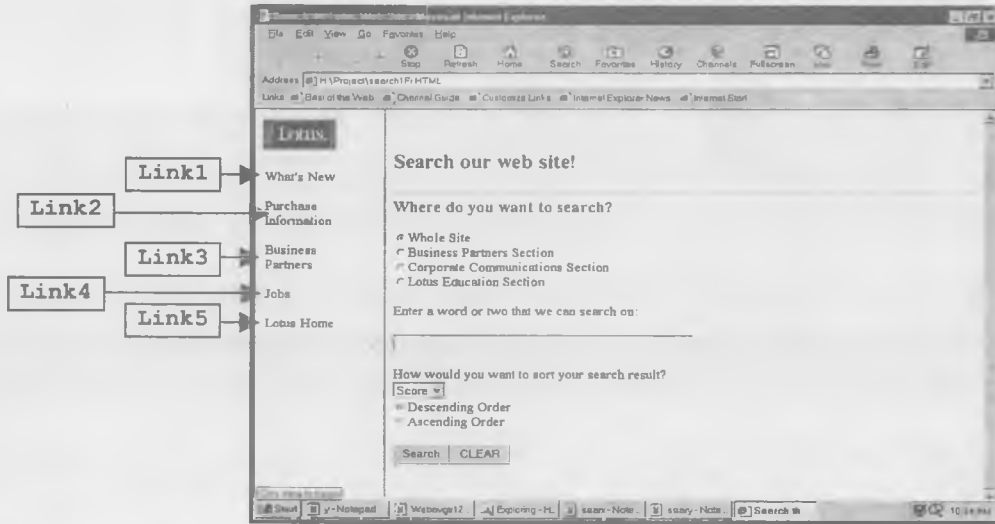


Diagram 1.1.1b: Links of the page search1.html.

The search1.html page links and their corresponding file names:

Link1	whatsnew1.html	Link2	buylots1.html	Link3	partners1.html
Link4	strategy1.html	Link5	webnvgtr1.html		

Output for whatsnew1.html called from the Home page[What's New]

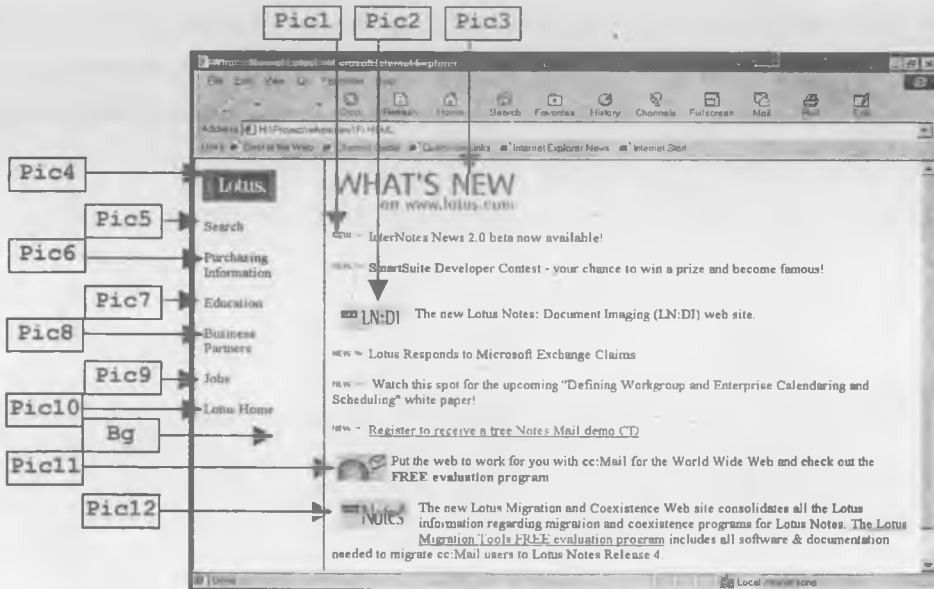


Diagram 1.1.2a: Pictures of the page whatsnew1.html.

The whatsnew1.html page objects and their corresponding file names:

Bg	purpband.gif	Pic1	genrcnew1.gif	Pic2	Lndinew.gif	Pic3	newtxt.gif	Pic4	plotus1.gif
Pic5	psearch.gif	Pic6	pbuy.gif	Pic7	yeducat.gif	Pic8	ppartner.gif	Pic9	pjobs.gif
Pic10	phome.gif	Pic11	ccwbsmal.gif	Pic12	notesnew.gif				

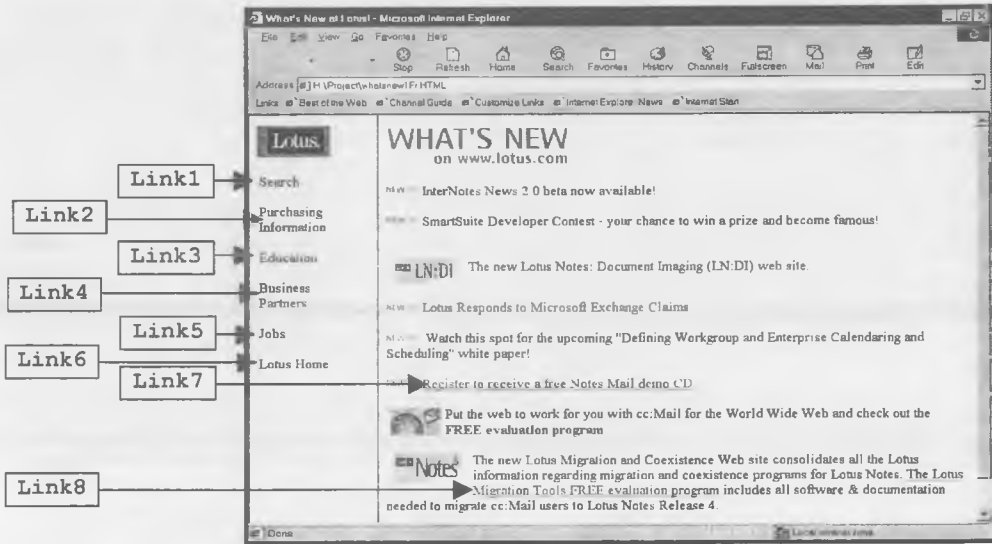


Diagram 1.1.2b: Links of the page whatsnew1.html.

The whatsnew1.html page links and their corresponding file names:

Link1	search1.html	Link2	buylots1.html	Link3	laec1.html	Link4	partners1.html
Link5	strategy1.html	Link6	webnvgr1.html	Link7	notesmail.html	Link8	migreg1.html

Output for notesmail.html called from Link7 on the whatsnew1.html page

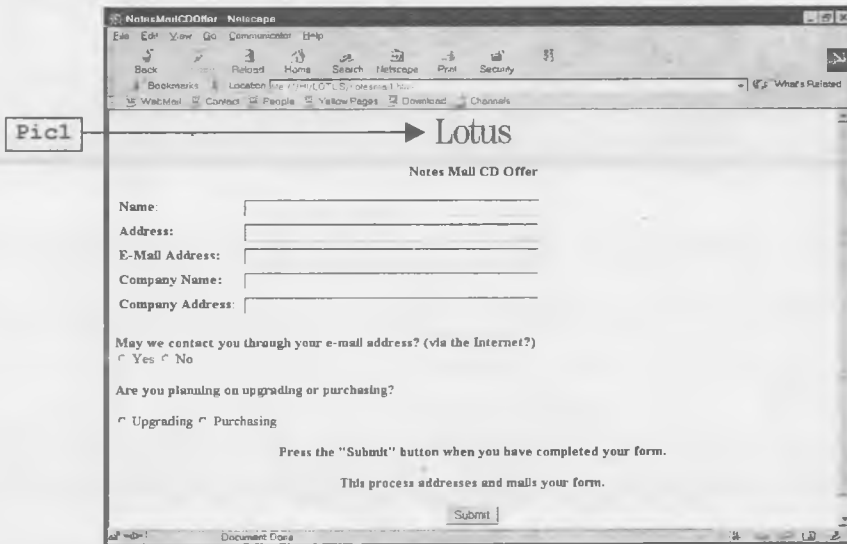


Diagram 1.1.2.1: Picture of the page notesmail.html.

The notesmail.html page object and it's corresponding file name:

Pic1	4ae 0.gif
------	-----------

Output for migreg1.html called from Link8 on the whatsnew1.html page

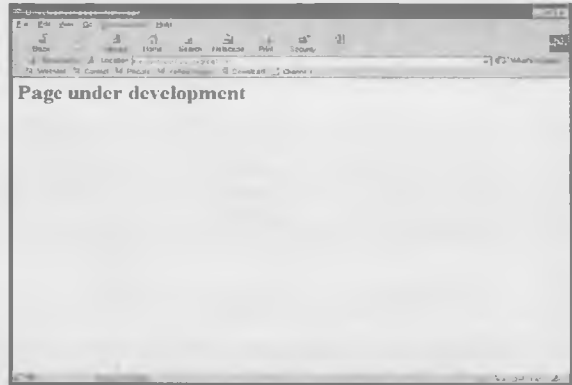


Diagram 1.1.2.2: The page migreg1.html.

Output for buyouts1.html called from the Home page [Purchase Information]

The buyouts1.html page objects and their corresponding file names:

Pic1	plotus1.gif
Pic2	buytxt.gif
Pic3	psearch.gif
Pic4	pwhatnew.gif
Pic5	yeducat.gif
Pic6	ppartner.gif
Pic7	pjobs.gif
Pic8	phome.gif
Pic9	nobull.gif
Bg	purpband.gif

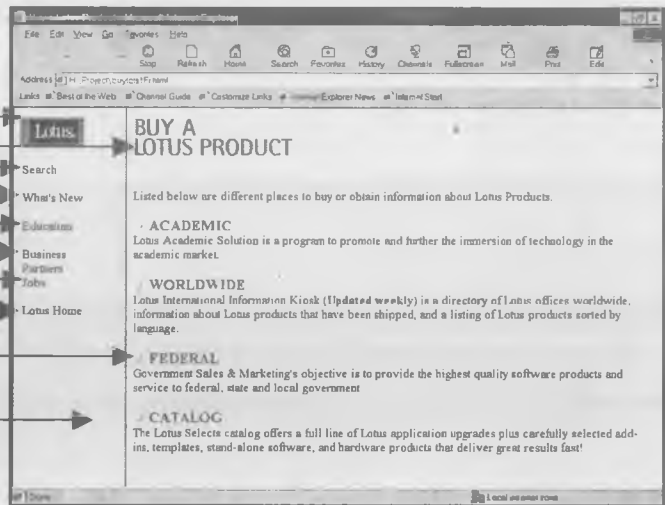
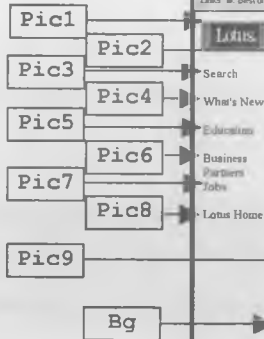


Diagram 1.1.3a: Pictures of the page buyouts1.html.

The buyouts1.html page links and their corresponding file names:

Link1	search1.html
Link2	whatsnew1.html
Link3	laec1.html
Link4	partners1.html
Link5	strategy1.html
Link6	webnvgr1.html

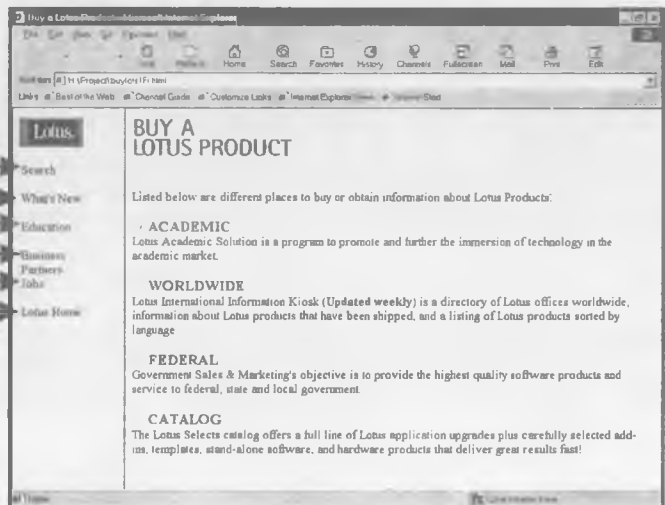
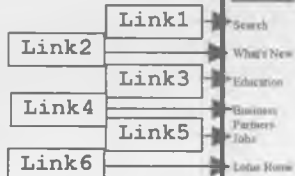


Diagram 1.1.3b: Links of the page buyouts1.html.

Output for laec1.html called from the Home page [Education]

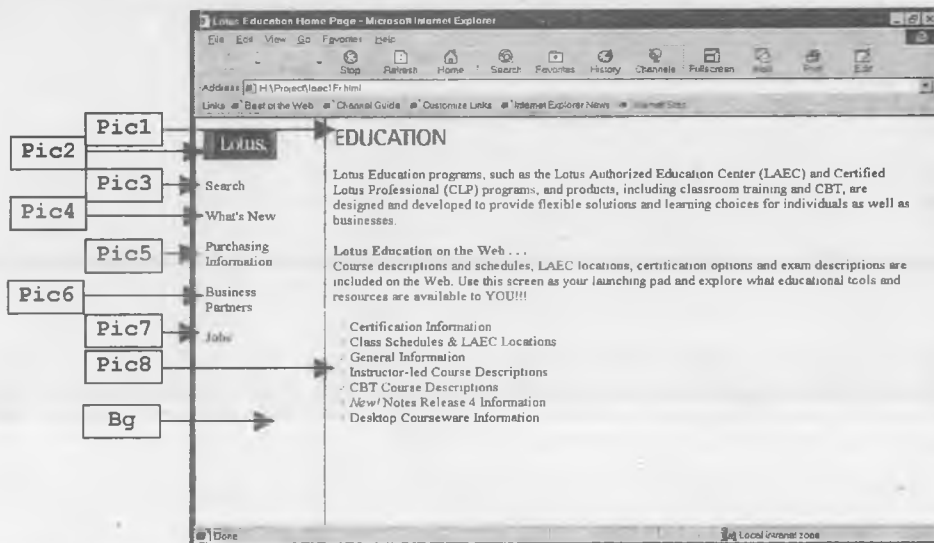


Diagram 1.1.4a: Pictures of the page laec1.html.

The laec1.html page objects and their corresponding file names:

Bg	purpband.gif	Pic1	edutxt.gif	Pic2	plotus1.gif	Pic3	psearch.gif	Pic5	pbuy.gif
Pic4	pwhatnew.gif	Pic6	ppartner.gif	Pic7	pjobs.gif	Pic8	nobull.gif		

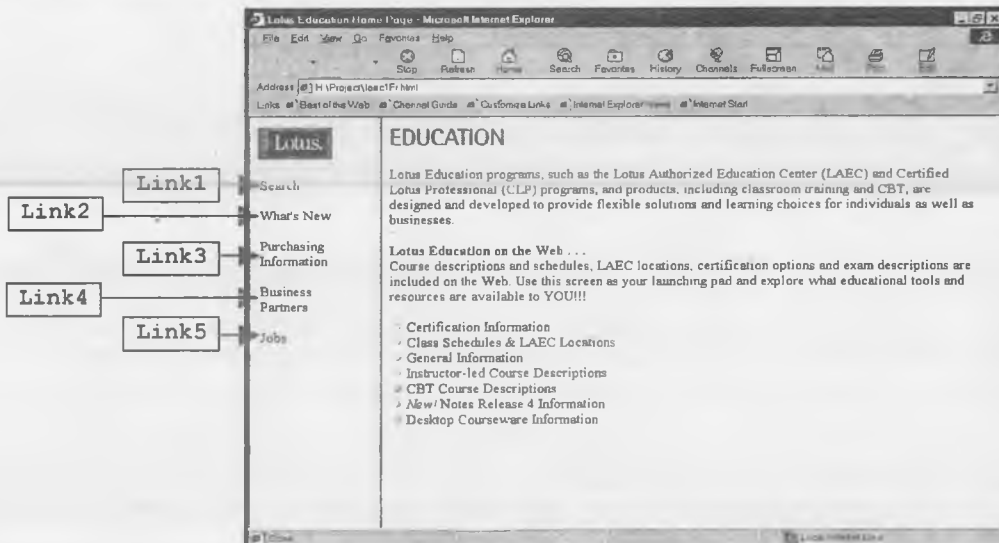
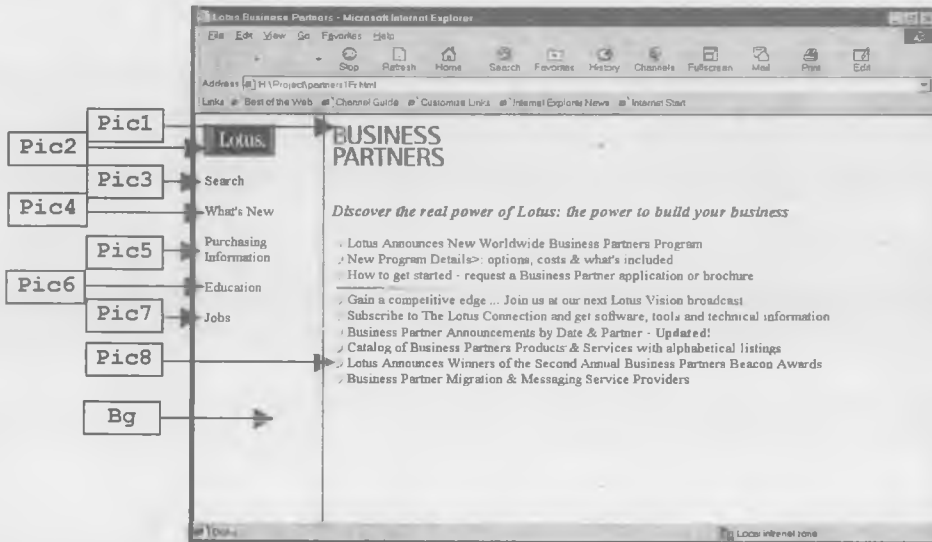


Diagram 1.1.4b: Links of the page laec1.html.

The laec1.html page links and their corresponding file names:

Link1	search1.html	Link2	whatsnew1.html	Link3	buylots1.html
Link4	partners1.html	Link5	strategy1.html		

**Output for partners1.html called from the Home page [Business Partners]**



**Diagram 1.1.5a:** Pictures of the page partners1.html.

The partners1.html page objects and their corresponding file names:

Bg	purpband.gif	Pic1	bptxt.gif	Pic2	plotus1.gif	Pic3	psearch.gif	Pic5	pbuy.gif
Pic4	pwhatnew.gif	Pic6	yeducat.gif	Pic7	pjobs.gif	Pic8	nobull.gif		

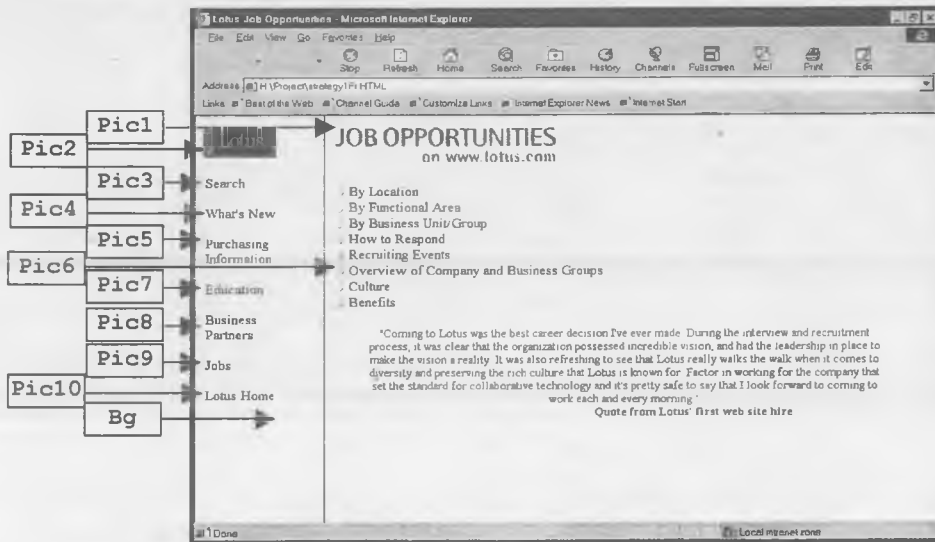


**Diagram 1.1.5b:** Links of the page partners1.html.

The partners1.html page links and their corresponding file names:

Link1	search1.html	Link2	whatsnew1.html	Link3	buylots1.html
Link4	laec1.html	Link5	strategy1.html		

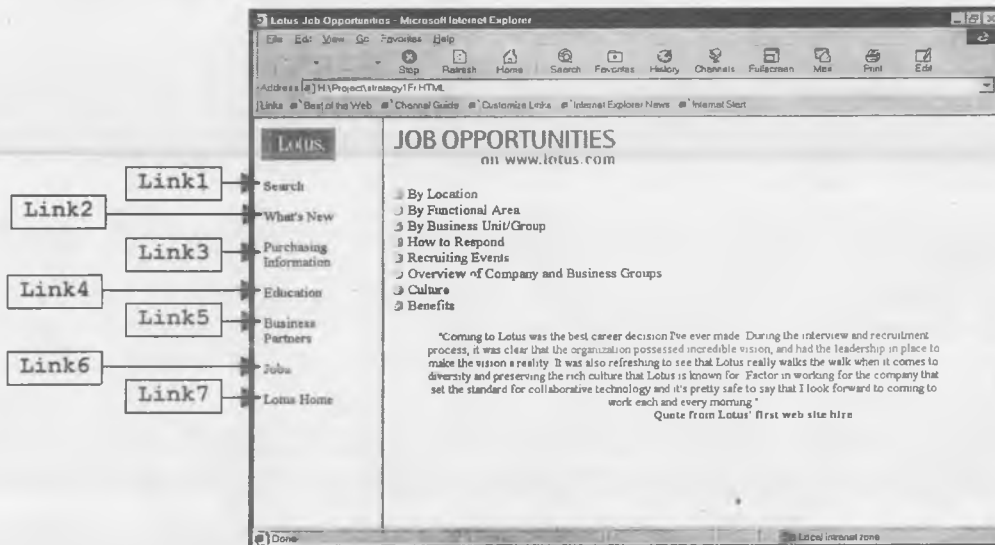
**Output for strategy1.html called from the Home page [Jobs]**



**Diagram 1.1.6a:** Pictures of the page strategy1.html.

The strategy1.html page objects and their corresponding file names:

Bg	purpband.gif	Pic1	jobtxt.gif	Pic2	plotus1.gif	Pic3	psearch.gif
Pic4	pwhatnew.gif	Pic5	pbuy.gif	Pic6	nobull.gif	Pic7	yeducat.gif
Pic8	ppartner.gif	Pic9	pjobs.gif	Pic10	phome.gif		



**Diagram 1.1.6b:** Links of the page strategy1.html.

The strategy1.html page links and their corresponding file names:

Link1	search1.html	Link2	whatsnew1.html	Link3	buylots1.html	Link4	laec1.html
Link5	partners1.html	Link6	strategy1.html	Link7	webnvgr1.html		



## Project Specifications For The Second Project In HTML

Since the learning of HTML is completed, it is time to consolidate this learning by building a small web site. The structure of web site is given in the following pages. This is a description of how the pages of this web site will be navigated through starting from the traditional first page, index.html.

Each web page called from index.html is described in complete detail.

- Textual content
- Simple visuals in the form of .gif or .jpeg files
- The look and feel of each page

The files required to construct these pages is available on the accompanying CD-ROM for immediate use.

There are two unique sets of files:

- One set of files, is the raw text files that can be used for formatting using appropriate HTML tags. In addition to the text files, there are .gif and .jpg (*where necessary*) that will help build the web site.
- The other set of files, are HTML files, which are a solution.

If required run the HTML files first in a Web Browser and get a look and feel of what the project could be like. Once this is done code the web pages according to what you believe is appropriate.

Each HTML file is really just a simple guideline to what the web page could look like.

Feel free to improvise and get a look and feel that satisfies you.

### For your guidance:

A broad idea of the HTML pages of the project is given in the diagrams below.

The images and links of each page have been explicitly indicated. Below each diagram is a table that indicates the names of both the **Text** files and the **Images** used.

- The image files that are used in the project are in the sub directory named **IMAGES**.
- The Text files, that have to be formatted, are in the sub directory named **TEXTFILES**.

### Output For The Project's Home Page

The **Home** page objects and their corresponding file names:

Pic1	Book.gif
------	----------

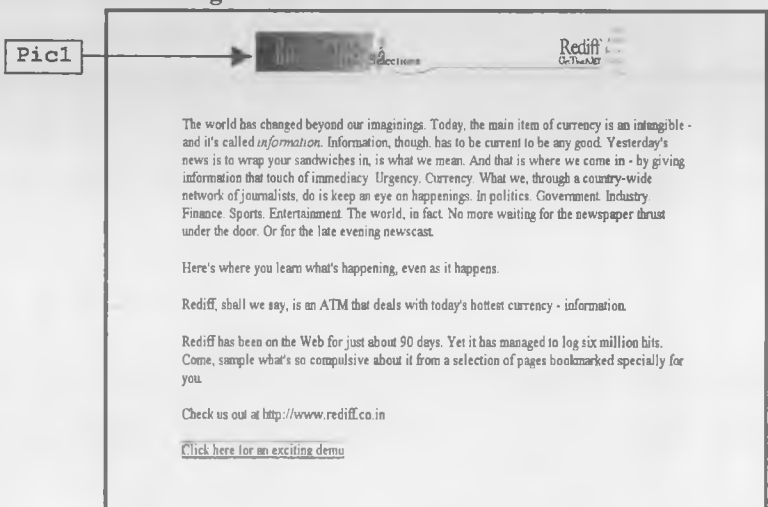


Diagram 2.1.1a: Pictures of the Home page Intro.htm.

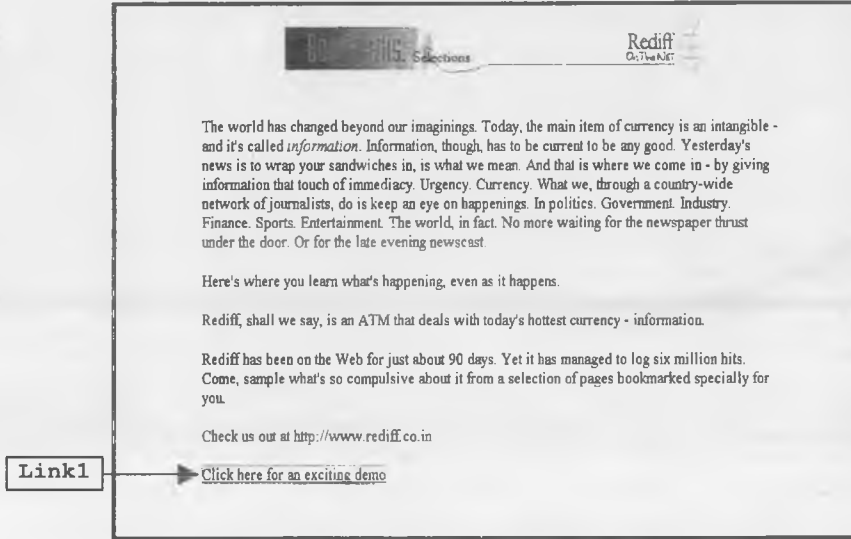


Diagram 2.1.1b: Links of the Home page Intro.htm.

The Home page link and it's corresponding file name:

Link1	2ndhome.htm
-------	-------------

Output For 2ndhome.htm called from the Home page[Click here for an exciting demo]

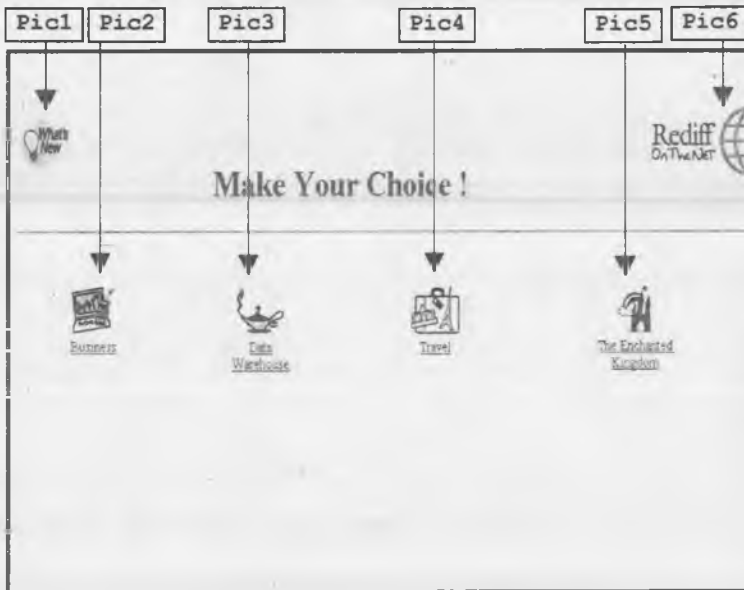


Diagram 2.1.2a: Pictures of the page 2ndhome.htm.

The 2ndhome.htm page objects and their corresponding file names:

Pic1	whatsne2.gif	Pic2	business2.gif	Pic3	archive2.gif
Pic4	travel2.gif	Pic5	kids2.gif	Pic6	logo1.gif

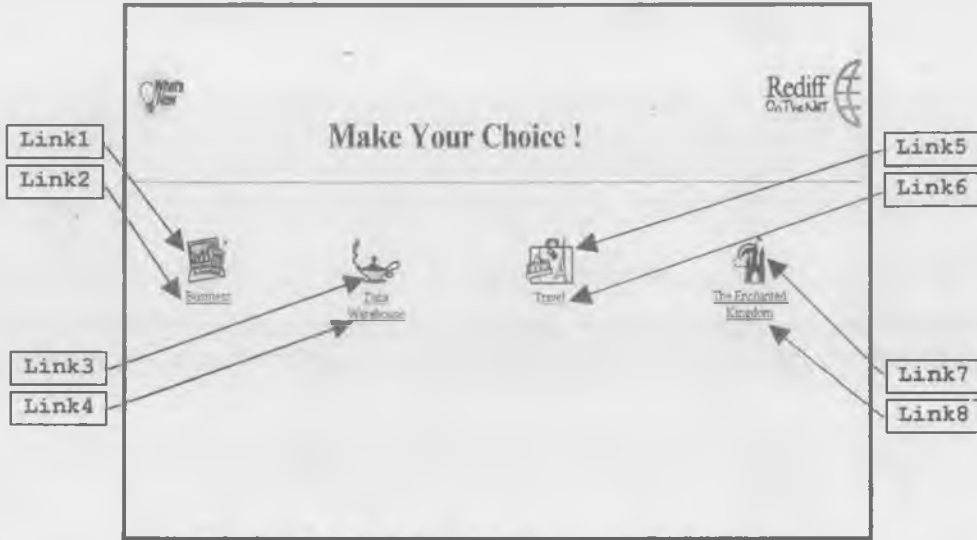


Diagram 2.1.2b: Links of the Home page 2ndhome.htm.

The 2ndhome.htm page links and their corresponding file names:

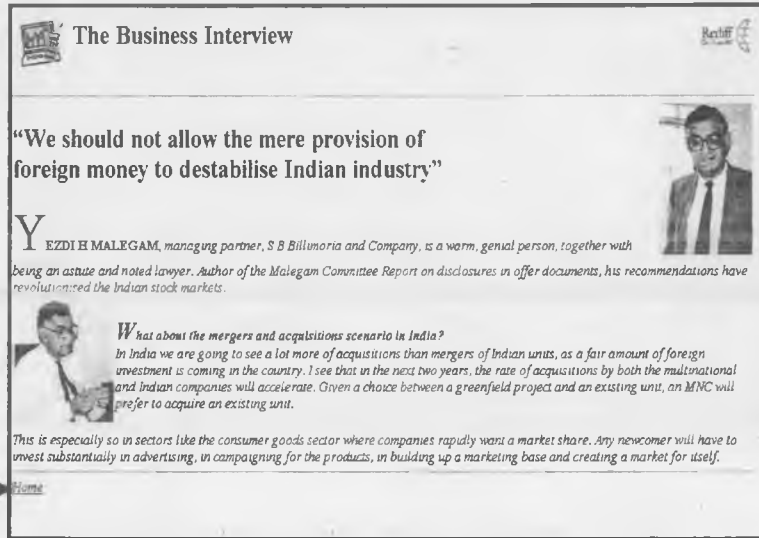
Link1	malegam.htm	Link3	fintabl2.htm	Link5	ladakh.htm	Link7	aquarium.htm
Link2	malegam.htm	Link4	fintabl2.htm	Link6	ladakh.htm	Link8	aquarium.htm

**Output For malegam.htm called from the 2ndhome.htm page [Business]**

Diagram 2.2.1a: Pictures of the page malegam.htm.

The malegam.htm page objects and their corresponding file names:

Pic1	business2.gif	Pic2	malegama.jpg	Pic3	malegam.jpg	Pic4	logo1.gif
------	---------------	------	--------------	------	-------------	------	-----------



Link1

Diagram 2.2.1b: Links of the page malegam.htm.

The malegam.htm page link and its corresponding file name:

Link1 | 2ndhome.htm

Output For fintabl2.html called from the 2ndhome.html page [Data Warehouse]

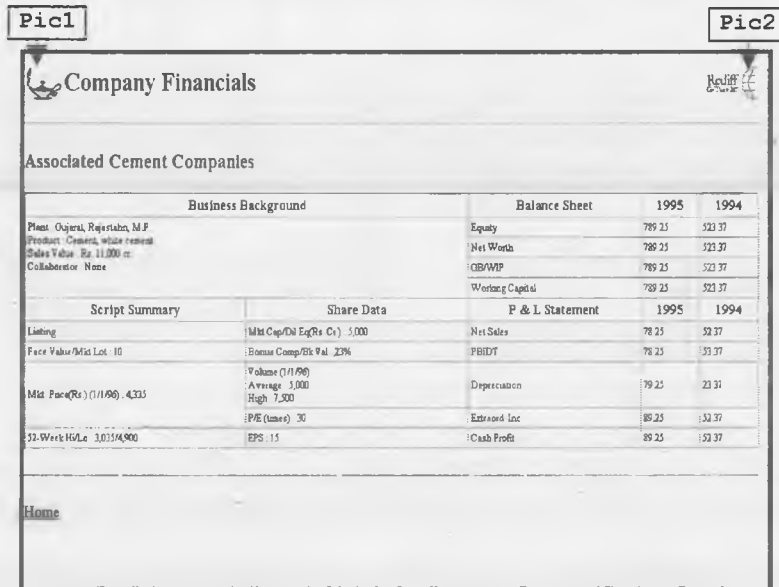




Diagram 2.2.2a: Pictures of the page fintabl2.htm.

The fintabl2.htm page objects and their corresponding file names:

Pic1 | archive2.gif | Pic2 | logo1.gif


Company Financials


---

**Associated Cement Companies**

Business Background		Balance Sheet	1995	1994
Plant: Qasrat, Rameshah, M.P.		Equity	789.25	523.37
Product: Cement, White Cement		Net Worth	789.25	523.37
Sales Value: Rs 1,000 cr		OB/WTD	789.25	523.37
Collaborator: None		Working Capital	789.25	523.37

Script Summary	Share Data	P & L Statement	1995	1994
Listing	Mix Cap/Dil Eq(Rs Cr) 5,000	Net Sales	78.25	53.37
Face Value/Mix Lot: 10	Bonus Comp/Bk 9 of 2%	PBITD	78.25	53.37
	Volume (1/1/96)	Depreciation	79.25	23.37
	Average 5,000			
	High 7,500	Extensd. Inc	89.25	52.37
	P/E (Times) 30	Cash Profit	89.25	52.37
	EPS 15			

Link1 → [Home](#)

Diagram 2.2.2b: Links of the page fintabl2.htm.

The **fintabl2.htm** page link and it's corresponding file name:



Link1 2ndhome.htm

**Output For ladakh.html called from the 2ndhome.html page [Data Warehouse]**

Pic1

Pic2

Pic3


Ladakh


---

**Postcards from the Edge**

*Joolay, Joolay*

The aircraft swooped into a valley perched on top of the world

Beautifully stark, the corrugated landscape stretches out as far as the eye can see. Snow capped mountains fold away from the verdant valleys like so many wrinkles of a rough burlap sack. An invigorating breeze whistles about your ears -- the only sound that disturbs the ringing stillness.

Where do you find a wee place located way up in the clouds where life has a celestial flow to it?

Disembark Leh, Ladakh. Ladakh is Little Tibet. But a miniature, remote and free Tibet. India's largest district, situated on a barren dry plateau, is part of the troubled state of Kashmir. Boundaries on a map is where the connection ends. I.

Ladakh: A Paradise not lost. There are no cockroaches in this Happy Valley. No beggars. Or graffiti. Cholesterol is not found either. Can anything be better? In summer, life has the brisk pace of the gently gurgling green Indus. And by winter life freezes up just like the good river. The lowering snow-speckled mountains do not cast mean shadows but only thick, velvety ones that are so reassuring. And the wispy clouds roll along in the brightest blue sky this side of the Khyber Pass.

**Farewell**

Diagram 2.2.3a: Pictures of the page ladakh.htm.

The **ladakh.htm** page objects and their corresponding file names:

Pic1 travel2.gif Pic2 01ladakh.jpg Pic3 logo1.gif

Output For aquarium.htm called from the 2ndhome.html page [The Enchanting Kingdom]

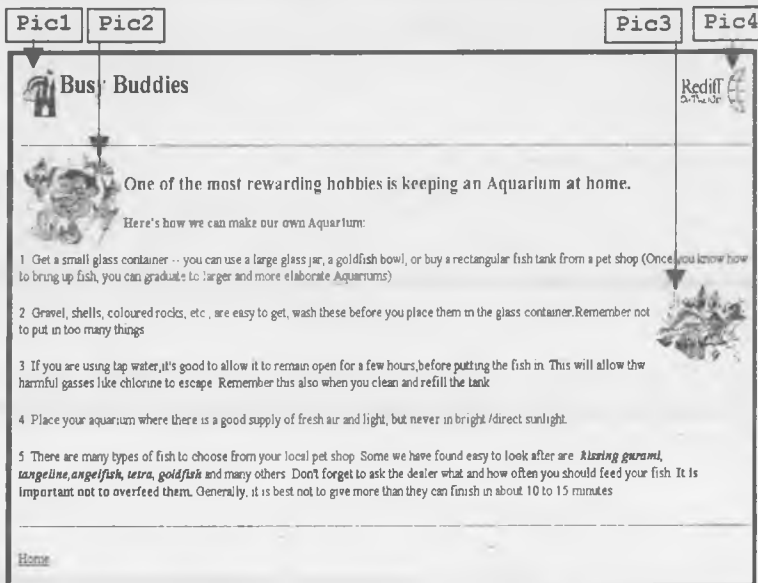


Diagram 2.2.4a: Pictures of the page aquarium.htm.

The aquarium.htm page objects and their corresponding file names:

Pic1	kids2.gif	Pic2	aqua1.jpg	Pic3	aqua2.jpg	Pic4	logol.gif
------	-----------	------	-----------	------	-----------	------	-----------

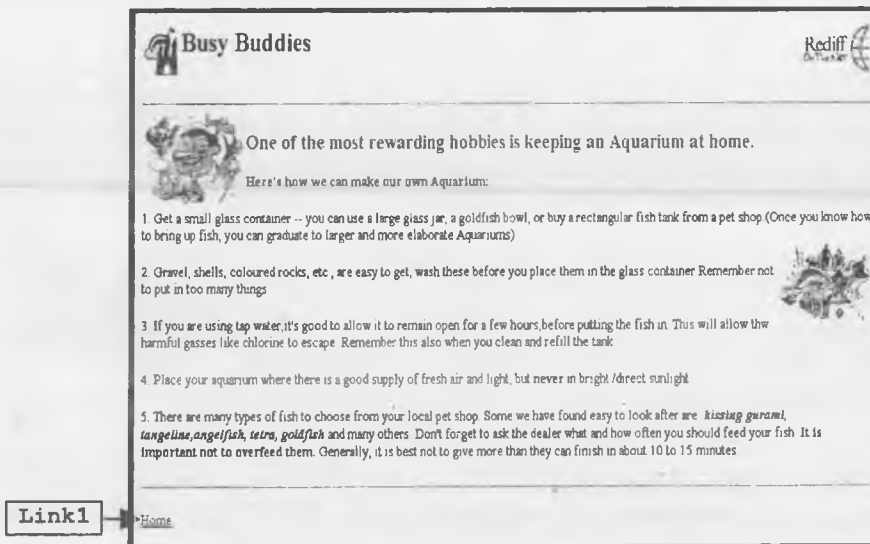


Diagram 2.2.4b: Links of the page aquarium.htm.

The aquarium.htm page link and it's corresponding file name:

Link1	2ndhome.htm
-------	-------------

## Project Specifications For The Third Project In HTML

Since the learning of HTML is completed, it is time to consolidate this learning by building a small web site. The structure of web site is given in the following pages. This is a description of how the pages of this web site will be navigated through starting from the traditional first page, index.html.

Each web page called from index.html is described in complete detail.

- Textual content
- Simple visuals in the form of .gif or .jpeg files
- The look and feel of each page

The files required to construct these pages is available on the accompanying CD-ROM for immediate use.

There are two unique sets of files:

- One set of files, is the raw text files that can be used for formatting using appropriate HTML tags. In addition to the text files, there are .gif and .jpg (*where necessary*) that will help build the web site.
- The other set of files, are HTML files, which are a solution.

If required run the HTML files first in a Web Browser and get a look and feel of what the project could be like. Once this is done code the web pages according to what you believe is appropriate.

Each HTML file is really just a simple guideline to what the web page could look like.

Feel free to improvise and get a look and feel that satisfies you.

### For your guidance:

A broad idea of the HTML pages of the project is given in the diagrams below.

The images and links of each page have been explicitly indicated. Below each diagram is a table that indicates the names of both the **Text** files and the **Images** used.

- The image files that are used in the project are in the sub directory named **IMAGES**.
- The Text files, that have to be formatted, are in the sub directory named **TEXTFILES**.

### Output For The Project's Home Page

The **Home** page objects and their corresponding file names:

Frame1	welleft.htm
Frame2	welright.htm
Bg1	pa05.jpg
Bg2	pa05.jpg
Pic1	rdglogo.jpg
Pic2	new.gif

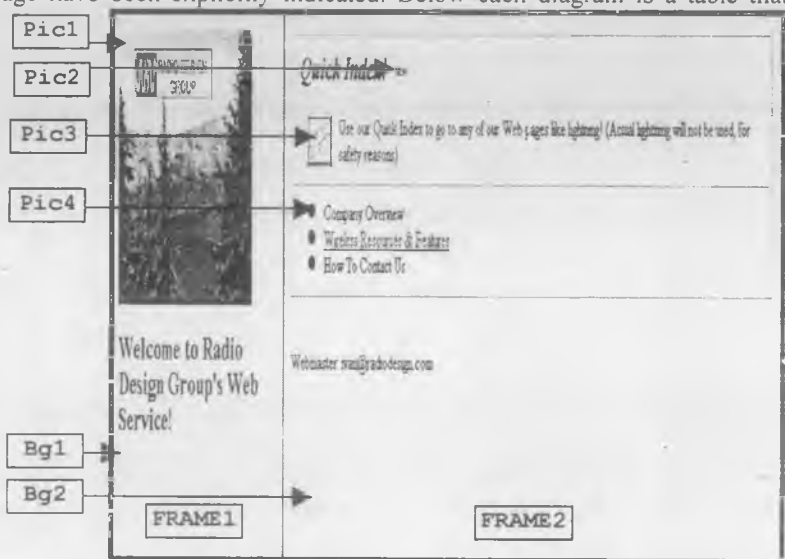


Diagram 3.1a: Pictures of the Home page welfrm.htm.

Pic3	ltngbutn.gif	Pic4	blueb.gif
------	--------------	------	-----------

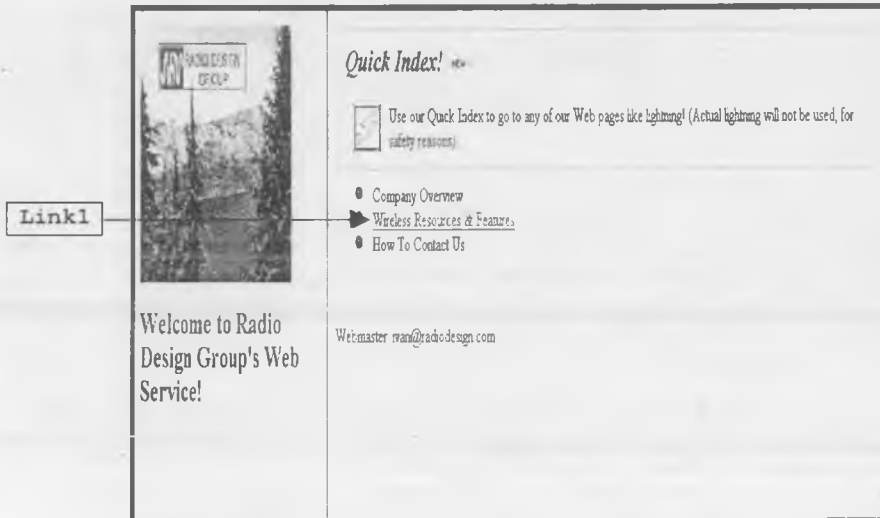


Diagram 3.1b: Link of the Home page welright.htm in the right side frame for welfrm.htm.

The Home page link and it's corresponding file name:

Link1	hwitwrks1.htm
-------	---------------

**Output For hwitwrks1.htm called from the Home page |Wireless Resource & Features|**

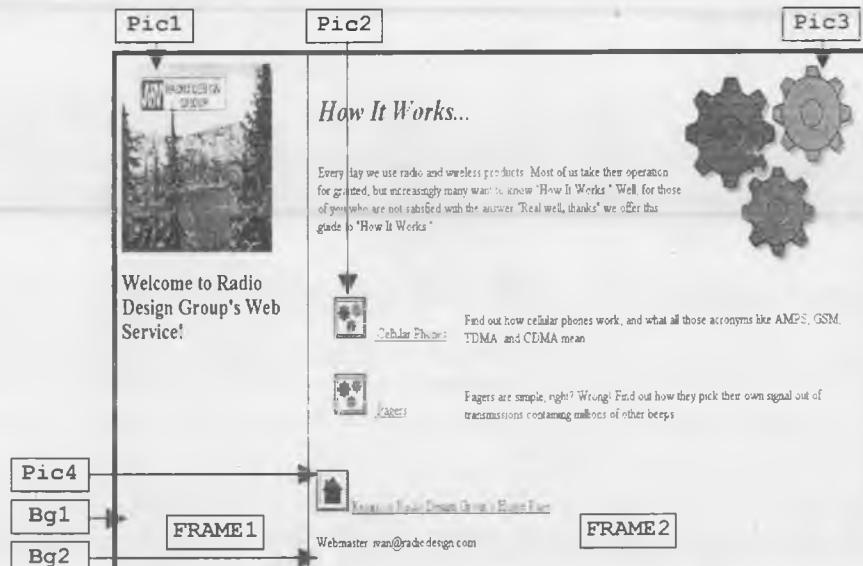


Diagram 3.2a: Pictures of the page hwitwrks1.htm in the right side frame for welfrm.htm.

The hwitwrks1.htm page objects and their corresponding file names:

Frame1	welleft.htm	Bg1	pa05.jpg	Pic1	rdglogo.jpg	Pic3	gears.gif
Frame2	welright.htm	Bg2	pa05.jpg	Pic2	gearbutn.gif	Pic4	house.gif



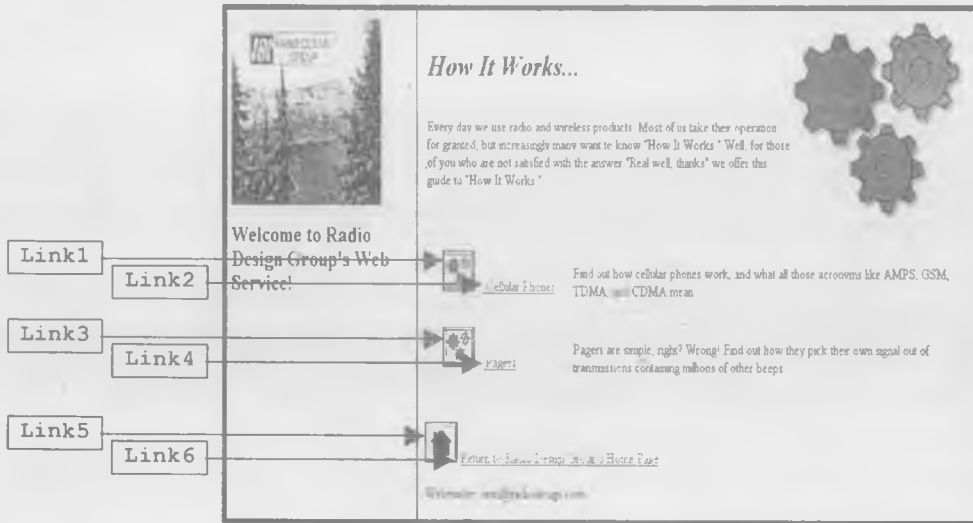


Diagram 3.2b: Links of the page hwitwrks1.htm in the right side frame for welfrm.htm.

The hwitwrks1.htm page links and their corresponding file names:

Link1	cellwrks1.htm	Link3	pgrwrks1.htm	Link5	welright.htm
Link2	cellwrks1.htm	Link4	pgrwrks1.htm	Link6	welright.htm

**Output For cellwrks1.htm called from the hwitwrks1.htm page | Cellular Phones**

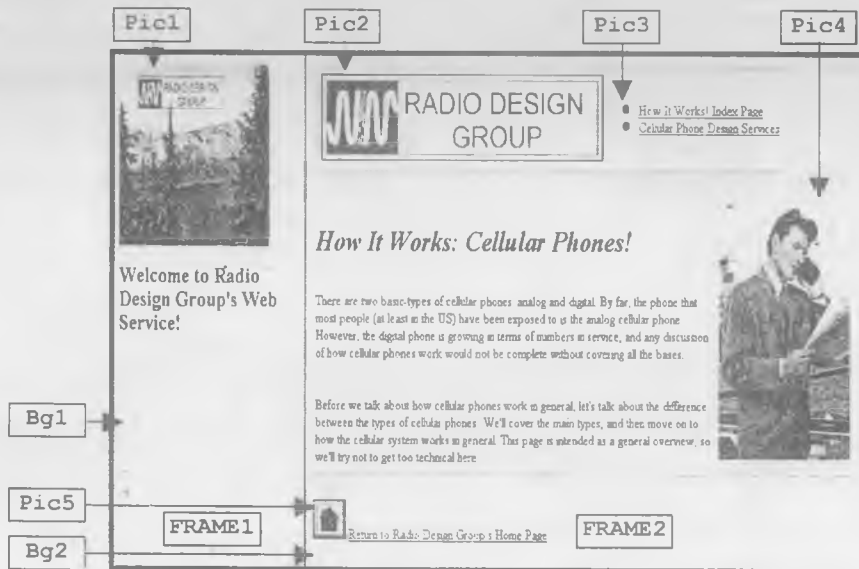


Diagram 3.3a: Pictures of the page cellwrks1.htm in the right side frame for welfrm.htm.

The cellwrks1.htm page objects and their corresponding file names:

Frame1	welleft.htm	Bg1	pa05.jpg	Pic1	rdglogo.jpg	Pic3	blueb.gif	Pic5	house.gif
Frame2	welright.htm	Bg2	pa05.jpg	Pic2	redlogo.gif	Pic4	mancell.jpg		

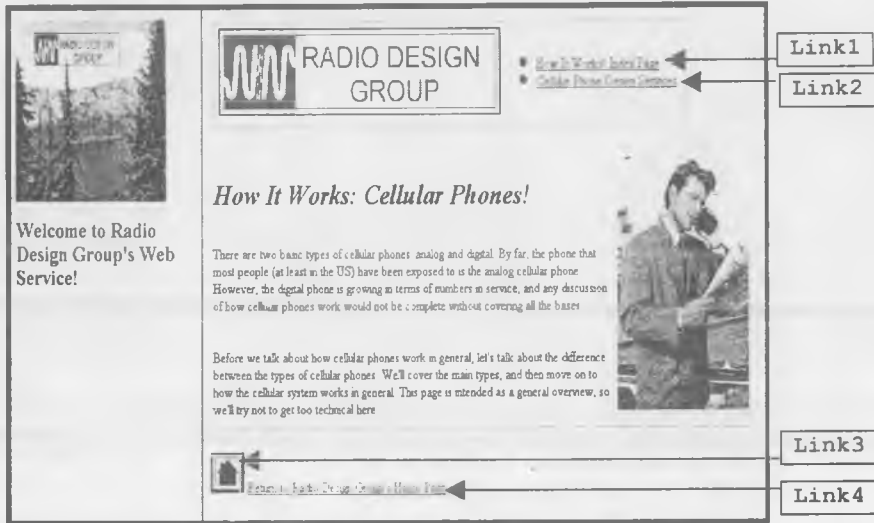


Diagram 3.3b: Links of the page cellwrks1.htm in the right side frame for welfrm.htm.

The cellwrks1.htm page links and their corresponding file names:

Link1	hwitwrks1.htm	Link2	pager1.htm	Link3	welright.htm	Link4	welright.htm
-------	---------------	-------	------------	-------	--------------	-------	--------------

Output For pgrwrks1.htm called from the hwitwrks1.htm page [Cellular Phones]



Diagram 3.4a: Pictures of the page pgrwrks1.htm in the right side frame for welfrm.htm.

The pgrwrks1.htm page objects and their corresponding file names:

Frame1	welleft.htm	Bg1	pa05.jpg	Pic1	rdglogo.jpg	Pic3	blueb.gif	Pic5	house.gif
Frame2	welright.htm	Bg2	pa05.jpg	Pic2	redlogo.gif	Pic4	pager.gif		

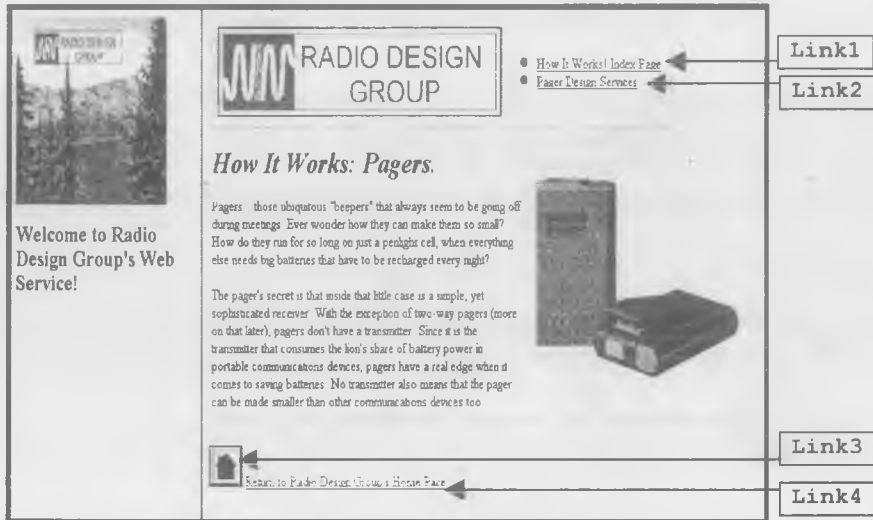


Diagram 3.4b: Links of the page `pgrwrks1.htm` in the right side frame for `welfrm.htm`.

The `pgrwrks1.htm` page links and their corresponding file names:

Link1	<code>hwitwrks1.htm</code>	Link2	<code>pager1.htm</code>	Link3	<code>welright.htm</code>	Link4	<code>welright.htm</code>
-------	----------------------------	-------	-------------------------	-------	---------------------------	-------	---------------------------

Output For `pager1.htm` called from the `cellwrks1.htm` page [Cellular Phones Design Services]

OR

Output For `pager1.htm` called from the `pgrwrks1.htm` page [Pager Design Services]

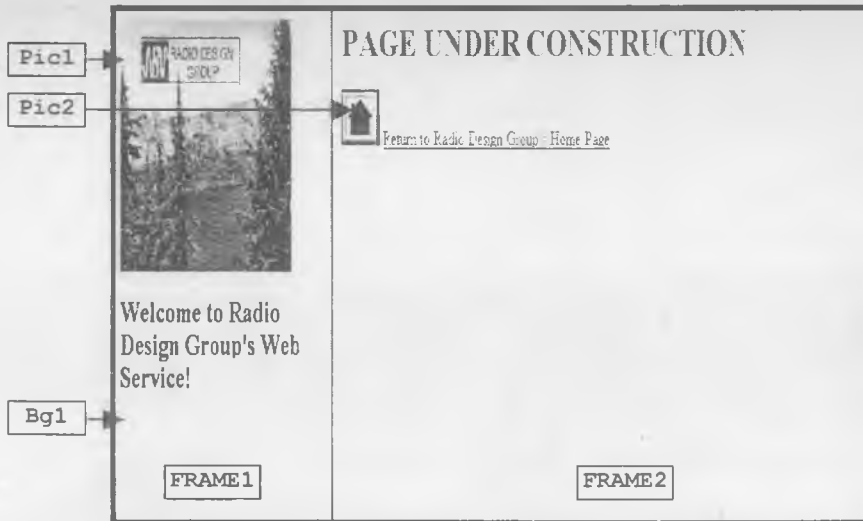


Diagram 3.5a: Pictures of the page `pager1.htm` in the right side frame for `welfrm.htm`.

The `pager1.htm` page objects and their corresponding file names:

Frame1	<code>welleft.htm</code>	Frame2	<code>welright.htm</code>	Bg1	<code>pa05.jpg</code>	Pic1	<code>rdglogo.jpg</code>	Pic2	<code>house.gif</code>
--------	--------------------------	--------	---------------------------	-----	-----------------------	------	--------------------------	------	------------------------

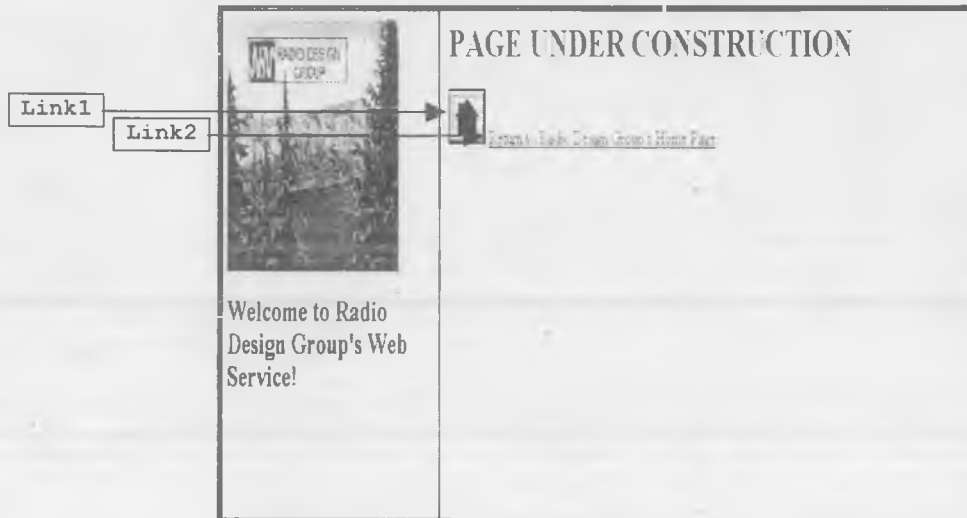


Diagram 3.5b: Links of the page pager1.htm in the right side frame for welfrm.htm.

The **pager1.htm** page links and their corresponding file names:

Link1	welright.htm	Link2	welright.htm
-------	--------------	-------	--------------

### Project Specifications For The Fourth Project In HTML

Since the learning of HTML is completed, it is time to consolidate this learning by building a small web site. The structure of web site is given in the following pages. This is a description of how the pages of this web site will be navigated through starting from the traditional first page, index.html.

Each web page called from index.html is described in complete detail.

- Textual content
- Simple visuals in the form of .gif or .jpeg files
- The look and feel of each page

The files required to construct these pages is available on the accompanying CD-ROM for immediate use.

There are two unique sets of files:

- One set of files, is the raw text files that can be used for formatting using appropriate HTML tags. In addition to the text files, there are .gif and .jpg (*where necessary*) that will help build the web site.
- The other set of files, are HTML files, which are a solution.

If required run the HTML files first in a Web Browser and get a look and feel of what the project could be like. Once this is done code the web pages according to what you believe is appropriate.

Each HTML file is really just a simple guideline to what the web page could look like.

Feel free to improvise and get a look and feel that satisfies you.

#### For your guidance:

A broad idea of the HTML pages of the project is given in the diagrams below.

The images and links of each page have been explicitly indicated. Below each diagram is a table that indicates the names of both the **Text** files and the **Images** used.

- The image files that are used in the project are in the sub directory named **IMAGES**.
- The Text files, that have to be formatted, are in the sub directory named **TEXTFILES**.

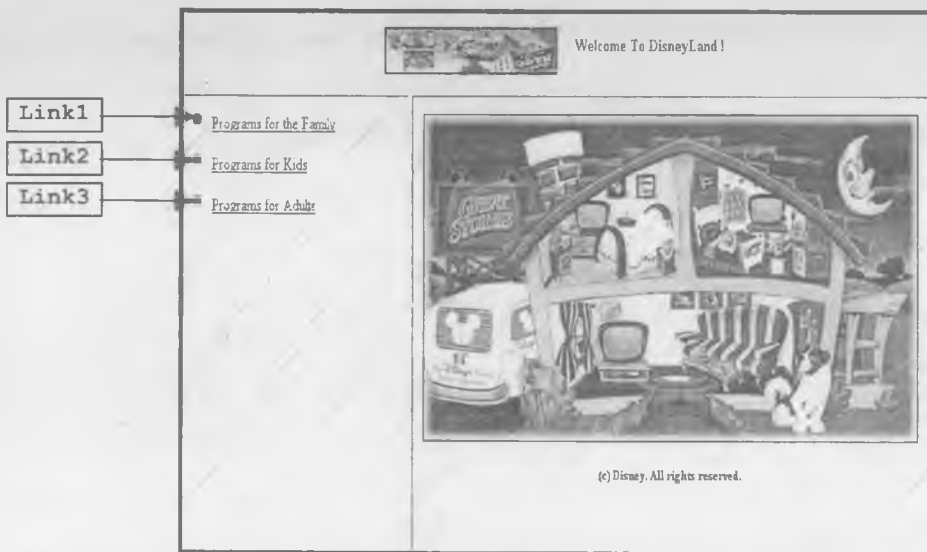
**Output For The Project's Home Page**



**Diagram 4.1a:** Pictures of the Home page Frames.htm.

The Home page objects and their corresponding file names:

Bg1	pinkwhit.gif	Pic1	blueb.gif	Pic2	blueb.gif	Pic3	house sp.gif
Bg2	pinkwhit.gif	Frame1	Head.htm	Frame2	Index.htm	Frame3	Home.htm



**Diagram 4.1b:** Links of the Home page Frames.htm.

The Home page links and their corresponding file names:

Link1	index18.htm	Link2	index19.htm	Link3	Index20.htm
-------	-------------	-------	-------------	-------	-------------

Output For index18.htm called from the Home page [Programs for the Family]

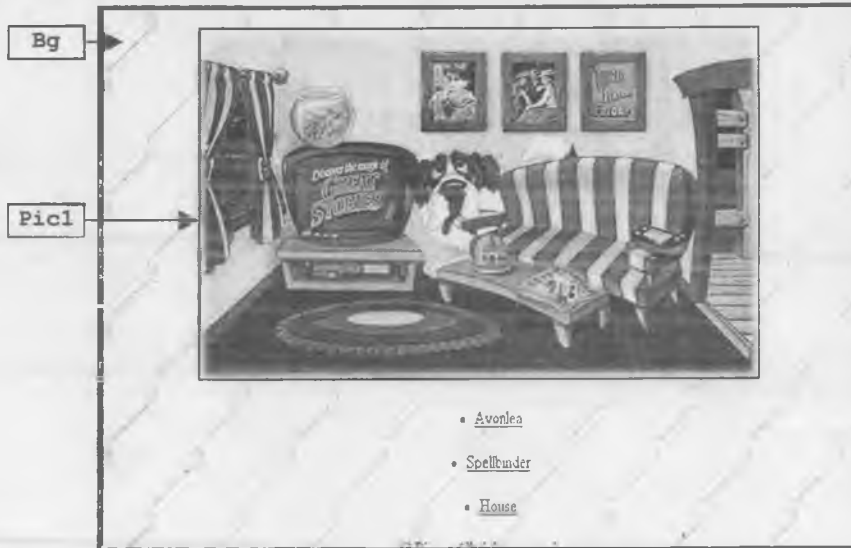


Diagram 4.2a: Pictures of the index18.htm.

The Home page objects and their corresponding file names:

Bg	pinkwhit.gif	Pic1	family r.jpg
----	--------------	------	--------------

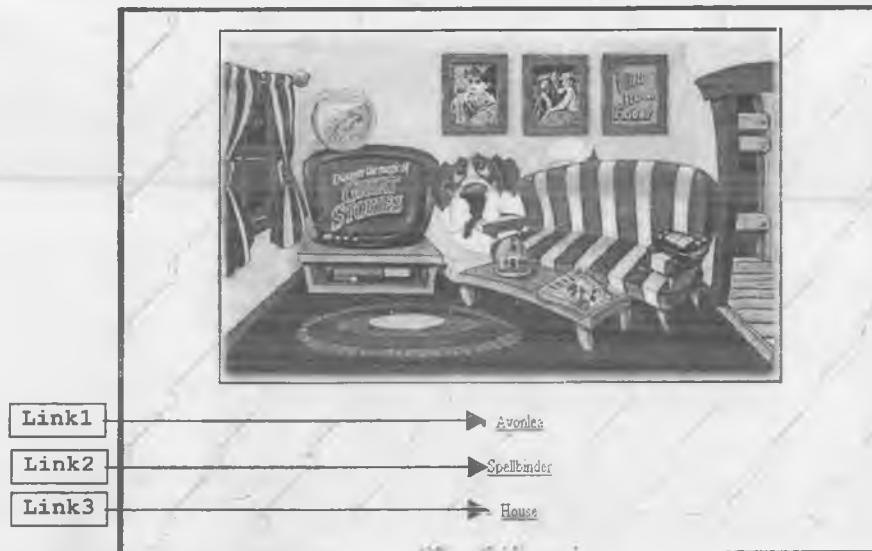


Diagram 4.2b: Links of the index18.htm.

The Home page links and their corresponding file names:

Link1	av.htm	Link2	sb.htm	Link3	home.htm
-------	--------	-------	--------	-------	----------

3229

Output For av.htm called from the index18.htm page [Avonlea]



Diagram 4.3a: Pictures of the av.htm.

The **av.html** page objects and their corresponding file names:

Pic1	av2h.gif
------	----------

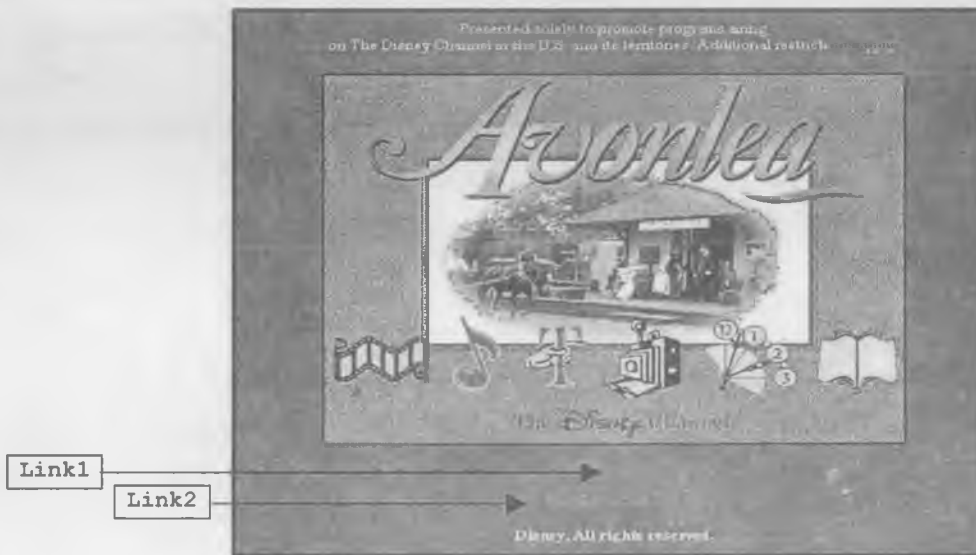


Diagram 4.3b: Links of the av.htm.

The **av.html** page links and their corresponding file names:

Link1	home.htm	Link2	index18.htm
-------	----------	-------	-------------

Output For sb.htm called from the index18.htm page [Spellbinder]

Diagram 4.4a: Pictures of the sb.htm.

The **sb.htm** page objects and their corresponding file names:

Pic1	sb2h.gif
------	----------



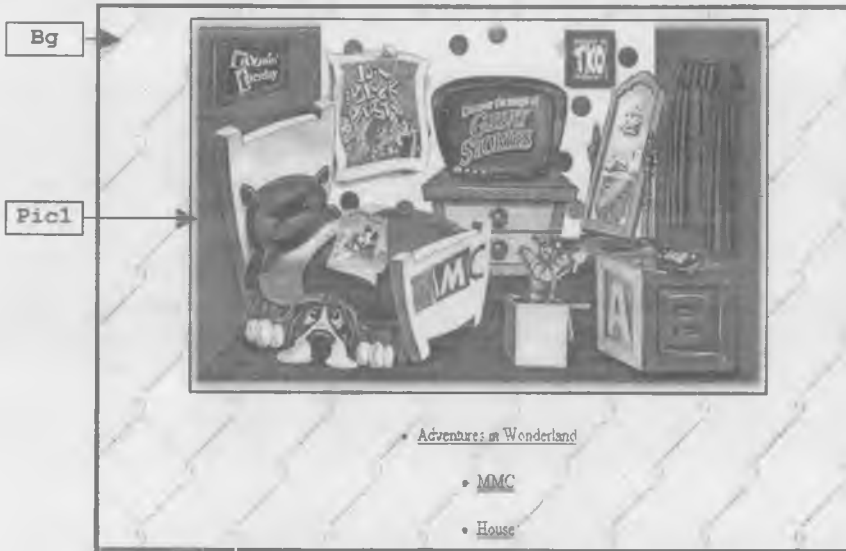
Diagram 4.4b: Links of the sb.htm.

The **sb.htm** page links and their corresponding file names:

Link1	home.htm	Link2	index18.htm
-------	----------	-------	-------------



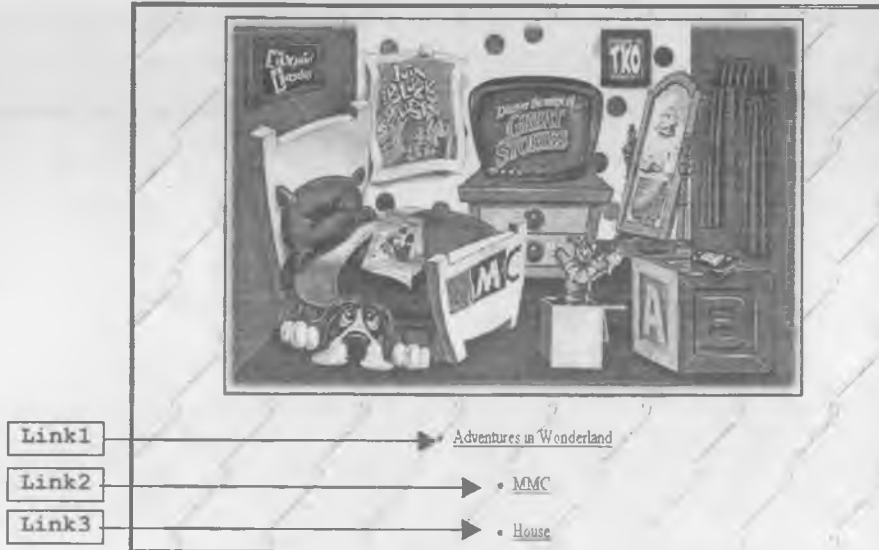
**Output For index19.htm called from the Index page [Programs for Kids]**



**Diagram 4.5a: Pictures of the index19.htm.**

The **index19.htm** page objects and their corresponding file names:

Bg	pinkwhit.gif	Pic1	kids roo.jpg
----	--------------	------	--------------



**Diagram 4.5b: Links of the index19.htm.**

The **index19.htm** page links and their corresponding file names:

Link1	ad.htm	Link2	mm.htm	Link3	home.htm
-------	--------	-------	--------	-------	----------

Output For ad.htm called from the index19.htm page [Adventures in Wonderland]

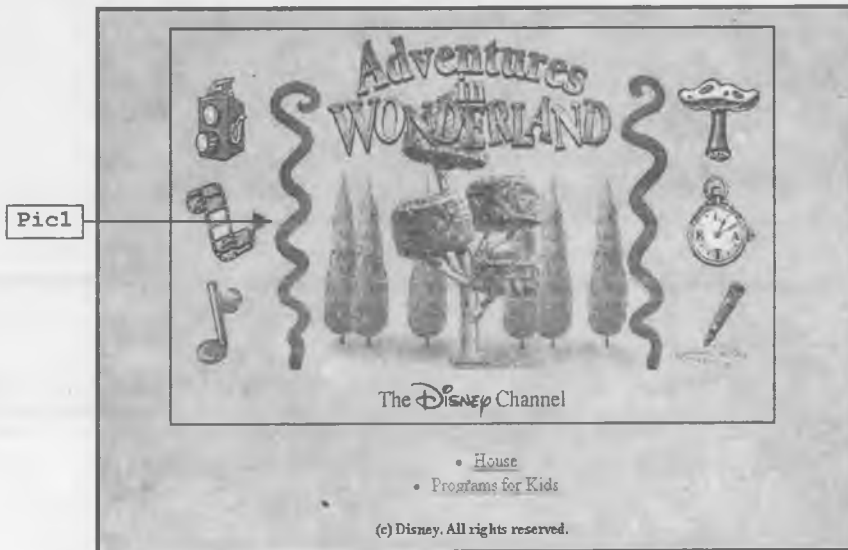


Diagram 4.6a: Pictures of the ad.htm.

The ad.html page objects and their corresponding file names:

Pic1	ad2h.gif
------	----------

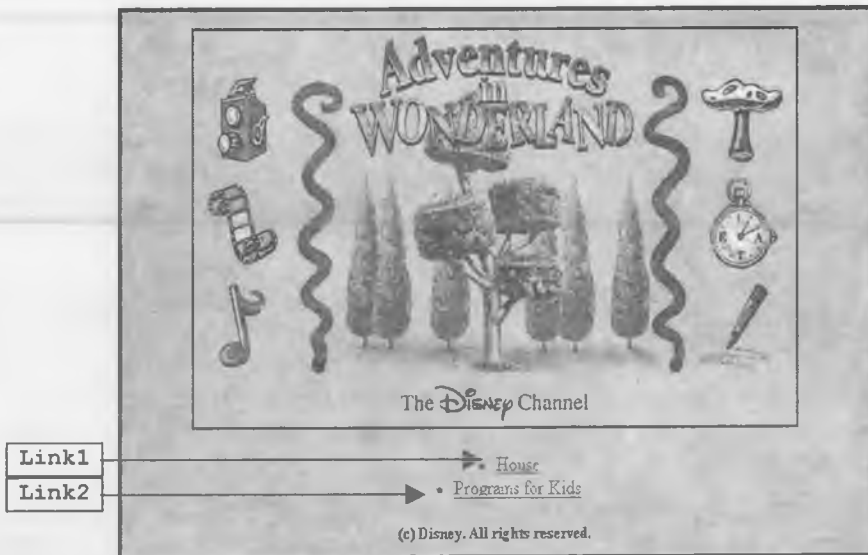
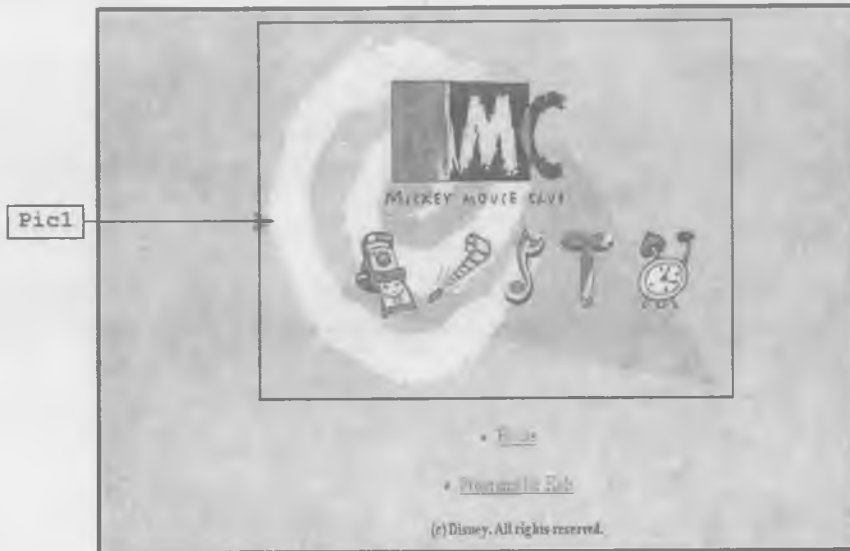


Diagram 4.6b: Links of the ad.htm.

The ad.htm page links and their corresponding file names:

Link1	home.htm	Link2	index19.htm
-------	----------	-------	-------------

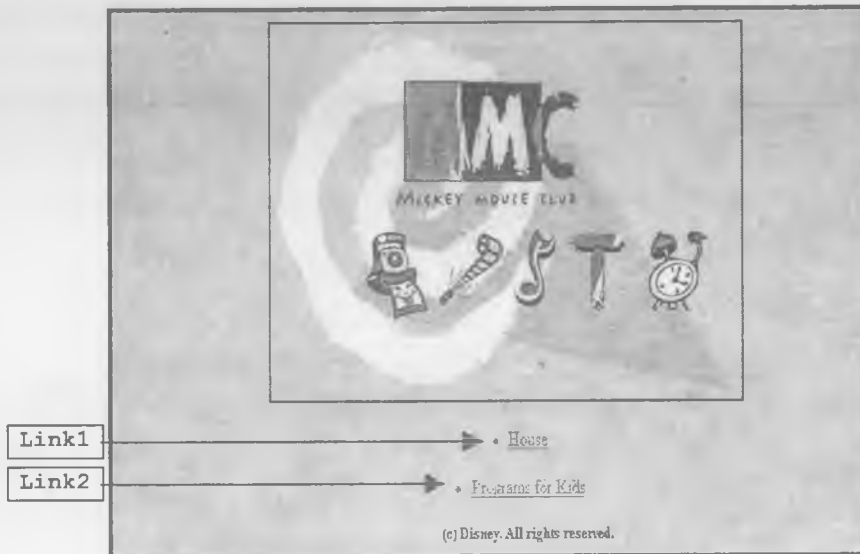
**Output For mm.htm called from the index19.htm page [MMC]**



**Diagram 4.7a:** Pictures of the mm.htm.

The **mm.html** page objects and their corresponding file names:

Pic1	mm2h.gif
------	----------



**Diagram 4.7b:** Links of the mm.htm.

The **mm.htm** page links and their corresponding file names:

Link1	home.htm	Link2	index19.htm
-------	----------	-------	-------------

**Output For index20.htm called from the Index page [Programs for Adults]**



Diagram 4.8a: Pictures of the index20.htm.

The **index20.htm** page objects and their corresponding file names:

Bg	pinkwhit.gif	Pic1	adult ro.jpg
----	--------------	------	--------------

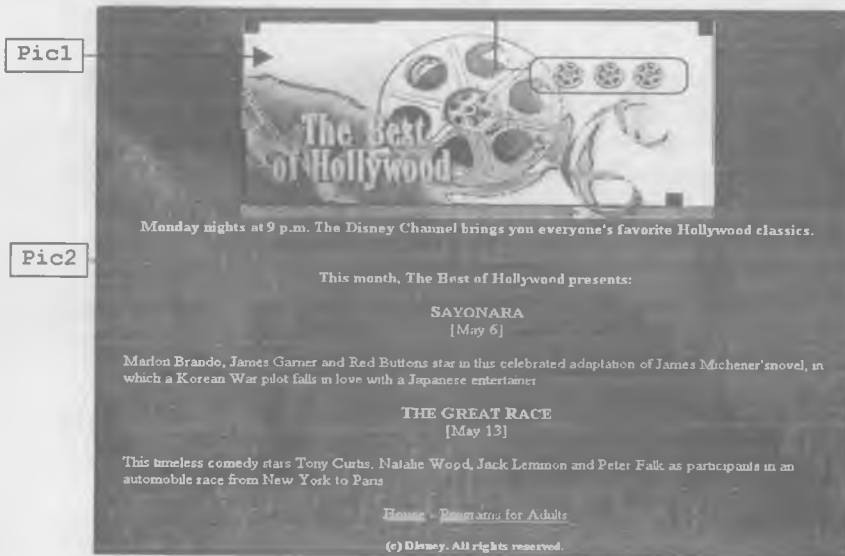


Diagram 4.8b: Links of the index20.htm.

The **index20.htm** page links and their corresponding file names:

Link1	bh.htm	Link2	al.htm	Link3	home.htm
-------	--------	-------	--------	-------	----------

**Output For bh.htm called from the index20.htm page [The Best of Hollywood]**



**Diagram 4.9a:** Pictures of the bh.htm.

The **bh.html** page objects and their corresponding file names:

Pic1	bh2h.gif	Pic2	bhlines2h.gif
------	----------	------	---------------



**Diagram 4.9b:** Links of the bh.htm.

The **bh.htm** page links and their corresponding file names:

Link1	home.htm	Link2	index20.htm
-------	----------	-------	-------------

Output For al.htm called from the index20.htm page [The American Legacy]



Diagram 4.10a: Pictures of the al.htm.

The **al.html** page objects and their corresponding file names:

Pic1	al2h.gif	Pic2	allines2h.gif
------	----------	------	---------------



Diagram 4.10b: Links of the al.htm.

The **al.htm** page links and their corresponding file names:

Link1	home.htm	Link2	index20.htm
-------	----------	-------	-------------

Output For home.htm called from the all page [Programs for the Family]



Diagram 4.11: Pictures of the home.htm.

The **home.htm** page objects and their corresponding file names:

Bg	pinkwhit.gif	Pic1	house sp.gif
----	--------------	------	--------------

# ANSWERS TO SELF REVIEW QUESTIONS

## 1. INTERNET BASICS

### FILL IN THE BLANKS

1. Gateway
2. Web Server Software
3. Physical Domain
4. InterNIC
5. TCP/IP
6. HTTP
7. Index.html

### TRUE OR FALSE

8. True
9. True
10. False

## 2. INTRODUCTION TO HTML

### FILL IN THE BLANKS

1. Hyper Text Markup Language
2. Server, Client, Web Pages, Web Developer
3. Transmission Control Protocol/Internet Protocol
4. <HEAD> </HEAD>, <BODY> </BODY>
5. <P>

### TRUE OR FALSE

6. False
7. True
8. True
9. True
10. False
11. False

## 3. LISTS

### FILL IN THE BLANKS

1. FILLROUND
2. START
3. Definition Description

### TRUE OR FALSE

4. True
5. False
6. False
7. True

## 4. ADDING GRAPHICS TO HTML DOCUMENTS

### FILL IN THE BLANKS

1. .GIF and .JPG
2. <IMG>
3. WIDTH and HEIGHT
4. BORDER
5. VSPACE
6. ALT

### TRUE OR FALSE

7. False
8. True
9. True
10. True



## 5. TABLES

### FILL IN THE BLANKS

1. Rows, Columns
2. <Table>, </Table>
3. CELLSPACING
4. Header, Data
5. ALIGN
6. WIDTH
7. CELLPADDING
8. <TR>

### TRUE OR FALSE

9. True
10. False
11. False
12. True

## 6. LINKING DOCUMENTS

### FILL IN THE BLANKS

1. Hyperlinks
2. Blue
3. External, Internal
4. Named Anchors
5. #
6. Hot regions
7. Rectangle, circle, polygon & default

### TRUE OR FALSE

8. False
9. True
10. False

## 7. FRAMES

### FILL IN THE BLANKS

1. Frames
2. <FRAMESET>
3. <FRAME>
4. SRC
5. NORESIZE
6. SCROLLING
7. NAME

### TRUE OR FALSE

8. False
9. False
10. True

# SOLUTIONS TO HANDS ON EXERCISES

## 2. INTRODUCTION TO HTML

```
<HTML>
```

```
<HEAD><TITLE>Password hackers and crackers!</TITLE></HEAD>
```

```
<BODY BgColor="Beige">
```

```
<H2><I><CENTER>Beware of password hackers and crackers!</CENTER></I></H2><HR>
```

```
<H4>The threat of hackers and crackers is for real, and is alike for everyone. If the security breach at the Bhabha Atomic Research Center last year by the hacker group milwOrM didn't convince you of their capabilities, then consider this: James Davis aka jdavis_4, a resident of <B><FONT Face="Comic Sans MS" Color=RED> www.geocities.com/SiliconValley/Bit/2483 </B></FONT> homestead community on the Internet, might just be reading your e-mail in your mailbox right now, or using your account for some benign Internet surfing, or exchanging it with someone else on the Net for more such accounts!<BR> Surprised? Or scared? <P>Whatever it is, jdavis_4 claims in his home page-Free VSNL passwords-that he has got exactly 108 working VSNL passwords (and growing daily), <I>36 Satyam, 31 MTNL, 11 WmiNet and 23 Mantra Online</I> passwords.</H4>
```

```
</BODY>
```

```
</HTML>
```

## 3. LISTS

```
<HTML>
```

```
<HEAD><TITLE>Lists</TITLE></HEAD>
```

```
<BODY>
```

```
// <B>Example on Unordered List</B> //
```

```
<UL TYPE=FILLROUND>
```

```
<LI>Sportstar
```

```
<LI>Business Week
```

```
<LI>Time
```

```
</UL>
```

```
// <B>Example on Ordered List</B> //
```

```
<OL TYPE="i" START=4>
```

```
<LI> Sportstar
```

```
<LI> Business Week
```

```
<LI> Time
```

```
</OL>
```

```
// <B>Example on Definition List</B> //
```

```
<DL>
```

```
<DT>Sports Magazine<DD>Sportstar
```

```
<DT>Business Magazine<DD>Business Week
```

```
<DT>General Magazine<DD>Time
```

```
</DL>
```

```
</BODY>
```

```
</HTML>
```

## 4. ADDING GRAPHICS TO HTML DOCUMENTS

```

<HTML>
<HEAD><TITLE>WORLD WIDE WEB</TITLE></HEAD>
<BODY BackGround="../images/texture1.gif">
  <CENTER><B><I>THE INTERCITY WEB</I></B><BR>
    <IMG Src="../images/House.GIF" Border=4><BR><BR></CENTER>
  <IMG Align=LEFT Height=100 Src="../images/Javacup.GIF" Width=100>
  <B><I>THE WORLD WIDE WEB</I></B><BR>
  <I>The World Wide Web, or Simply the Web, has been a 'Killer App' of the Internet. Certainly its
  capability to display text and graphics and provide access to other pages and information has made
  it the fastest growing component of the Internet. Major online services are America Online,
  CompuServe and Prodigy.</I><BR><BR><BR>
  <IMG Align=RIGHT Height=100 Src="../images/Computer.GIF" Width=100>
  <B><I>GET CONNECTED</I></B><BR>
  <I>A way to access the web is to get an account with Internet Service Provider, or ISP. These
  accounts include accounts include access to the World Wide Web and other Internet resources,
  and often provide space to store Web pages one will create. A very helpful feature of the web is
  the capability to move from page to page by selecting specific highlighted words and prases or
  images, which are called as links.</I><BR><BR><BR>
</BODY>
</HTML>

```

## 5. TABLES

```

<HTML>
<HEAD><TITLE>Table Test</TITLE></HEAD>
<BODY BackGround="../images/texture1.gif">
  <TABLE Align=CENTER Border=2 CellPadding=3 >
    <CAPTION><B>Time Table And Fare List<B></CAPTION><TR>
      <TH RowSpan=2>Name of Train</TH>
      <TH RowSpan =2>Place</TH>
      <TH RowSpan =2>Destination</TH>
      <TH ColSpan=2> Time</TH>
      <TH RowSpan =2>Fare</TH>
    </TR><TR><TH>Arrival</TH><TH>Departure</TH></TR><TR>
      <TD>Rajdhani Express</TD>
      <TD Align=CENTER>Bombay</TD>
      <TD Align=CENTER>Delhi</TD>
      <TD Align=CENTER>07.30</TD>
      <TD Align=CENTER>08.45</TD>
      <TD Align=RIGHT>Rs 989.00</TD>
    </TR><TR>
      <TD>Madras Mail</TD>
      <TD Align=CENTER>Bombay</TD>
      <TD Align=CENTER>Madras</TD>
      <TD Align=CENTER>09.00</TD>
      <TD Align=CENTER>10.15</TD>
      <TD Align=RIGHT>Rs 450.00</TD>
    </TR><TR>

```

```

<TD>Konya Express</TD>
<TD Align=CENTER>Bombay</TD>
<TD Align=CENTER>Banglore</TD>
<TD Align=CENTER>11.30</TD>
<TD Align=CENTER>12.25</TD>
<TD Align=RIGHT>Rs 645.00</TD>
</TR><TR>
<TD>Konkan Express</TD>
<TD Align=CENTER>Bombay</TD>
<TD Align=CENTER>Manglore</TD>
<TD Align=CENTER>13.30</TD>
<TD Align=CENTER>14.45</TD>
<TD Align=RIGHT>Rs 756.00</TD>
</TR><TR>
<TD>Deccan Express</TD>
<TD Align=CENTER>Bombay</TD>
<TD Align=CENTER>Pune</TD>
<TD Align=CENTER>16.00</TD>
<TD Align=CENTER>17.30</TD>
<TD Align=RIGHT>Rs 345.00</TD>
</TR></TABLE>
</BODY>
</HTML>

```

## 6. LINKING DOCUMENTS

### Code For Menu.htm:

```

<HTML>
<BODY>
<FONT Color="Green" FACE="Arial">Welcome to our homepage.<BR>This page has links to
the website of </FONT><FONT Color="Brown" Face="ZappedChancellor" Size=5><B> ABC
Lever Inc.</B></FONT>
<P><FONT Color="Green" Face="Arial">
For further information click on any one of the following:
<UL><LI><A HRef="AboutUs.html">About ABC Lever Inc.</A>
<P><LI><A HRef="AboutUs.html#SECTION 1">Contact information</A></UL>
</FONT>
</BODY>
</HTML>

```

### Code for AboutUs.htm

```

<HTML>
<HEAD><TITLE> ABC Lever Inc. </TITLE></HEAD>
<BODY>
<CENTER><FONT Color="Brown" Face="Brush Script MT" Size=6>
<IMG Align=BOTTOM Src=" ../images/corp.gif" ><I>ABC Lever Inc.</I>
</FONT></CENTER>
<A HRef="#SECTION 1">Contact us</A><HR><BR>

```

```

<FONT Color="Blue" Face="Arial" Size=3><B>ABC Lever Inc. is a conglomerate that has
interests ranging from bodycare products to toilet soaps.<BR>A couple of years ago we entered
the frozen food industry through mergers and acquisitions.<BR>Last year we started our first plant
to manufacture salt and this year it is wheat flour.<BR>Our current turnover is about Rs. 7500 cr
and by the next decade we are looking at a target of 15000 cr.</FONT><P><HR>
<P><A Name="SECTION 1"><FONT SIZE=5 COLOR="Brown">
  <B><I>Contact Us </I></B>
</FONT></A><HR>
<FONT SIZE=3 COLOR="Green">You can contact us at the following address:</FONT>
<P><FONT Color="Blue" Face="Arial"><B> ABC Lever Inc.<BR>
  101 Maker Chambers III,<BR>Nariman Point,<BR>Mumbai-21<BR>Tel 2012011
<P>You can also email us at customerservice@abclever.com</P></B></FONT></P>
</BODY>
</HTML>

```

## 7. FRAMES

### Code for Handsonframe.htm

```

<HTML>
  <HEAD><TITLE> Hands on</TITLE></HEAD>
  <FRAMESET Cols = "35%,*">
    <FRAME Name="First" Src="Menu.html">
    <FRAME Name="Second" Src="Desc.html">
  </FRAMESET>
</HTML>

```

### Code for Menu.htm

```

<HTML>
  <BODY ><FONT Color="Green" Face="Arial">Click on any one of the following:<UL>
    <LI><A HRef="AboutUs.htm" Target="Second">About us</A>
    <P><LI><A HRef="Contact.htm" Target="Second">Contact us</A>
  </UL></FONT></BODY>
</HTML>

```

### Code for Desc.htm

```

<HTML>
  <HEAD><TITLE> Welcome</TITLE></HEAD>
  <BODY><FONT Color="Brown" Size=5><B><I>Welcome</I></B></FONT><HR>
    <FONT FACE="Arial" SIZE=3 COLOR="Green">Thank you for visiting our website.
    <P>Click on any of the choices on the left.
  </BODY>
</HTML>

```

### Code for AboutUs.htm

```

<HTML>
  <HEAD><TITLE>ABC Lever Inc.</TITLE></HEAD>
  <BODY><CENTER>
    <FONT Color="Brown" Face="Brush Script MT" Size=6>
    <IMG Align=Bottom Src="./images/corp.gif"><I>ABC Lever Inc.</I></FONT>

```

```
<HR><BR>
<FONT Color="Blue" Face="Arial" Size=3>
  <B>ABC Lever Inc. is a conglomerate that has interests ranging from bodycare products to
  toilet soaps. A couple of years ago we entered the frozen food industry through mergers and
  acquisitions. Last year we started our first plant to manufacture salt and this year it is wheat
  flour. Our current turnover is about Rs. 7500 cr and by the next decade we are looking at a
  target of 15000 cr.</B>
</FONT>
</CENTER></BODY>
</HTML>
```

### Code for Contact.htm

```
<HTML>
  <HEAD><TITLE>Contact Us </TITLE></HEAD>
  <BODY><FONT SIZE=5 COLOR="Brown"><B><I>Contact Us </I></B></FONT><HR>
    <FONT SIZE=3 COLOR="Green">You can contact us at the following address:</FONT>
    <P><FONT FACE="Arial" COLOR="Blue"><B>ABC Lever Inc.<BR>
      101 Maker Chambers III,<BR>Nariman Point,<BR>Mumbai-21<BR>Tel 2012011
    <P>You can also email us at customerservice@abclever.com</P></B></FONT>
  </BODY>
</HTML>
```

## *SECTION - II: JavaScript*

### 8. INTRODUCTION TO JAVASCRIPT

#### JAVASCRIPT IN WEB PAGES

Today's web sites need to go much beyond HTML. There is a definite need to allow users, browsing through a web site, to actually interact with the web site. The web site must be intelligent enough to accept users input and dynamically structure web page content, tailor made, to a user's requirements.

This may be as simple as ensuring a web page delivered to a user, having a background color that the user is comfortable with or as complex as delivering a web page with special textual formatting for a user with visual disabilities.

Users, who browse through a web site today, prefer to choose to view what interests them. Hence even the **content** of a web page needs to be dynamic, based on what a user wishes to see.

This requires a web site development environment that will allow the creation of **Interactive** web pages. These web pages will accept input from a user. Based on the input received return customized web pages, both in content and presentation, to the user.

In the absence of any user input the web site must be intelligent enough to return a default web page containing predetermined information and text formatting.

This calls for a web site development environment with coding techniques capable of accepting a client's requests and processing these requests. The result of the processing being passed back to the client via standard HTML pages.

The need to return standard HTML pages, that map to a user's input, is due to the fact that browsers use HTTP to communicate with a web server and are designed to interpret and render HTML on a client's machine.

Capturing user requests is traditionally done via a **Form**. Hence the web site development environment needs to have the facilities to create forms.

After a form captures user input, the form must have a *built in* technique for sending the information captured back to the web server for processing. Processing user requests is generally done via scripts (*small programs*) that are based on the server.

The web site development environment should also provide the facility for **Validating** user input. Invalid user input, will either cause data to be sent back from the web server to the browser, which is not what the user wants or give rise to an error message being sent back to the browser from the web server. Neither of which would really attract repeat visits to the web site.

Hence, the web site development environment must also facilitate coding which runs in a browser at client side for data validation. Most development environments offer standard constructs for data validation. Standard programming constructs are:

- Condition checking constructs
- Case checking constructs
- Super controlled Loop constructs

Additionally, all development environments provide syntax to create and use memory variables, constants, and functions.

If the development environment is object oriented it will provide an object hierarchy to work with. An **Object Oriented Programming (OOP)** environment always offers event driven programming. This means that the programming environment will recognize object based events and allow the connection of code snippets to these events. When an event occurs, the code snippets will execute.

All these facilities and more are available in JavaScript.

## Netscape and JavaScript

JavaScript is a scripting language (*web site development environment*) created by Netscape hence JavaScript works best with the Netscape suite of Client and Server products.

The Netscape client **browser** product is called Netscape Communicator. The default scripting language that Netscape Communicator understands is JavaScript.

One Netscape server product, from its suite of server products, is Netscape Commerce Server. The default scripting language that Netscape Commerce Server understands is JavaScript.

## Database Connectivity

Netscape has a product called **Live Wire**, which permits server side, JavaScript code, to connect to **Relational DataBase Management Systems (RDBMS)**. RDBMS such as Oracle, MS SQL Server, MySQL, mSQL and a host of other widely used relational databases, which generally use ANSI SQL as their natural language. **Live Wire** database drivers also support a number of non-relational databases.

## Client side JavaScript

Client-side JavaScript is traditionally embedded into a standard HTML program. JavaScript is embedded between the `<SCRIPT> . . . </SCRIPT>` HTML tags. These tags are embedded within the `<HEAD> . . . </HEAD>` or `<BODY> . . . </BODY>` tags of the HTML program.

JavaScript is embedded into an HTML program because JavaScript uses the *filename.html* and the HTTP protocol to transport itself from the web server to the client's browser where the JavaScript executes and processes client information.

Only a browser that is JavaScript enabled will be able to interpret JavaScript code. Netscape Communicator does this best as JavaScript is the natural language of Netscape Communicator.

Microsoft's Internet Explorer also interprets JavaScript. However, Internet Explorer latest releases support an old version of JavaScript. Hence, the total functionality of the latest release of JavaScript as is available in Netscape Communicator is not available in Internet Explorer.

## Note



---

The default scripting language of Internet Explorer is VB Script. Netscape Communicator does not support VB Script.

---

## Capturing User Input

Web site interactivity starts from being able to capture user input. The `<FORM> . . . </FORM>` HTML tags can be used to create a user **Request-form**. Between these HTML tags the HTML `<INPUT> </INPUT>` tags can be used to instantiate HTML objects in the HTML form to facilitate the capture of user data.



The HTML objects used in HTML form creation are **Text**, **TextArea**, **Radio Buttons**, **Push Buttons**, **Check Boxes** and so on. These will be passed as **values** to the **TYPE** attribute of the `<INPUT> . . . </INPUT>` tags as will be seen in later chapters. Once the HTML form has been coded in *filename.html* JavaScript can be embedded in the same HTML file to make it interactive and facilitate client side data validation and/or processing.

The concept being that standard HTML form objects are used to *capture* user input while client side, JavaScript (*embedded in the HTML file*) is used to *validate* and/or *process* such user input. The application of *validation rules* to the data captured by HTML form objects and *error handling* is conveniently done using JavaScript which executes in the client's browser.

Once user input passes the validation tests applied, and/or has been processed appropriately by client side, JavaScript, the form data captured will have to be passed backward to the web server from where the HTML file originated.

At the web server an appropriate program (*written in JavaScript, VB Script, CGI-PERL, Java and so on*) will accept this user input. Based on user input information will be assembled on the web server, converted to an HTML file (*page*) and returned to the user. Such an HTML page is created on **Demand**.

An HTML file (*page*) created on demand, based on user input, is a dynamically created HTML page. This then is a clear indication of a dynamic and interactive web site.

HTML itself is static. HTML allows a very minimum interaction with users by providing hyperlinks. Truly interactive pages as described above, cannot be created using standard HTML **Tags** alone. Embedding JavaScript in an HTML program does this

For instance, a Web Site can be created and hosted on a web server to take orders for products. Typically, an HTML form is used to capture 'User orders for products'. At the same time, form data (*the user order's*) validation must be done. For example:

- Orders should be accepted only for products which are available
- Any quantity ordered should not exceed the quantity currently available
- The order date should be set to the current date
- The quantity required cannot be left blank
- The place of delivery (*address*) cannot be left blank

HTML alone cannot do any of this. At the maximum, HTML can provide an elegant interface for capturing **Order** information. HTML (as mentioned earlier) does not provide any techniques for validating user entries. This makes it necessary to introduce client side JavaScript in the HTML (*page*) program, which extends the functionality of the web page by introducing client side data processing.

## JAVASCRIPT

JavaScript is an object-oriented language that allows creation of interactive Web Pages. JavaScript allows user entries, which are loaded into an HTML form to be processed as required. This empowers a web site to return site information according to a user's requests.

JavaScript offers several advantages to a Web developer such as a short development cycle, ease of learning, small size scripts and so on. The strengths of JavaScript can be easily and quickly used to extend the functionality of HTML pages, already on a web site.

### The Advantages Of JavaScript

**An Interpreted Language:** JavaScript is an interpreted language, which requires no compilation steps. This provides an easy development process. The syntax is completely interpreted by the browser just as it interprets HTML tags.

**Embedded Within HTML:** JavaScript does not require any special or separate editor for programs to be written, edited or compiled. It can be written in any text editor like Notepad, along with appropriate HTML tags, and saved as *filename.html*. HTML files with embedded JavaScript commands can then be read and interpreted by any browser that is JavaScript enabled.

**Minimal Syntax - Easy to Learn:** By learning just a few commands and simple rules of syntax, complete applications can be built using JavaScript.

**Quick Development:** Because JavaScript does not require time-consuming compilations, scripts can be developed in a short period of time. This is enhanced by the fact that many GUI interface features, such as alerts, prompts, confirm boxes, and other GUI elements, are handled by client side JavaScript, the browser and HTML code.

**Designed for Simple, Small Programs:** It is well suited to implement simple, small programs (*for example, a unit conversion calculator between miles and kilometers, or pounds and kilograms*). Such programs can be easily written and executed at an acceptable speed using JavaScript. In addition, they can be easily integrated into a web page.

**Performance:** JavaScript can be written such that the HTML files are fairly compact and quite small. This minimizes storage requirements on the web server and download time for the client.

Additionally, because JavaScript programs are usually included in the same file as the HTML code for a web page, they require fewer separate network accesses.

**Procedural Capabilities:** Every programming language needs to support facilities such as Condition checking, Looping and Branching. JavaScript provides syntax, which can be used to add such procedural capabilities to web page (*filename.html*) coding.

**Designed for Programming User Events:** JavaScript supports Object/Event based programming. JavaScript recognizes when a form **Button** is pressed. This event can have suitable JavaScript code attached, which will execute when the **Button Pressed** event occurs.

JavaScript can be used to implement context sensitive help. Whenever an HTML form's **Mouse** cursor **Moves Over** a button or a link on the page a helpful and informative message can be displayed in the status bar at the bottom of the browser window.

**Easy Debugging and Testing:** Being an interpreted language, scripts in JavaScript are tested line by line, and the errors are also listed as they are encountered, i.e. an appropriate error message along with the line number is listed for every error that is encountered. It is thus easy to locate errors, make changes, and test it again without the overhead and delay of compiling.

**Platform Independence / Architecture Neutral:** JavaScript is a programming language that is completely independent of the hardware on which it works. It is a language that is understood by any JavaScript enabled browser. Thus, JavaScript applications work on any machine that has an appropriate JavaScript enabled browser installed. This machine can be anywhere on the network.

Since each browser is for a specific platform, JavaScript interpretation will be with respect to the specific platform. The browser will add whatever platform specific information is required to the JavaScript while it interprets the code. Thus, JavaScript is truly platform independent. A JavaScript program developed on a Unix machine will work perfectly well on a Windows machine.

The fact that a platform specific browser, maintained at the client end, does the interpretation of JavaScript, relieves the developer of the responsibility of maintaining multiple source code files for multiple platforms.

## WRITING JAVASCRIPT INTO HTML

JavaScript syntax is embedded into an HTML file. A browser reads HTML files and interprets HTML tags. Since all JavaScripts need to be included as an integral part of an HTML document when required, the browser needs to be informed that specific sections of HTML code is JavaScript. The browser will then use its built-in JavaScript engine to interpret this code.

The browser is given this information using the HTML tags `<SCRIPT> . . . </SCRIPT>`. The `<SCRIPT>` tag marks the beginning of a snippet of scripting code. The paired `</SCRIPT>` marks the end of the snippet of scripting code.

Like most other HTML tags, the `<SCRIPT>` tag takes in an optional attribute, as listed below:

Attributes	Description
Language	Indicates the scripting language used for writing the snippet of scripting code. If left undefined Netscape Communicator will assume JavaScript. If left undefined Internet Explorer will assume VB Script

Table 8.1

### Syntax:

```
<SCRIPT LANGUAGE= "JavaScript">
  // Javascript code snippet written here
</SCRIPT>
```

## BASIC PROGRAMMING TECHNIQUES

JavaScript offers the very same programming capabilities found in most programming languages. Creating variables, constants, programming constructs, user defined functions and so on.

All these programming techniques can be used in JavaScript embedded in any HTML program. These are the techniques that make JavaScript an exciting programming language that extends the functionality of HTML and makes web pages interactive.

The `<HEAD> . . . </HEAD>` tags make an ideal place to create JavaScript variables and constants and so on. This is because the head of an HTML document is always processed before the body. Placing JavaScript memory variables, constants and user defined JavaScript functions in this section of an HTML document will cause them to be defined (*by the JavaScript interpreter*) before being used. This is important because any attempt to use a variable (*or any other JavaScript object*) before it is defined, results in an error.

Variables are used to store values that can be used in other parts of a program. Variables always have a name associated with them via which they can be referenced later. When naming variables, it is good programming practice to structure a descriptive name.

Variable names can begin with an uppercase letter (A through Z), lowercase letter (a through z), and underscore character (`_`) or dollar sign character (`$`). The remaining characters can consist of letters, the underscore character, the dollar sign character or digits 0 to 9. Variable names are case sensitive.

### Caution



The dollar sign (\$) character is reserved for machine generated code and should not be used in scripts. In particular it should not be used in scripts that will be run by earlier browsers that are not fully ECMAScript-compatible.

If two words are used to name a variable then it is a programming convention to start the first letter of the first word in lower case and the first letter of the second word in uppercase for example, *variableName*.

JavaScript does not allow the data type of the variable to be declared when a variable is created. The same variable may be used to hold different types of data at different times when the JavaScript code snippet runs.

## Data Types And Literal

JavaScript supports four primitive types of values and supports complex types such as arrays and objects. Primitive types are types that can be assigned a single literal value such as number, string or boolean value. Literals are fixed values, which *literally* provide a value in a program.

The primitive data types that JavaScript supports are:

### Number

Consists of integer and floating point numbers and the special **NaN** (*not a number*) value.

Integer literals can be represented in JavaScript in decimal, hexadecimal, and octal form.

### *Note*



Hexadecimal and octal integers are converted to decimal before they are displayed.

Floating-point literals are used to represent numbers that require the use of a decimal point, or very large or very small numbers that must be written using exponential notation. A floating-point number must consist of either a number containing a decimal point or an integer followed by an exponent.

For example, 33, 12.10, -35.8, 2E3, 0x5F

### Boolean

Consists of the logical value *true* and *false*.

JavaScript supports a pure Boolean type that consists of the two values *true* and *false*. Logical operators can be used in Boolean expressions.

JavaScript automatically converts the Boolean values *true* and *false* into 1 and 0 when they are used in numerical expressions.

### *Note*



Values 1 and 0 are not considered Boolean values in JavaScript.

### String

Consists of string values that are enclosed in single or double quotes.

JavaScript provides built-in support for strings. A string is a sequence of zero or more characters that are enclosed by double (") or single (') quotes. If a string begins with a double quote it must end with a double quote. If a string begins with a single quote it must end with a single quote.

For example, "Rahul", '24, Sanjay Nagar, Bangalore'

## Note



If a string has to include quote character in the string the quote character must be preceded by the backslash (\) escape character.

## Null

Consists of a single value, *null*, which identifies a null, empty or nonexistent reference.

The null value is common to all JavaScript types. It is used to set a variable to an initial value that is different from other valid values. Use of the null value prevents the sort of errors that result from using un-initialized variables. The null value is automatically converted to default values of other types when used in an expression.

## Type Casting

In JavaScript variables are loosely cast. The type of a JavaScript variable is **implicitly defined** based on the **literal** values that are assigned to it from time to time.

For instance, combining the string literal "Total amount is " with the integer literal 1000 results in a string with the value "Total amount is 1000". By contrast, adding together the numeric literal 10.5 and the string "20" results in the floating point integer literal 30.5. This process is known as **type casting**.

## Creating Variables

In order to make working with data types convenient, variables are created. In JavaScript variables can be created that can hold any type of data.

In order to use a variable, it is good programming style to declare it. Declaring a variable tells JavaScript that a variable of a given name exists so that the JavaScript interpreter can understand references to that variable name throughout the rest of the script.

Although it is possible to declare variables by simply using them, declaring variables helps to ensure that programs are well organized and helps keep track of the scope of the variables. Variables can be declared using **var** command.

### Syntax:

```
var <variable name> = value;
```

### Examples:

```
var first_name;  
var last_name = "Shah";  
var phone = 6128879;
```

The equal sign (=) used in assigning a value to a variable is known as an *assignment* operator.

## Note



Like properties and method names in JavaScript, variable names are case sensitive.

## Incorporating Variables In A Script

### Example 1:

The following illustrates incorporating variables in a script.

```
<HTML>
  <HEAD>
    <SCRIPT Language = JavaScript>
      var name = prompt("Enter your name", "Name");
    </SCRIPT>
  </HEAD>
  <BODY>
    <SCRIPT Language = "JavaScript">
      document.write("<H2> Hello " + name + "</H2>");
    </SCRIPT>
  </BODY>
</HTML>
```

The JavaScript **prompt()** method picks up a string (i.e. *User Name*) from the user which is then assigned to the variable *name*. The JavaScript code **document.write()** embedded in the **<BODY> . . . </BODY>** tags writes the contents of the variable *name* to the client browser.

Since the **<HEAD> . . . </HEAD>** section of the HTML program is interpreted first a 'User Name' is picked up first before anything is displayed in the client browser.

## The JavaScript Array

Arrays are JavaScript objects that are capable of storing a sequence of values. These values are stored in indexed locations within the array. The length of an array is the number of elements that an array contains. The individual elements of an array are accessed by using the name of the array followed by the *index value* of the array element enclosed in square brackets.

### Note



The array element index starts with 0. Hence the last array element index number is one less than the length of the array.

An array must be declared before it is used. An array can be declared using any one of the following techniques.

```
arrayName = new Array(Array length)
arrayName = new Array( )
```

In the first example the array size is explicitly specified. Hence this array will hold a pre-determined set of values. The second example creates an array of the size 0.

### Note



JavaScript automatically extends the length of any array when new array elements are initialized.

**Example:**

```

cust_Orders = new Array()
cust_Orders[50] = "Lion Pencils"
cust_Orders[100] = "Steadler eraser"

```

When JavaScript encounters the reference to order [50], in the above example, it will extend the size of the array `cust_Orders` to 51 and initialize `order[50]`. When JavaScript encounters the reference to order [100], in the above example, it will extend the size of the array `cust_Orders` to 101 and initializes `order[100]`.

Even if an array is initially created of a fixed length it may still be extended by referencing elements that are outside the current size of the array. This is done in the same manner as with zero-length arrays.

**Dense Arrays**

A dense array is an array that has been created with each of its elements being assigned a specific value. Dense arrays are used exactly in the same manner as other arrays. They are declared and initialized at the same time.

Listing the values of the array elements in the array declaration creates dense arrays. For example a dense array can be created as:

```
arrayName = new Array(value0, value1, . . . , valuen)
```

In this array, since the element count starts from 0 to  $n$ , the array length is  $n+1$ .

Since array is a JavaScript object, arrays have several methods associated with them via which the array and its element content can be manipulated.

**Join()** returns all elements of the array joined together as a single string. This takes one argument; a string to be used as a separator between each element of the array in the final string. If the argument is omitted, `join()` uses a comma-space as the separator.

**Reverse()** reverses the order of the elements in the array.

**Example 2:**

An array is used with hard coded values. Displaying the values of the array elements in the browser makes use of an array's **join()** method to print the array elements in a single line..

<HTML>

```
<HEAD><TITLE>Viewing the array elements of a JavaScript Array</TITLE></HEAD>
```

```
<BODY>
```

```
  <SCRIPT language = "JavaScript">
```

```
    <!-- Begin Hiding JavaScript
```

```
      friends = new Array(5);
```

```
      friends[0] = "Ananth";
```

```
      friends[1] = "Cedric";
```

```
      friends[2] = "Ketan";
```

```
      friends[3] = "Rohan";
```

```
      friends[4] = "Leela";
```

```
      document.write(friends[0] + "<BR>");
```

```
      document.write(friends[1] + "<BR>");
```

```
      document.write(friends[2] + "<BR>");
```

```
      document.write(friends[3] + "<BR>");
```

```
      document.write(friends[4] + "<BR>");
```

```

    join_crit = friends.join();
    document.write(join_crit);
    // End hiding JavaScript -->
</SCRIPT>
</BODY>
</HTML>

```

### The Elements Of An Array

JavaScript does not place any restrictions on the values assigned to the elements of an array. These values could be of different types or could refer to other arrays or objects.

#### Example:

```
multiTypeArray = new Array( "Val1", "Val2", 1, 2, true, false, null, new array(3, 4))
```

The array named multiTypeArray has a length of eight and its elements are:

```

multiTypeArray[0] = "Val1"
multiTypeArray[1] = "Val2"
multiTypeArray[2] = 1
multiTypeArray[3] = 2
multiTypeArray[4] = true
multiTypeArray[5] = false
multiTypeArray[6] = null
multiTypeArray[7] = a new dense array consisting of the values of 3,4

```

The last element of the array, multiTypeArray, contains a dense array as its value. The two elements of this array can be accessed using a *second set of subscripts*:

```

num1 = multiTypeArray[7][0];
num2 = multiTypeArray[7][1];

```

### The JavaScript Array and its length Property

JavaScript arrays are implemented as objects. *Objects* are named collections of data that have **properties** and whose values may be accessed via **methods**. A *property* returns a value that identifies some aspect of the state of an object. *Methods* are used to read or modify the data contained in an object's property.

The length of an array is a property of an array. Access to any JavaScript object's property is done by using **objectname.propertyname**.

For example, to find out the length of the multiTypeArray:

```
myvar = multiTypeArray.length;
```

The length of the multiTypeArray will be assigned to the variable *myvar*. By accessing the contents of myvar the length of (*no. of elements in multiTypeArray*) the multiTypeArray can be determined.

## **OPERATORS AND EXPRESSIONS IN JAVASCRIPT**

An *operator* is used to transform one or more values into a single resultant value. The values to which the operator is applied is referred to as *operands*. A combination of an operator and its operands is referred to as an *expression*.

Expressions are evaluated to determine the value of the expression. This is the value that results when an operator is applied to its operands.



*Note*



For some operators, such as multiplication (\*) the *order* of the operands do not matter. For example,  $A * Y = Y * A$  is *true* for all integers and floating point numbers.

Other operators such as string concatenation (+), the order of the operands matter. For example, "ab" + "cd" is not the same as "cd" + "ab".

**Arithmetic Operators**

Arithmetic operators are the most familiar operators because they are used every day to solve common math calculations. The arithmetic operators that JavaScript supports are;

Operator	Description	Operator	Description
+	Addition	-	Subtraction or Unary negation
*	Multiplication	/	Division
%	Modulus	++	Return the value then Increment
--	Return the value then Decrement		

Table 8.2

*Note*



The % (modulus) operator calculates the remainder by dividing two integers. For example,  $17 \% 3 = 2$  because  $17 / 3 = 5$  with a remainder of 2.

An operator requiring a single operand is referred to as a **Unary** operator and one that requires two operands is a *binary* operator.

The above standard arithmetic operators are binary operators. In addition to these binary operators, there are unary arithmetic operators. They are (++) and (--).

Both these increment and decrement operators can be used in two different ways. Before the operand or after the operand. For example, ++x increments x by **one** and returns the result, while x++ returns x and then increments the value of x by one. Similarly, --x decreases the value of x by **one** before returning a result, while x-- returns the value of x before decreasing its value by one.

**Example:**

```
X = 3;
Y = ++X;
Z = X++;
```

In these lines of code, X is first assigned the value of 3, which is then increased to 4 and assigned to Y. The new value of 4 is assigned to Z, and then the value of X is increased to 5. Finally, X is 5, Y is 4 and Z is 4.

**Logical Operators**

Logical operators are used to perform Boolean operators on Boolean operands **AND**, **OR**, **NOT**. The logical operators supported by JavaScript are:

Operator	&&		!
Description	Logical AND	Logical OR	Logical NOT

Table 8.3

## Comparison Operators

Comparison operators are used to compare two values. The comparison operators supported by JavaScript are:

Operator	Description	Operator	Description
==	Equal	===	Strictly Equal
!=	Not Equal	!==	Strictly Not Equal
<	Less than	<=	Less than or equal to
>	Greater than	>=	Greater than or equal to

Table 8.4

### Note



The equal (==) and not equal (!=) operators perform type conversions before testing for equality. For example, "5" == 5 evaluate to **true**.

The strictly equal (===) and the strictly not equal (!===) do not perform type conversions before testing for equality. For example, "5" === 5 will return the value **false**.

## String Operators

String operators are those operators that are used to perform operations on strings. Currently JavaScript supports only the **string concatenation** (+) operator.

This operator is used to join two strings. For example, "pq" + "ab" produces "pqab".

## Assignment Operators

The assignment operator is used to update the value of a variable. Some assignment operators are combined with other operators to perform a computation on the value contained in a variable and then update the variable with the new value. Some of the assignment operators supported by JavaScript are:

Operator	Description
=	Sets the variable on the left of the = operator to the value of the expression on its right
+=	Increments the variable on the left of the += operator by the value of the expression on its right. When used with strings, the value to the right of the += operator is appended to the value of the variable on the left of the += operator
-=	Decrements the variable on the left of the -= operator by the value of the expression on the right
*=	Multiplies the variable on the left of the *= operator by the value of the expression on its right
/=	Divides the variable on the left of the /= operator by the value of the expression on its right
%=	Takes the modulus of the variable on the left of the %= operator using the value of the expression on its right

Table 8.5

## The Conditional Expression Ternary Operator

JavaScript supports the conditional expression operator. They are ? and : The conditional expression operator is a ternary operator since it takes three operands, a condition to be evaluated and two alternative values to be returned based on the truth or falsity of the condition.

**Syntax:**

condition ? value1 : value2

The condition expressed must return a value true or false. If the condition is true, value1 is the result of the expression, otherwise value2 is the result of the expression.

**Special Operators**

JavaScript supports a number of special operators that do not fit into the operator categories covered above.

**The delete Operator**

The delete operator is used to delete a property of an object or an element at an array index.

**Example:**

To delete the **sixth** element of myArray.

delete myArray[5]

**The new Operator**

The new operator is used to create an instance of an object type.

**Example:**

To create a new JavaScript object of the type array and assign this array to a context area in memory called myArray.

myArray = new Array( )

**The void Operator**

The void operator does not return a value. It is typically used in JavaScript to return a URL with no value.

**JAVASCRIPT PROGRAMMING CONSTRUCTS**

Most programming languages support a common (**core**) set of constructs. Languages only differ in the syntax used for structuring these constructs.

Languages may also differ in the degree to which they support programming features such as **Object Oriented Programming**, abstract data definition, inference rules and list processing.

JavaScript provides a complete range of basic programming constructs. While it is not an object oriented programming environment JavaScript is an **object-based** language.

The constructs provided by JavaScript are as follows:

Construct / Statement	Purpose	Example
Assignment	Assigns the value of an expression to a variable.	x = y + z
data declaration	Declares a variable and optionally assigns a value to it.	var myVar = 10
if	Program execution depends upon the value of returned by the condition. If the value returned is True the program executes else the program does not execute.	if(x > y) { z = X; }
while	Repeatedly executes a set of statements until a condition becomes false	while( x != 7) { x %= n - n }

Table 8.6

Construct / Statement	Purpose	Example
switch	Selects from a number of alternatives	<pre>switch(val) {   case 1:     // First alternative     break;   case 2:     // Second alternative     break;   default     // Default action }</pre>
for	Repeatedly executes a set of statements until a condition becomes false	<pre>for(i = 0; i &lt; 7; ++i) {   document.write(x[i]); }</pre>
do while	Repeatedly executes a set of statements while a condition is true	<pre>do {   // Statements } while(i &gt; 0)</pre>
label	Associates a label with a statement	<pre>LabelName:   Statement</pre>
break	Immediately terminates a do while or for loop construct	<b>if(x &gt; y) break</b>
continue	Immediately terminates the current iteration of a do, while or for construct	<b>if(x &gt; y) continue</b>
function call	Invokes a function	<b>x = abs(y)</b>
return	Returns a value from a function call	<b>return x*y</b>
with	Identifies the default object	<pre>with(Math) {   d = PI * 2 * r; }</pre>
delete	Deletes an object property or an array element	<b>delete a[5]</b>
Method invocation	Invokes a method of an object	<b>document.write("Hello")</b>

Table 8.6 (continued)

## CONDITIONAL CHECKING

### The if - then - else Statement

The conditional construct in JavaScript offers a simple way to make a decision in a JavaScript program. The conditional construct in JavaScript will either return a **True** or a **False** depending upon how the **condition** evaluated.

Using the **if-else** construct the flow of the JavaScript program can be altered i.e. the **if** condition determines which section of the program code will be executed based on whether the condition evaluates to **TRUE** or **FALSE**.

#### Syntax:

```
if(condition) {
  // JavaScript code
}
```

If the **condition** evaluation returns **True** the JavaScript code is *executed* if the evaluation returns **False** this section of JavaScript code will be *skipped*.

**Example:**

```
var day = "Saturday"
if (day == "Saturday") {
    document.writeln("It's the weekend");
    alert("It's the weekend");
}
```

**Immediate if (Conditional expression)**

A conditional expression can evaluate to either **True** or **False** based on the **evaluation** of the condition. The structure of a conditional expression is:

**Syntax:**

```
(condition) ? value1 : value2
```

where, **condition** is an expression that can be evaluated to a boolean value. Based on the result, the whole expression evaluates to either **value1** (true condition) or **value2** (false condition).

**Example:**

```
var day = "Saturday"
(day == "Saturday") ? "Weekend!" : "Not Saturday"
```

This expression will evaluate to:

"Weekend!"	where <b>day</b> holds Saturday!	(Condition TRUE)
"Not Saturday!"	if <b>day</b> holds any other string	(Condition FALSE)

**SUPER CONTROLLED - ENDLESS LOOPS**

Looping refers to the ability of a block of code to repeat itself. This repetition can be for a predefined number of times or it can go until certain conditions are met. For instance, a block of code needs to be executed till the value of a variable becomes 20 (Conditional Looping), or a block of code needs to be repeated 7 times.

For this purpose, JavaScript offers 2 types of loop structures:

- **for Loops** - These loops iterate a specific number of times as specified.
- **while Loops** - These are Conditional Loops, which continue until a condition is met.

**For Loop**

The **for** loop is the most basic type of loop and resembles a for loop in most other programming languages, including ANSI 'C'.

**Syntax:**

```
for(expression1; condition; expression2) {
    // JavaScript commands
}
```

where, **expression1** sets up a counter variable and assigns the initial value. The declaration of the counter variable can also be done here itself, **condition** specifies the final value for the loop to fire (i.e. the loop fires till **condition** evaluates to true), **expression2** specifies how the initial value in **expression1** is **incremented**.

**Example:**

The following block prints numbers from 10 to 1 on the VDU screen (Reverse Order).

```
for (var num = 10; num >=1; num--) {  
    document.writeln(num);  
}
```

**While Loop**

The **while** loop provides a similar functionality. The basic structure of a **while** loop is:

**Syntax:**

```
while (condition) {  
    // JavaScript commands  
}
```

Where, the *condition* is a valid JavaScript expression that evaluates to a Boolean value. The JavaScript commands execute as long as the condition is **true**.

**Example:**

The following block prints numbers from 1 to 10 on the screen.

```
var num = 1;  
while (num <= 10) {  
    document.writeln(num);  
    num++;  
}
```

**FUNCTIONS IN JAVASCRIPT**

Functions are blocks of JavaScript code that perform a specific task and **often** return a value. A JavaScript function may take zero or more parameters. Parameters are a standard technique via which control data can be passed to a function. Control data passed to a function, offers a means of controlling what a function returns.

**Built-in Functions**

JavaScript provides several built-in functions that can be used to perform explicit type conversions. Some of them are **eval()**, **parseInt()** and **parseFloat()**.

The **eval()** function can be used to convert a string expression to a numeric value.

**Example:**

The following results in the value 105 being assigned to the variable `grand_Total`.

```
var grand_Total = eval("10 * 10 + 5");
```

**Note**

---

Even if the string value passed as a parameter to **eval()** does represent a numeric value the use of **eval()** results in an error being generated.

---

The **parseInt()** function is used to convert a string value to an integer. The **parseInt()** function returns the first integer contained in a string or 0 if the string does not begin with an integer.

**Example:**

The following results in the integer 123 being assigned to the variable string2Num.

```
var string2Num = parseInt("123xyz");
```

The following results in "NaN" (Not a Number) being assigned to the variable string2Num.

```
var string2Num = parseInt("xyz");
```

The **parseFloat()** function returns the first floating point number contained in a string or 0 if the string does not begin with a valid floating point number.

**Example:**

The following results in the float 1.2 being assigned to the variable string2Num.

```
var string2Num = parseFloat("1.2xyz");
```

The following results in "NaN" (Not a Number) being assigned to the variable string2Num.

```
string2Num = parseInt("xyz");
```

## USER DEFINED FUNCTIONS

Functions offer the ability to group together JavaScript program code that performs a specific task into a single unit that can be used repeatedly whenever required in a JavaScript program.

A user defined function first needs to be declared and coded. Once this is done the function can be invoked by calling it using the name given to the function when it was declared.

Functions can accept information in the form of arguments and can return results.

Appropriate syntax needs to be followed for declaring functions, invoking them, passing them values and accepting their return values.

### Declaring Functions

Functions are declared and created using the **function** keyword. A function can comprise of the following:

- A name for the function
- A list of parameters (arguments) that will accept values passed to the function when called
- A block of JavaScript code that defines what the function does

**Syntax:**

```
function function_name(parameter1, parameter2, ... ) {  
    // Block of JavaScript code  
}
```

A **function\_name** is case sensitive, can include underscores (`_`), and has to start with a letter. The list of **parameters** passed to the function appears in parentheses and commas separate members of the list.

### Note



Defining a function does not execute the JavaScript code that makes up the function.

### Place Of Declaration

Functions can be declared anywhere within an HTML file. Preferably, functions are created within the `<HEAD>...</HEAD>` tags of the HTML file. This ensures that all functions will be *parsed* before they are invoked or called. If the function is called before it is declared and parsed, it will lead to an error condition, as the function has not been evaluated and the 'Browser' does not know that it exists.

## Note



The term **parsed** refers to the process by which the JavaScript interpreter evaluates each line of script code and converts it into a pseudo-compiled bytecode, before attempting to execute it. At this time, syntax errors and other programming mistakes that would prevent the script from running are trapped and reported.

## Passing Parameters

Values can be passed to function parameters when a 'parameterized' function is called. Values are passed to the function by listing them in the parentheses following the function name. Multiple values can be passed, separated by commas provided that the function has been coded to accept multiple parameters.

Both JavaScript built-in functions and user-defined functions can accept parameters, process them and return values. During declaration, a function needs to be informed about the number of values that will be passed.

### Example:

#### Function Declaration:

```
function printName(user) {
    document.write("<HR>Your Name is <B><I>");
    document.write(user);
    document.write("</B></I><HR>");
}
```

where, **printName** is a function, which has a parameter called **user**. The parameter **user** can be passed a value at the time of invoking the function. Within the function, reference to **user** will then refer to the **value** passed to the function.

#### Function Call:

A static value passed:

```
printName("Bob");
```

(will cause the string "Bob" to be assigned to the parameter **user**.)

A Variable Passed:

```
var user = "John";
printName(user);
```

(will cause the value "John" to be assigned to the parameter **user**.)

## Note



- Both variables and literals can be passed as arguments when calling a function.
- If a variable is passed to the function, changing the value of the parameter within the function does not change the value of the variable passed to the function.
- Parameters exist only for the life of the function.

### Example 3:

Once an HTML page completes loading it greets a user with a message 'Welcome'. When a user leaves this page, the 'Good-bye' alert dialog box is displayed.

```
<HTML>
```

```
<HEAD>
```

```
<TITLE>Creating and Using User Defined Functions</TITLE>
```



```

<SCRIPT Language="JavaScript">
  var name = "";
  function hello() {
    name = prompt('Enter Your Name:', 'Name');
    alert('Greetings ' + name + ', Welcome to my page!');
  }
  function goodbye() {
    alert('Goodbye ' + name + ', Sorry to see you go!');
  }
</SCRIPT>
</HEAD>
<BODY onLoad="hello();" onUnload="goodbye();">
  <IMG SRC="images/Pinkwhit.gif">
</BODY>
</HTML>

```

## Variable Scope

The parameters of a function are local to the function. They come into existence when the function is called and cease to exist when the function ends. For instance, in the example `printName()`, `user` exists only within the function `printName()` - it cannot be referred to or manipulated outside the function.

Also, any variable declared using `var` variable-name *within* the function would have a scope limited to the function.

If a variable is declared outside the body of the function, it is available to all statements within the JavaScript.

If a *local variable* is declared within a function has the same name as an existing *global variable*, then within the function code, that variable name will refer to the *local variable* and **not** the global variable. It is as though the global variable does not exist with respect to the JavaScript code within the function.

## Return Values

As with some JavaScript built-in functions, user defined functions can return values. Such values can be returned using the **return** statement. The return statement can be used to return any valid expression that evaluates to a single value.

### Example:

```

function cube(number) {
  var cube = number * number * number;
  return cube;
}

```

where, `cube` is a function, which accepts a parameter `number`, calculates its cube, assigns this calculation to a variable `cube` and returns the value of cube.

This function can also be written as follows:

```

function cube(number) {
  return number * number * number;
}

```

## Recursive Functions

Recursion refers to a situation, wherein *functions call themselves*. In other words, there is a call to a specific function from within the same function. Such functions are known as Recursive Functions.

### Example:

The following JavaScript function is an example of a Recursive Function that calculates the factorial of a number. (A factorial is a mathematical function. For example, factorial 5 is equal to  $5*4*3*2*1$ )

```
function factorial(number) {  
    if(number>1) {  
        return number * factorial(number-1);  
    }  
    else {  
        return number;  
    }  
}
```

This function receives a number as an argument and relies on the fact that the factorial of a number is the number multiplied by the factorial of one less than the number.

The function applies the formula and returns the number multiplied by the factorial of one less than the number.

### Note



---

Recursive functions are powerful, but can lead to infinite recursions. Infinite recursions occur when the function is designed in such a way as to call itself without being able to stop.

---

It is important to note that the function **factorial()** prevents infinite recursions because the **if-else** construct ensures that eventually the function will stop calling itself once the number passed to the function is equal to one. Additionally, if the function is *initially* called with a value *less than two*, no recursion will take place at all.

## PLACING TEXT IN A BROWSER

Using JavaScript a string can be written to the browser from within an HTML file. The **document** object in JavaScript has a **method** for placing text in a browser. This method is called **write()**. Methods are called by combining the object name with the method name:

Object-name.Method-Name

The **write()** method accepts a string value passed to it within its parentheses. The string value can then be written to the browser. The **write()** method accepts this string and places it in the current browser window. For example:

```
document.write("Test");
```

The string "Test" will be placed in the browser.

### Example 4:

The following example illustrates how JavaScript code places text in the browser window using **document.write()**.

```
<HTML>
  <HEAD><TITLE>Outputting Text </TITLE></HEAD>
  <BODY><CENTER><BR><BR>
    <IMG Height=100 Src="Images/sctfamil.gif" Width=100>Silicon Chip Technologies.<BR>
    <SCRIPT Language = "Javascript">
      document.write("<BR><BR>");
      document.write('<IMG Height=100 Src="Images/sctfamil.gif" Width=100>');
      document.write("<B>Silicon Chip Technologies.</B> <BR>");
    </SCRIPT>
  </CENTER></BODY>
</HTML>
```

**Output For Example 4:**  
(Refer to diagram 8.1)

### DIALOG BOXES

JavaScript provides the ability to pickup user input or display small amounts of text to the user by using dialog boxes. These dialog boxes appear as separate windows and their content depends on the information provided by the user. This content is independent of the text in the HTML page containing the JavaScript script and does not affect the content of the page in any way.



Diagram 8.1

There are three types of dialog boxes provided by JavaScript:

#### The Alert Dialog Box

The simplest way to direct small amounts of textual output to a browser's window is to use an alert dialog box. The JavaScript `alert()` method takes a string as an argument and displays an alert dialog box in the browser window when invoked by appropriate JavaScript.

The alert dialog box displays the string passed to the `alert()` method, as well as an  button. The JavaScript and the HTML program, in which this code snippet is held, will not continue processing until the  button is clicked.

The alert dialog box can be used to display a cautionary message or display some information. For instance:

- A message is displayed to the user when incorrect information is keyed in a form
- An invalid result is the output of a calculation
- A warning that a service is not available on a given date/time

#### Syntax:

```
alert("<Message>");
```

#### Example:

```
alert("Click OK to continue");
```

#### Example 5:

The following example shows an alert dialog box, welcoming a user. As soon as the  button is clicked, an image is displayed in the browser. This illustrates that all background processing **stops** until an `alert` has been responded to.

```
<HTML>
  <HEAD><TITLE>Example</TITLE></HEAD>
  <BODY><SCRIPT Language="Javascript">
    alert("Welcome To My Web Site!");
    document.write('<IMG Src="Images/welcome.gif">');
  </SCRIPT></BODY>
</HTML>
```

### Output For Example 5: (Refer to diagram 8.2)

### The Prompt Dialog Box

As seen, the alert dialog box simply displays information in a browser and does not allow any interaction. The addition of the  button provides some very minimal control over form events *i.e.* program execution halts completely until some user action takes place (*clicking on the  button*).

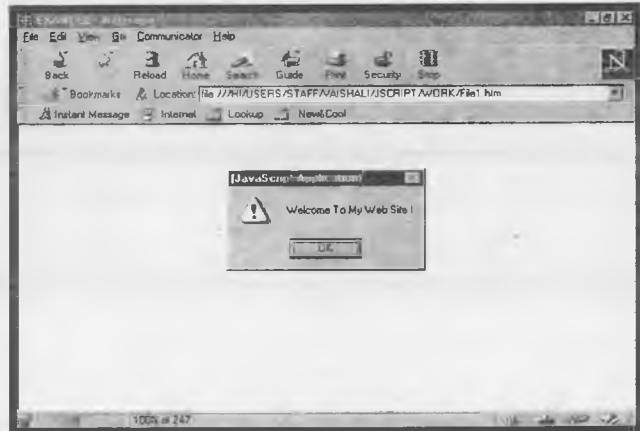


Diagram 8.2

An alert dialog box, cannot be used to customize any web page output based on user input, which is what user interaction requires.

JavaScript provides a prompt dialog box for this. The **prompt()** method instantiates the prompt dialog box which displays a specified message. In addition, the prompt dialog box also provides a single data entry field, which accepts user input. Thus, a prompt dialog box:

- Displays a predefined Message
- Displays a textbox and accepts user input
  - Can pass what the user keyed into the textbox back to the JavaScript
- Displays the  and the  buttons

The prompt dialog box also causes program execution to halt until user action takes place. This could be the  button being clicked, or the  button being clicked, which causes the following action to take place.

- Clicking on the  button causes the text typed inside the textbox to be passed to the program environment (*i.e.* *JavaScript*)
- Clicking on the  button causes a NULL value to be passed to the environment.

When the **prompt()** method is used to instantiate and use a dialog box the method requires two blocks of information:

- A message to be displayed as a prompt to the user
- Any message to be displayed in the textbox (*this is optional*)

#### Syntax:

```
prompt("<Message>", "<Default value>");
```

#### Example:

```
prompt("Enter your favorite color:", "Blue");
```

The value that the user keys into the textbox on the prompt dialog box is accepted and can be stored in a variable.

**Example 6:**

The following example shows a welcoming image on the screen. Asks the user for a name. Then displays the name keyed into the prompt dialog box along with a Greeting message.

```
<HTML>
  <HEAD><TITLE>Example 2.6 </TITLE></HEAD>
  <BODY>
    <SCRIPT LANGUAGE="JavaScript">
      document.write('<IMG Src="Images/welcome.gif">');
      document.write("<HI>Greetings, ");
      document.write(prompt("Enter Your Name: ", "Name"));
      document.write("Welcome to My HomePage!</HI>");
    </SCRIPT>
  </BODY>
</HTML>
```

**Output For Example 6:**

(Refer to diagram 8.3)

**The Confirm Dialog Box**

JavaScript provides a third type of a dialog box, called the confirm dialog box. As the name suggests, this dialog box serves as a technique for confirming user action. The confirm dialog box displays the following information:

- A pre-defined message
- and  button

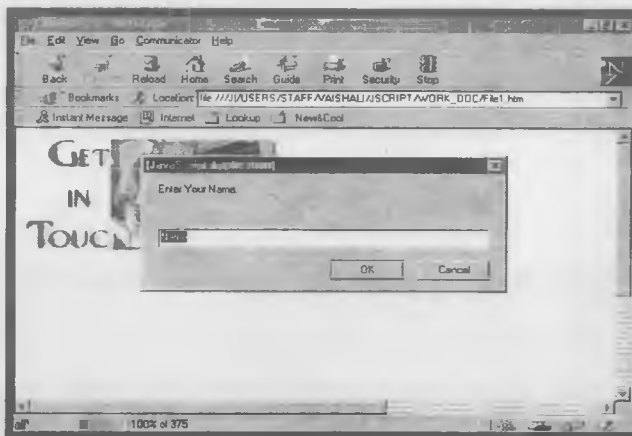


Diagram 8.3

The confirm dialog box, causes program execution to halt until user action takes place. User action can be either the  button being clicked, or the  button being clicked, which causes the following action to take place.

- Clicking on the  button causes TRUE to be passed to the program which called the confirm dialog box.
- Clicking on the  button causes FALSE to be passed to the program which called the confirm dialog box.

Display of a confirm dialog box thus requires only one block of information:  
A pre-defined message to be displayed

**Syntax:**

```
confirm("<Message>");
```

**Example:**

```
confirm("Are you sure you want to exit out of the system");
```

**Example 7:**

The following JavaScript example asks a question and accepts an answer. The user is given three chances. The second and third chance to provide an answer can be accepted or rejected, if accepted the program prompts for an answer again.

```

<HTML>
<HEAD>
<TITLE> Confirm Method </TITLE>
<SCRIPT LANGUAGE="JavaScript">
  var question = "What is 10+10 ?";
  var answer = 20;
  var correct = '<IMG Src="images/man2.gif">';
  var incorrect = '<IMG Src="images/man1.gif">';
  var Response = prompt(question,"0");
  for(count = 1; count < 3; count++) {
    if(Response != answer) {
      if( confirm("Wrong, Press OK For Another Chance")) {
        Response = prompt(question,"0");
      }
    }
    else {
      alert("Better Luck Next Time");
      count = 3;
    }
  }
  else {
    alert("Great!! Your Are Right");
    count = 3;
  }
}
var output = (Response == answer) ? correct : incorrect;
document.write("<BR>");
document.write(output);
</SCRIPT>
</HEAD>
<BODY></BODY>
</HTML>

```

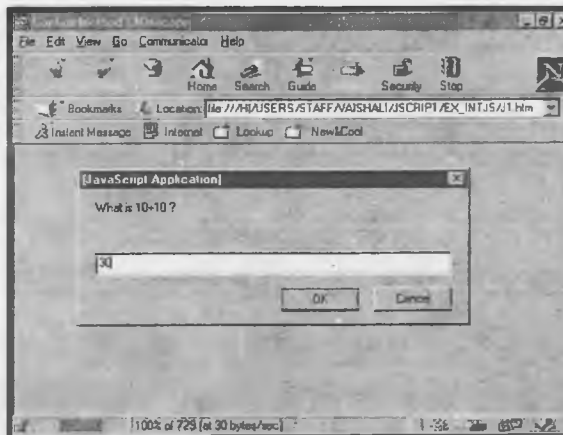
**Output:**

Diagram 8.4



Diagram 8.5

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. Capturing user requests is traditionally done via a \_\_\_\_\_.
2. JavaScript is a scripting language created by \_\_\_\_\_.
3. JavaScript is embedded between the \_\_\_\_\_ HTML tags.
4. A user request form can be created with the \_\_\_\_\_ HTML tags.
5. \_\_\_\_\_ are used to store values that can be used in other parts of a program.
6. The \_\_\_\_\_ is reserved for machine generated code and should not be used in scripts.
7. In JavaScript, variable is \_\_\_\_\_ based on the \_\_\_\_\_ value that is assigned to it.
8. A \_\_\_\_\_ is a sequence of zero or more characters that are enclosed by double (") or single (') quotes.
9. \_\_\_\_\_ are block of JavaScript code that perform a specific task and return a value.
10. \_\_\_\_\_ are JavaScript objects that are capable of storing a sequence of values.
11. A \_\_\_\_\_ array is an array that has been created with each of its elements being assigned a specific value.
12. \_\_\_\_\_ are named collections of data that have properties and may be accessed via methods.
13. The \_\_\_\_\_ operator calculates the remainder by dividing two integers.

### TRUE OR FALSE

14. JavaScript is not an interpreted language.
15. A JavaScript program developed on a Unix machine will work perfectly well on a Windows machine.
16. The <BODY></BODY>HTML tags make an ideal place to create JavaScript variables and constants and so on.
17. JavaScript does not allow the data type of the variable to be declared when a variable is created.
18. If the quote character is to be included in the string, the quote character must be preceded by the frontslash (/) escape character.
19. eval(), parseInt() and parseFloat() are three functions provided by JavaScript to perform explicit type conversions.
20. *Methods* are used to read or modify the data contained in an object.
21. The combination of an operator and its operands is referred to as an *expression*.
22. If a variable is declared outside the body of the function, it is available throughout a script inside all functions and elsewhere in the program.

## HANDS ON EXERCISES

1. Write a JavaScript code block using arrays and generate the current date in words, this should include the day, the month and the year. The output should be as follows, **Saturday, January 01, 2000**.
2. Write a JavaScript code block, which checks the contents entered in a form's Text element. If the text entered is in the lower case, convert to upper case. Make use of function *toUpperCase()*.
3. Write a JavaScript code block, which validates a username and password against hard coded values.

If either the name or password field is not entered display an error message showing:  
"You forgot one of the required fields. Please try again."

In case, the fields entered do not match the hard coded values, display an error message showing:  
"Please enter a valid username and password"

If the fields entered match, display the following message: "Welcome (username)"



## 9. THE JAVASCRIPT DOCUMENT OBJECT MODEL

### INTRODUCTION

An HTML page is rendered (*painted*) in a browser. The browser assembles all the elements (*objects*) contained in the HTML page, downloaded from the web server, in its memory. Once done the browser then renders (*paints*) these objects in the browser window. Once the HTML page is rendered (*painted*) in the browser window, the browser can no longer recognize individual HTML elements (*objects*).

To create an interactive web page it is imperative that the browser continues to recognize individual HTML objects even after they are rendered in the browser window. This allows the browser to access the properties of these objects using the built-in methods of the object. Once the properties of an object are accessible then the functionality of the object can be controlled at will.

JavaScript enabled browsers are capable of recognizing individual objects in an HTML page, after the page has been rendered in the browser, because the JavaScript enabled browser recognizes and uses the Document Object Model (i.e. *the DOM*).

Using the Document Object Model (DOM) JavaScript enabled browsers identify the collection of web page objects (*web page elements*) that have to be dealt with while rendering an HTML based, web page in the browser window.

The HTML objects (i.e. *the collection of web page elements*), which belong to the DOM, have a *descending* relationship with each other.

The topmost object in the DOM is the **Navigator** (i.e. *the browser*) itself. The next level in the DOM is the browser's **Window**. The next level in the DOM is the **Document** displayed in the browser's window.

Should the document displayed in the browser's window have an HTML **Form** coded in it, then the next level in the DOM is the **Form** itself.

The DOM hierarchy continues downward to encompass individual elements on a **FORM**, such as Text boxes, Labels, Radio buttons, Check boxes, Push buttons and so on, which belong to the form.

JavaScript's object hierarchy is mapped to the DOM, which in turn is mapped to the web page elements in the browser window. Hence, when a web page is rendered in a JavaScript enabled browser window, JavaScript is capable of uniquely identifying each element in the web page, because major elements of a web page are bound to the DOM.

The DOM that JavaScript recognizes is described in diagram 9.1.

JavaScript's DOM is referred to as an *instance hierarchy*.

**The Navigator** – i.e. Netscape Navigator, Internet Explorer, Opera, Mosaic and so on.

```
Window
|-> Document
    |-> Anchor
    |-> Link
    |-> Form
        |-> textbox
        |-> textarea
        |-> radiobutton
        |-> checkbox
        |-> select
        |-> button
```

Diagram 9.1: JavaScript's DOM.

## Instance

No HTML object is registered in the DOM by a JavaScript enabled browser unless they are assembled in memory prior being rendered in the browser window. What this means is, if a document does not have any **Anchor**s described in it the **Anchor** object will exist but it will be empty. If the document does not have any **Link**s described in it the **Link** object will exist but it will be empty.

## Hierarchy

All objects on a web page are not created equal. Each exists in a set relationship with other objects on the web page. From diagram 9.1 the navigator occupies the topmost slot in the DOM followed by the Window object and so on.

Below the Window is the **Document** object. Below the document object three other objects exist. They are the **Anchor**, **Link** and **Form** objects. Individual form elements are found under the **Form** object.

In addition to the DOM, other objects currently recognized by a JavaScript enabled browser are *Plug-ins*, *Applets* and *Images*. Hence using a JavaScript enabled browser and JavaScript most of the major web page objects are accessible.

However, every single element of a web page rendered in the browser window, is **not** part of the DOM.

For example, HTML tags such as `<HEAD> . . . </HEAD>` or `<BODY> . . . </BODY>` are not part of the DOM. Presentation styles, headings, body text, H1 to H6 and so on are **not** part of the DOM hence **not** recognized by JavaScript.

Diagram 9.1 shows web page objects that are part of the DOM.

JavaScript however, recognizes presentation styles, headings, body text, H1 to H6 and so on, when JavaScript assisted Style Sheets [JSSS] are in a web page. JSSS is usually between the `<HEAD> . . . </HEAD>` HTML tags in a web page.

## THE JAVASCRIPT ASSISTED STYLE SHEETS DOM [JSSS DOM]

JSSS use JavaScript syntax to control a document's presentation style. When a JSSS is embedded in an HTML page within the `<HEAD> . . . </HEAD>` tags, then the JavaScript DOM picks up a whole new set of objects, which **add** to the standard DOM objects already recognized by JavaScript. The additional objects brought into the DOM by JSSS are shown in diagram 9.2.

By extending the DOM recognized by JavaScript by embedding JSSS in a web page, developers of web pages can access every element of a web page whether this element appears on the page when it is rendered in a client browser or not.

By accessing appropriate properties of the **Navigator** object, (i.e. *the Browser*), the topmost object in the DOM, JavaScript can recognize the browser type (i.e. *Netscape Navigator, Internet Explorer, Opera, Mosaic and so on*) and subsequently dispatch all HTML pages to the browser from the web server, with a **style** based on this knowledge. This is where the power of JavaScript really becomes visible in providing finely tuned web page content to a client's browser.

### Objects added to the DOM by the use of JSSS are:

```

|-> Document
    |-> Tags
        |   |-> P
        |   |-> DIV
        |   |-> SPAN
        |   |-> H1 through H6
    |-> classes
        |   |-> Tag Names
    |-> IDS
  
```

Diagram 9.2: JSSS's DOM

Since JavaScript understands the DOM and can extend the DOM with the use of JSSS in a web page JavaScript understands **Objects**.

All **objects** have:

- Properties** that determine the functionality of the object
- Methods** that allow access to these properties
- Events** that allow JavaScript code snippets to be connected to the object by being mapped to appropriate JavaScript event handlers.

Hence when a pre-determined event occurs the code snippet will execute. This is the traditional Object, Event driven, Code execution model of any object based programming environment.

Using appropriate JavaScript code snippets, which reference the properties of an object via its built-in methods, developers of web pages can actually control the functionality of any HTML object in the DOM (or *extended DOM*) while the HTML program executes (i.e. *at run time*).

JavaScript can access the methods of all objects belonging to the DOM and JSSS DOM. Hence using JavaScript, truly interactive web pages can be created.

## Note



---

JavaScript is an **object-based** programming language it is **not** fully object oriented. It does not fully support basic object oriented programming capabilities such as classification, inheritance, encapsulation and information hiding.

---

JavaScript features are geared towards providing developers the capability to quickly generate scripts that will execute in the context of a web page within a JavaScript enabled browser **or** on a web server that understands JavaScript.

Although JavaScript does not provide all of the features of a full object oriented programming language, it does provide a suite of object-based features that are especially tailored to Browser or Server side scripting.

These features include the recognition of a number of predefined browser and server objects. JavaScript has the ability to control the behavior of these objects through their properties and methods.

In the following chapters the focus will be only on **browser** side JavaScripting. The browser in which JavaScript code snippets will always run correctly will be Netscape Communicator as JavaScript is a Netscape product. JavaScript is the natural language of Netscape Communicator.

## UNDERSTANDING OBJECTS IN HTML

HTML can be used to create a Graphical User Interface (GUI) in a web page. HTML is capable of accessing and using a number of objects that actually belong to the operating system (i.e. Windows). One such object is a *textbox*. A 'textbox' is used in an HTML form to accept user input.

The text box is an object, which belongs to the DOM. JavaScript recognizes a text box. JavaScript facilitates access to all the *methods* of a *textbox*. One of the methods of the textbox permits access to the contents of the text box. Hence, JavaScript can process the contents of any textbox in an HTML form.

### Properties Of HTML Objects

Just like real world objects, HTML objects have a number of **properties** that determine the behavior of that object. An object's properties can be referenced as:

**ObjectName.PropertyName**

For example, a *textbox* can have properties like *name*, *size* and so on.

## Methods Of HTML Objects

As seen, properties determine the state of an object. While building interactive web pages an object's properties need to be set dynamically, *i.e. when the object is being used*. Thus all object based programming environments **must** have facilities to *set* or *get* the value of object properties. This allows program code to control the state of the object at run time.

**Methods** of an object are used to *set* or *get* a value of an object's property. Thus determining how an object behaves. An object's methods can be referenced as:

**ObjectName.MethodName**

Using JavaScript it is possible to use an object's built-in methods and manipulate the object's properties at run time. This gives a great deal of creative control to web page developers when web pages have to be created on demand.

Once JavaScript code accepts client side input and / or reads a client's browser parameters, on demand web pages can be structured and sent to the browser from a web server.

## BROWSER OBJECTS

When any JavaScript enabled browser loads a web page, the browser automatically creates a number of JavaScript objects that map to the DOM (*or JSSS DOM*). It is the DOM, which provides JavaScript access to the HTML objects that are contained in the web page.

The JavaScript code snippets imbedded as part of the web page itself (*i.e. embedded within the <SCRIPT></SCRIPT>* tags of *filename.html*) makes use of these objects to interact with the HTML objects in the web page.

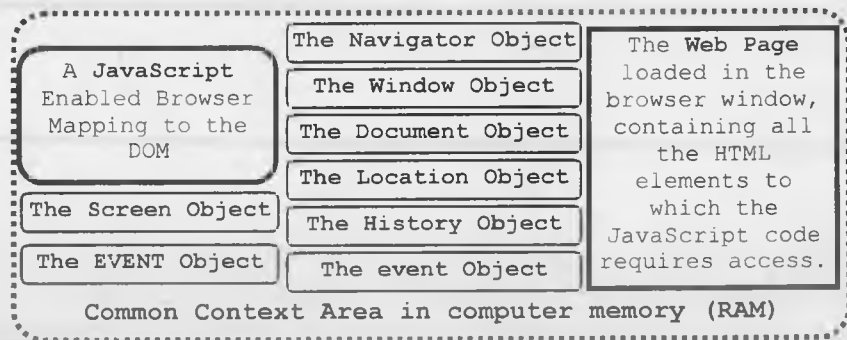


Diagram 9.3

The JavaScript objects created by [Netscape Communicator] are listed below:

Object Name	Its Use
navigator	To access information about the browser that is executing the current script.
window	To access a browser <i>window</i> or a <i>frame</i> within the window. The window object is <u>assumed to exist</u> and does not require the <b>window</b> prefix when referring to its properties and methods.
document	To access the document currently loaded into a window. The document object refers to an HTML document that provides content, that is, one that has HEAD and BODY tags.
location	To represent a URL. It can be used to create a URL object, access parts of a URL, or modify an existing URL.
history	To maintain a history of the URL's accessed within a window.
event	To access information about the occurrence of an event.
EVENT	The EVENT ( <i>capitalized</i> ) object provides constants that are <u>used to identify</u> events.
screen	To access information about the size and color depth of a client computer's screen in which the current browser is running.

Table 9.1

### How A JavaScript Enabled Browser Handles The Document Object

Any document (i.e. *the HTML page*) can contain various HTML objects such as:

- Images
- Image Maps
- Hyperlinks
- Frames
- Anchors
- Applets
- Multimedia objects such a audio files, streaming video files
- A Form with various form elements

The browser creates one array in memory per HTML object in the document, thus registering each of these HTML objects.

If these HTML objects are actually contained in the HTML page then these arrays will hold indexed elements, which will point to the context area in memory where the HTML object are. Otherwise the array will exist, but will be empty (i.e. *have no elements in it*).

If there are multiple, similar HTML objects in the document (i.e. *multiple images*) each array will have multiple (*indexed*) elements.

The array index value mapped to an HTML object will correspond to where the HTML object was described in the document. The first image in the document will have the array index as [0] (i.e. Images[0]), the next image in the document will have the array index of [1] and so on.

JavaScript provides access to the arrays and their elements. The values of *any/all* elements of each array can be identified, obtained. The values held in the elements of these arrays point to the context area in memory where the HTML objects actually are.

Once the context area in memory of an HTML object is known then using the **Methods** of the object, specific object **Properties** can be set using JavaScript. Thus the functionality of an HTML object can be controlled while the HTML page is running in the browser.

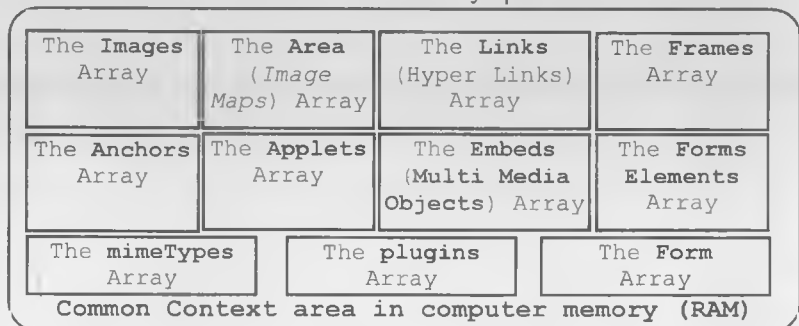


Diagram 9.4

The JavaScript arrays [created by Netscape Communicator] are listed below:

Image/Images Array	To access an image that is embedded in an HTML document. The <b>images array</b> is used to access all image objects in a document.
Link / Links Array	To access a text or image-based source anchor of a hypertext link. The <b>links array</b> is used to access all link objects within a document.
Area	To access an <b>area</b> within a client-side image map.
Frame / Frames Array	To access an HTML frame. The <b>frames array</b> is used to access all frames within a window.

Table 9.2

The JavaScript arrays [created by Netscape Communicator] are listed below: (Continued)

Anchor/Anchors array	To access the target of a hyperlink. The <b>anchors array</b> is used to access all anchor objects within a document.
Applet/Applets array	To Access a Java applet. The <b>applets array</b> being used to access all the applets in a document.
Embed / Embeds array	To access an embedded object. The <b>embeds array</b> provides access to all the embedded objects in a document.
MimeType / MimeType array	To access information about a particular MIME type supported by a browser. The <b>mimeTypes array</b> is an array of all the mimeType objects supported by a browser.
Plugin / Plugins array	To access information about a particular browser plug-in. The <b>plugins array</b> is an array of all plug-ins supported by a browser.
Form / Forms array	To access an HTML form. The <b>forms array</b> is used to access all forms within a document.

Table 9.2 (Continued)

The JavaScript **form elements** array [created by Netscape Communicator]:

elements	Access to all the form elements in the form.
text	To access a text field of a form.
textarea	To access a text area of a form.
radio	To access a set of radio buttons on the form or to access an individual radio button within the set.
checkbox	To access a checkbox on a form.
button	To access a form button that is not a reset or submit button.
submit	To access a submit button on a form.
reset	To access a reset button on a form.
select	To access a select list of a form.
option	The option object is used to access the elements of a select list.
password	To access a password field on a form.
hidden	To access a hidden object on a form.
fileupload	To access a file upload element of a form.

Table 9.3

## THE WEB PAGE HTML OBJECT HIERARCHY

This is an *instance hierarchy*. This means if a web page does not have a specific HTML object defined in it the array associated with that specific HTML object will exist, but will have no elements.

### Access To Elements Of A Web Page

Conceptually once a web page is rendered (*painted*) in a browser window it is completely static.

For any program code to be able to interact with the web page, each element of the web page would have to be held in memory, with a unique name. The unique name translates to a context area in memory where the web page element resides.

Hence, while a web page is being assembled in memory (RAM) prior being rendered (*made visible*) in the browser's window, a JavaScript enabled browser creates several arrays as described earlier. These arrays hold references to individual web page objects in their (*indexed*) elements.

Referencing an appropriate element in its associated array provides access to each element of a web page. Hence, using JavaScript web page elements can be updated or processed. Once processed, the web page can be re-rendered in the browser. When re-rendered in the browser changes made to the web page elements will be visible.

If updation or processing of any web page element is done, *based on client input*, the web page is then an *interactive* web page.

### How A Web Page Element Is Manipulated

HTML tags are used to create (*instantiate*) objects in a web page. For example, `<INPUT Type="TEXT">` will instantiate a text box on the web page when encountered in HTML code.

Each HTML object instantiated in a web page has **properties** and **methods** that allow access to the object's properties. Each HTML object has an **event** or several **events** bound to the object when the object was created. Thus an HTML object can recognize a specific event when it occurs.

Once the HTML object recognizes that an event has occurred this knowledge has to be passed to JavaScript so that JavaScript also recognizes that the 'event' occurred. To facilitate this JavaScript provides a number of named JavaScript 'Event Handlers'. The names of these event handlers are descriptively bound to an HTML object's event name.

#### Example:

A **Change** event is recognized by a text box when its contents change. JavaScript provides an event handler called **onChange** that is internally bound to the **Change** event of the text box. The **Change** event of the text box talks to the **onChange** event handler of JavaScript. The **onChange** event handler of JavaScript can then execute an appropriate JavaScript code snippet. For example:

```
<INPUT Type="TEXT" onChange="<myFunction">
```

Here the JavaScript function **myFunction** is bound to the JavaScript event handler **onChange**. The assignment operator (=) does this binding.

The JavaScript, **onChange** event handler, is bound to the HTML object **TEXT** by being passed as one of its attributes.

Hence, as soon as the **Change** event of the text box occurs the JavaScript event handler **onChange** is invoked.

Since the JavaScript function **myFunction** is bound to the JavaScript event handler **onChange**, as soon as the **onChange** event handler is invoked the JavaScript code in **myFunction** executes.

### HANDLING (WEB PAGE) EVENTS USING JAVASCRIPT

A web page event could be associated with the action of the mouse cursor on the web page. Such as:

- A mouse-click on an object in a web page
- The movement of the mouse cursor across a web page
- The mouse cursor hovering at a specific place on a web page and so on

These will be events recognized by the Window object of the DOM.

Other web page events could be the opening or closing of a Window, the loading of an image in a web page and so on.

JavaScript's approach to working with web page elements is a multi step process:

- Identify a web page object
- Choose an appropriate event associated with the object
- Have a standard method of connecting an object's event and JavaScript code snippets. JavaScript **event handlers** mapped to an object's **events** do this.

JavaScript has several named event handlers that are mapped to an HTML object's events. To work with JavaScript it is necessary to understand how to use JavaScript 'Event Handlers' correctly.

JavaScript, event handlers, can be divided into two types **Interactive** and **Non Interactive**.

An interactive event handler depends on user interaction with an HTML page. For example, the JavaScript **onMouseOver** event handler is an interactive event handler. This requires the user to move the mouse cursor over a web page.

A JavaScript, non-interactive, event handler, does not need user interaction to be invoked. For example the JavaScript 'onLoad' event handler is a non-interactive event handler as it automatically executes whenever a form is loaded into a web page.

Table 9.4 details JavaScript event handlers that descriptively bound to HTML object events. As long as the HTML object has an associated event, JavaScript provides an associated, named, event handler.

### Named JavaScript Event Handlers

JavaScript Event Handler	Will Be Called When
onAbort	The loading of an image is aborted as a result of user action
onBlur	A document, window, frame set, or form element loses current input focus.
onChange	A text field, text area, file-uploaded field or selection is modified and loses the current input focus.
onClick	A link, client-side image map area or document is clicked
onDbIcClick	A link, client-side image map area or document is double clicked
onDragDrop	A dragged object is dropped in a window or frame
onError	An error occurs during loading of an image, window or frame
onFocus	A document, window, frame set, or form element receives the current input focus
onKeyDown	The user presses a Key
onKeyPress	The user presses and releases a Key
onKeyUp	The user releases a Key
onLoad	An image, document or frame set is loaded
onMouseDown	The user presses a mouse button
onMouseMove	The user moves the mouse
onMouseOut	The mouse is moved out of a link or an area of a client side image map
onMouseOver	The mouse is moved over a link or an area of a client side image map
onMouseUp	The user releases a mouse button
onReset	The user resets a form by clicking on the form's reset button
onResize	The user resizes a window or frame
onSelect	Text is selected in a text field or a text area
onSubmit	The user presses a form's submit button
onUnload	The user exits a document or frame set

Table 9.4



The naming convention followed by Java Script makes it easy to identify a JavaScript event handler. The JavaScript, event handler's name, simply has the string 'on' added to the HTML object's event name (e.g. *onMouseOver*).

**Example:**

`<A href=http://www.sctindia.com onMouseOver="<JavaScript code snippet itself> or <A call to a JavaScript function">Text associated with the Link</A>`

**Tip**

Here, the **onMouseOver** JavaScript event handler is bound to the hyperlink `<A> . . . </A>` HTML tag. When the mouse cursor moves over the hyperlink its **MouseOver** event occurs. When **MouseOver** event occurs a call is made to the JavaScript event handler **onMouseOver**. When the JavaScript **onMouseOver** event handler is invoked a JavaScript code snippet can execute directly or a JavaScript function can be called.

**Note**

If a JavaScript code snippet is directly passed as a value to an HTML attribute **and** is enclosed in **double quotes** ("), then double quotes (") cannot be used within the enclosed JavaScript code. Replacing these double quotes with single quotes (') where necessary.

To use the JavaScript DOM and manipulate its objects properties will help in raising this basic comfort level to a confidence level required for coding in JavaScript.

To achieve this several examples which use the HTML `<FORM> . . . </FORM>` tags along with their attributes mapped to JavaScript code will be used.

**SELF REVIEW QUESTIONS****FILL IN THE BLANKS**

- Using the \_\_\_\_\_ JavaScript enabled browsers identify the collection of web page objects that have to be dealt with while rendering an HTML based web page in the browser window.
- \_\_\_\_\_ use JavaScript to control a document's presentation style
- \_\_\_\_\_ of an object are used to set or get a value of an object's property.
- JavaScript event handlers can be divided into two types \_\_\_\_\_ and \_\_\_\_\_.
- \_\_\_\_\_ object is used to access information about the browser that is executing the current script

**TRUE OR FALSE**

- If the document does not have any links described in it, the Link object does not exist.
- JavaScript 'onLoad' event handler is an interactive event handler as it automatically executes whenever a form is loaded into a web page.
- The browser creates one array in memory per HTML object in the document.

## HANDS ON EXERCISES

1. Create a Web page using two image files, which switch between one another as the mouse pointer moves over the images. Use the onMouseOver and onMouseOut event handlers. The output is as shown in the diagrams 9.5 and 9.6.



Diagram 9.5: Output for Hands on Exercise.

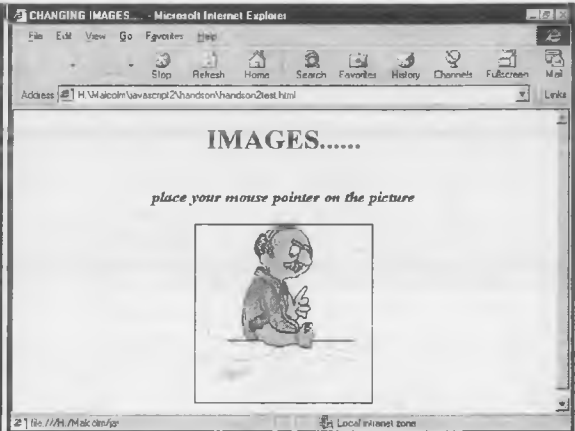


Diagram 9.6: Output for Hands on Exercise.

## 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 its 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

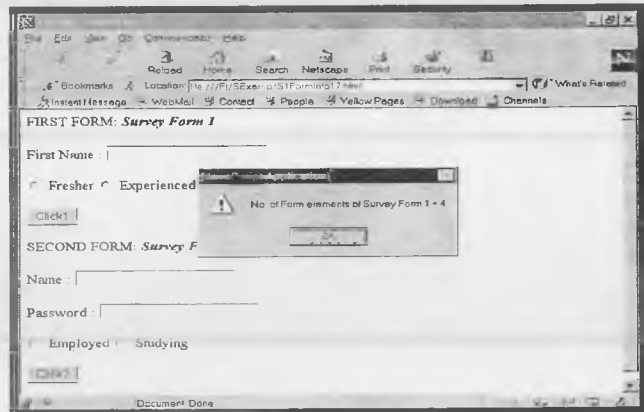
**Form2.Elements**

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:**

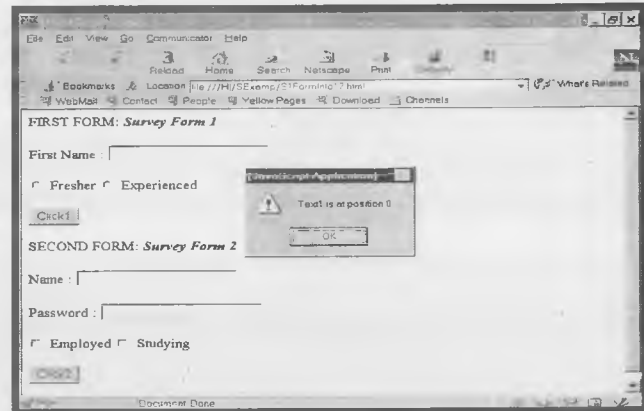
**Focus:** Count the number of elements in a Form's, elements array. Check the number returned against the number of form elements described between the **<FORM>...</FORM>** tags in the HTML page that is running in the Browser. Recognize that the number of elements in the elements array match the number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



**Diagram 10.2.1: First message for Exercise 1.**

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



**Diagram 10.2.2: Second message for Exercise 1.**

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.

## The Code Listing For Exercise 1:

```

<HTML>
  <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.

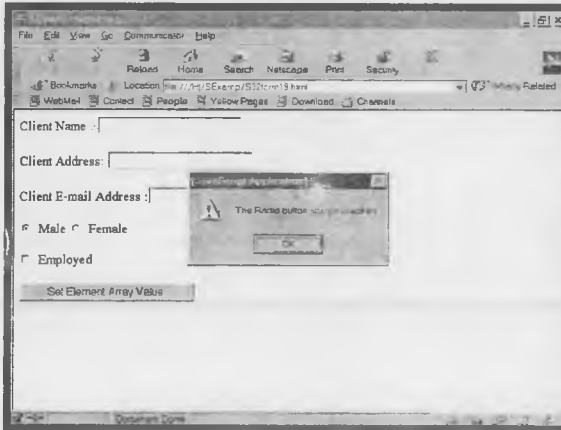


Diagram 10.3.1: First message for Exercise 2.

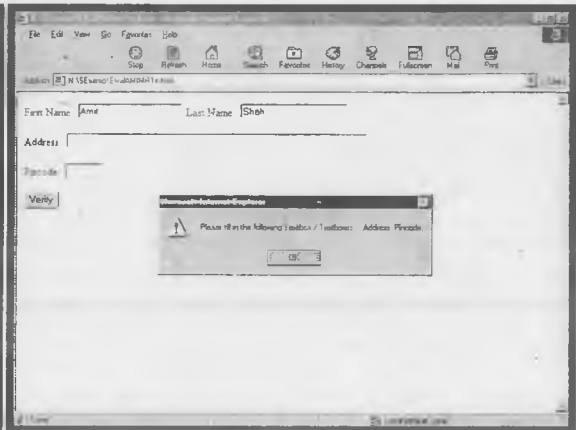


Diagram 10.3.2: Second message for Exercise 2.

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:

```
<HTML>
  <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>
```

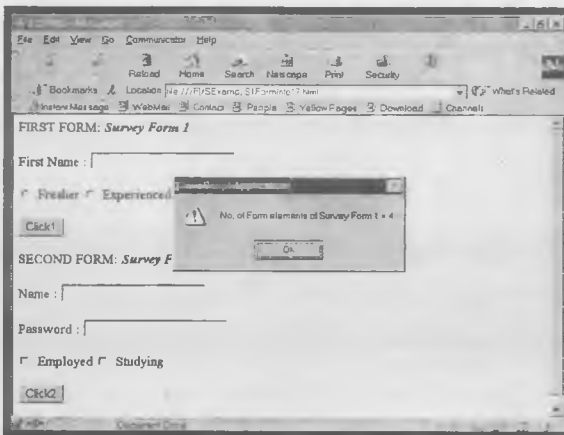
```
</HTML>
```

**Exercise 3:**

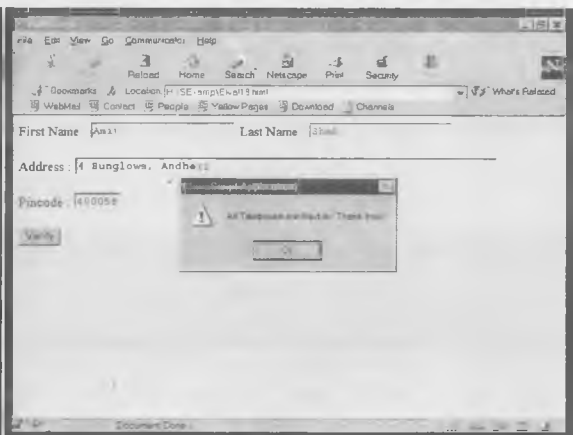
**Focus:** Create a HTML form that has a number of Textboxes. When the form runs in the Browser fill the Textboxes with data. Write JavaScript code that verifies that all Textboxes have been filled. If a Textbox has been left empty, pop up an Alert indicating which Textbox has been left empty. When the 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>
    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, ANSIC - (*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 ( <code>&lt;INPUT Type="Text"&gt;</code> )
Password	A password text field in which each keystroke appears as an asterisk(*) ( <code>&lt;INPUT Type="Password"&gt;</code> )
Button	A new element that provides a button other than a submit or reset button ( <code>&lt;INPUT Type="Button"&gt;</code> )
Checkbox	A checkbox ( <code>&lt;INPUT Type="Checkbox"&gt;</code> )
Radio	A radio button ( <code>&lt;INPUT Type="Radio"&gt;</code> )
Reset	A reset button ( <code>&lt;INPUT Type="Reset"&gt;</code> )
Submit	A submit button ( <code>&lt;INPUT Type="Submit"&gt;</code> )
Select	A selection list ( <code>&lt;SELECT&gt;&lt;OPTION&gt;option1&lt;/OPTION&gt;&lt;OPTION&gt;option2&lt;/OPTION&gt;&lt;/SELECT&gt;</code> )
TextArea	A multi line text entry field ( <code>&lt;TEXTAREA&gt;Default Text&lt;/TEXTAREA&gt;</code> )
Hidden	A field that may contain a value but is not displayed within a form ( <code>&lt;INPUT Type="Hidden"&gt;</code> )

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

**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		

Table 10.3: Methods Of Form Elements

Form Element Name	Method Name	Description
Button	onClick()	Fires when an object is clicked on.
Radio		
Checkbox		
Submit		
Reset		

**Table 10.3:** Methods Of Form Elements (Continued)

## 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...">
```

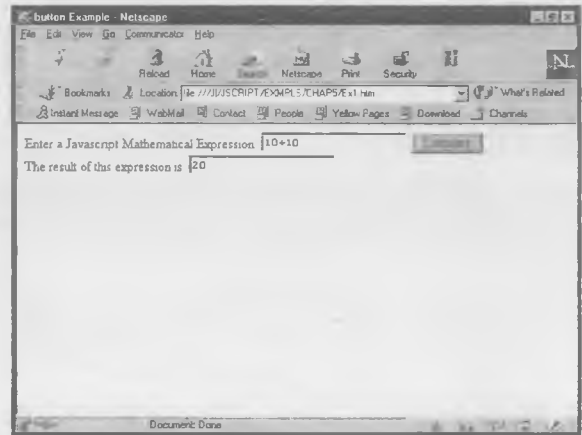
This places a **Button** on the HTML form named **btn\_check**. The button will display the text **Verify...** on its face as a label.

**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 Server 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.

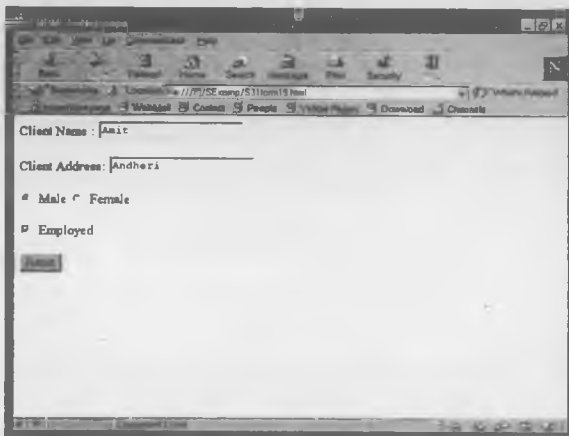
This will ensure that whenever a form is closed its fields will be emptied so that the next time the form is opened, its form elements will be empty, ready for reuse.

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

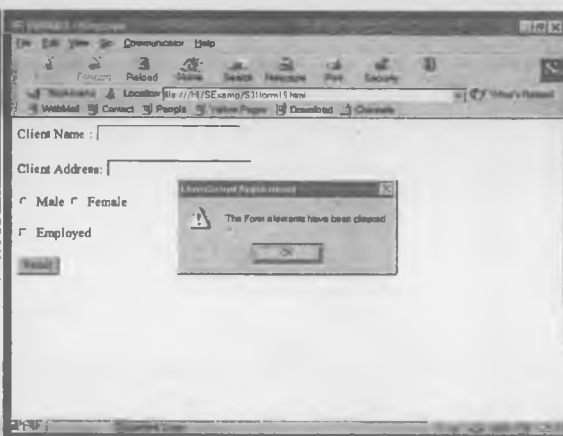
**Focus:** Create a form having Textboxes, Radio buttons, a Checkbox and a Reset button as shown in diagram 10.6.1. On clicking the Reset button, the entire form is reset (i.e. cleared). Refer to the HTML and JavaScript code described in Exercise 5.

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>FORMS</TITLE>
<!-- Using Reset Button //-->
  <SCRIPT>
    function func(fl) { 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 <b>not</b> checked	<b>Double</b> the numeric value entered
Checkbox is checked	The <b>square</b> 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 <b>not</b> checked	Half the numeric value entered
Checkbox is checked	The <b>square root</b> 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.

### 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>
```

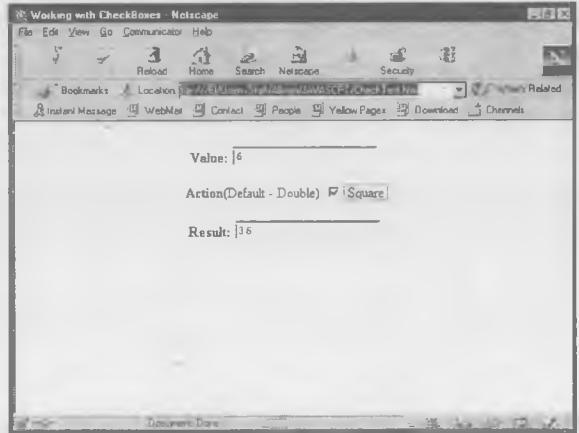


Diagram 10.7: Output for Exercise 6.

### 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>
```

### Example:

```
<INPUT Type="Radio" Name="Numbers" Value="1" 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 is 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.

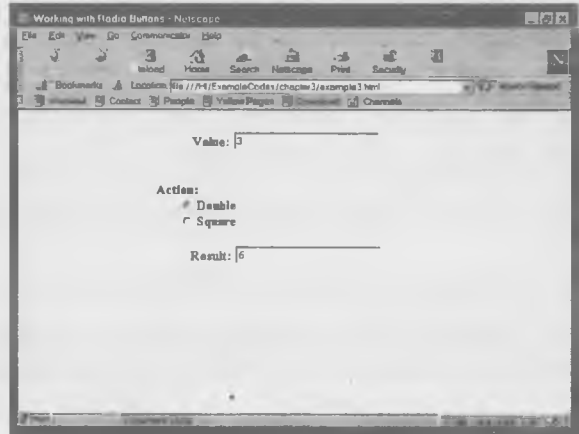


Diagram 10.8: Output for Exercise 7.

### 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>
```

### 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 Javascript 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>Hamburgers
  <OPTION>Hot Dogs
  <OPTION>Ice Cream Cones
  <OPTION>Salads
</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>Hamburgers
  <OPTION>Hot Dogs
  <OPTION>Ice Cream Cones
  <OPTION >Salads
</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.

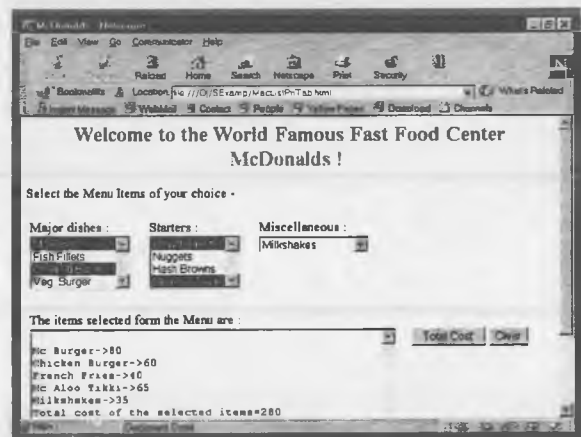
- The first Multiple choice list displays the Major dishes available.
- The second Multiple choice list displays the Starters available.
- The single choice list displays the Miscellaneous (Milkshakes, Soft drinks, Softy) available.

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.

**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;
```



**Diagram 10.9:** Output for Exercise 8.

```

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>
    <FONT Color="Blue">Welcome to the World Famous Fast Food Center </FONT>
    <FONT Color="Red"> McDonalds !</FONT>
  </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 Value="Fish Fillets->70"> Fish Fillets
      <OPTION Value="Chicken Burger->60"> Chicken Burger
      <OPTION Value="Veg. Burger->45"> Veg. Burger
    </SELECT><BR><BR>
  </TD><TD></TD><TD></TD><TD></TD>
  Starters :<BR>
  <SELECT Name="s2" MULTIPLE onBlur="pick(this.form)">
    <OPTION Value="French Fries->40"> French Fries
    <OPTION Value="Nuggets->50">Nuggets
    <OPTION Value="Hash Browns->55">Hash Browns
    <OPTION Value="Mc Aloo Tikki->65">Mc Aloo Tikki
  </SELECT><BR><BR>
</TD><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 Value="Soft drinks->20">Soft drinks
    <OPTION Value="Softy->25">Softy
  </SELECT><BR><BR>

```

```

</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	returns	ll
sample.toLowerCase()		hello		HELLO
sample.CharAt(3)		l		Hello
sample.italics()		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

Table 10.8

Method	Description
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 (Continued)

**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.

**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><FONT Color=red Size=+1><FORM Name=time_form><BR>
      <BR> Current Date & Time :')
      document.write('<INPUT Name=date Size=19 Value=""></FORM></FONT></CENTER>')
      begin(document.time_form);
    }
    display_clock();
  </SCRIPT></BODY>
</HTML>
```

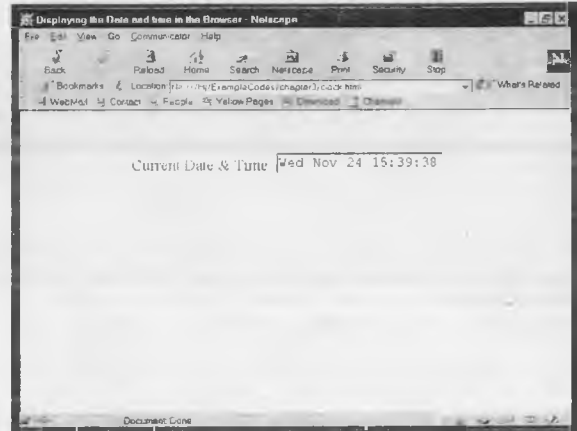


Diagram 10.10: Output for Exercise 9.

**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 an 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. """);
      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 BgColor=#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 BgColor="' + form.bg.value + '" Text="' + 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>")  
    write("<B>appVersion:<B>")  
    writeln(navigator.appVersion + "<BR>")  
    write("<B>appName:<B>")  
    writeln(navigator.appCodeName + "<BR>")  
    write("<B>platform:<B>")  
    writeln(navigator.platform + "<BR>")  
    write("<B>useragent:<B>")  
    writeln(navigator.userAgent + "<BR>")  
    write("<B>language: <B>")  
    writeln(navigator.language + "<BR>")  
    write("<B>Number of mimeTypes: <B>")  
    writeln(navigator.mimeTypes.length + "<BR>")  
    write("<B>Number of plugins:<B>")  
    writeln(navigator.plugins.length)  
  }  
}  
function displayExplorerProperties() {  
  with(document) {  
    write("<B>appName:<B>")  
    writeln(navigator.appName + "<BR>")  
    write("<B>appVersion:<B>")  
    writeln(navigator.appVersion + "<BR>")  
    write("<B>appMinorVersion:<B>")  
    writeln(navigator.appMinorVersion + "<BR>")  
    write("<B>appCodeName:<B>")  
    writeln(navigator.appCodeName + "<BR>")  
    write("<B>platform:<B>")  
    writeln(navigator.platform + "<BR>")  
    write("<B>cpuClass:<B>")  
    writeln(navigator.cpuClass + "<BR>")  
    write("<B>useragent:<B>")  
    writeln(navigator.userAgent + "<BR>")  
    write("<B>cookieEnabled:<B>")  
    writeln(navigator.cookieEnabled + "<BR>")  
    write("<B>browserLanguage:<B>")  
    writeln(navigator.browserLanguage + "<BR>")  
    write("<B>userLanguage:<B>")  
    writeln(navigator.userLanguage + "<BR>")  
    write("<B>systemLanguage:<B>")  
    writeln(navigator.systemLanguage + "<BR>")  
    write("<B>onLine: <B>")  
    writeln(navigator.onLine + "<BR>")  
    write("<B>Number of mimeTypes: <B>")  
    writeln(navigator.mimeTypes.length + "<BR>")  
  }  
}
```

```

write("<B>Number of plugins:<B>")
writeln(navigator.plugins.length)
write("<B>userProfile: <B>")
writeln(navigator.userProfile)
}
}
function displayBrowserProperties() {
  if(navigator.appName=="Netscape")
    displayNavigatorProperties()
  else
    if(navigator.appName=="Microsoft Internet Explorer")
      displayExplorerProperties()
}
displayBrowserProperties()
//-->
</SCRIPT></HEAD>
</HTML>

```

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

- HTML provides \_\_\_\_\_ and \_\_\_\_\_ tags with which an HTML form can be created to capture user input.
- The form tag has two properties namely \_\_\_\_\_ and \_\_\_\_\_.
- The method property is used to specify the method used to send data, which can be \_\_\_\_\_ or \_\_\_\_\_.
- The \_\_\_\_\_ attribute of the < FORM> tag points to the URL of a program on the web server that will process the form data.
- Various form elements can be specified as attributes of the \_\_\_\_\_ tag.
- The \_\_\_\_\_ button sends the current information held in each field of the form to the web server for further processing.
- To have a select object from which multiple choices can be made from a list the \_\_\_\_\_ attribute must be set in the select object.
- \_\_\_\_\_ is the only property of the string object.
- The \_\_\_\_\_ object provides methods and properties to move beyond simple arithmetic manipulations offered by arithmetic operators.
- Information can be stored locally in the browser which can be sent to the server whenever required by using \_\_\_\_\_.
- Cookie information is shared between the client browser and the server using fields in the \_\_\_\_\_ headers.
- When the user requests a page for the first time a cookie can be stored in the browser by a \_\_\_\_\_ entry in the header of the response from the server.



13. If a matching cookie is found among all the stored cookies, the browser sends a cookie field to the server in a \_\_\_\_\_.
14. The \_\_\_\_\_ attribute specifies that the cookie should be transmitted only over a secure link.

**TRUE OR FALSE**

15. As soon as the <FORM></FORM> tags are encountered by the browser, the browser creates an array called form1 in the absence of a Form name being specified in the HTML code.
16. The Post method sends data captured by the form elements to the web server, encoded as part of the URL that points to the web server.
17. Text fields accept a single line of text entry.
18. The Select object allows multiple choices from a list of choices that are offered.
19. Date object enables the creation of an object that contains information about a particular date.
20. In addition to the wide range of built-in objects, JavaScript permits the creation of user defined objects.
21. The name Cookie has a special significance with respect to object oriented programming language.
22. The set-cookie field includes the information to be stored in the cookie along with several optional piece of information.

**HANDS ON EXERCISES**

1. Create a web page, which accepts user information and user comments on the web site. Design the web page using form elements and check if all the Text fields have been entered with data else display an alert. Obtain an output as shown in the diagrams 10.11.1, 10.11.2 and 10.11.3.

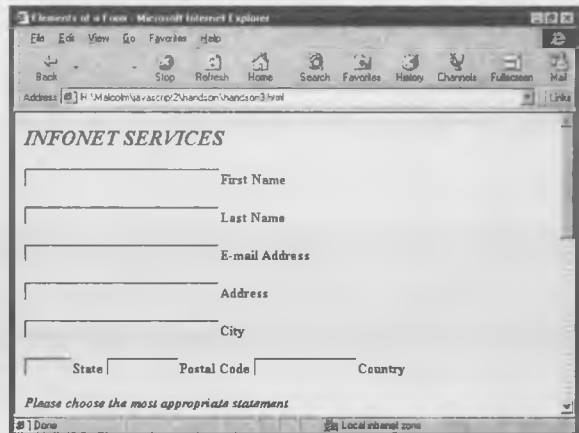


Diagram 10.11.1: Hands On Exercise output part 1.

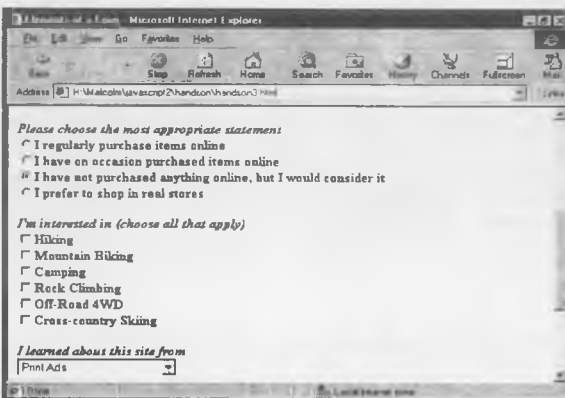


Diagram 10.11.2: Hands On Exercise output part 2.



Diagram 10.11.3: Hands On Exercise output part 3.

# 11. COOKIES

## WHAT ARE COOKIES

One of the challenges of writing applications for the World Wide Web has been inability of the web to maintain state. That is, after a user sends a request to the server and a web page is returned, the server forgets all about the user and the page that has been downloaded. If the user clicks on a link, the server doesn't have background information about what page the user is coming from, and more importantly, if the user returns to the page at a later date, there is no information available to the server about the user's previous actions on the page.

Maintaining state can be important to developing complex interactive applications. However, browsers address this problem with **cookies**, which is a method of storing information locally in the browser and sending it to the server whenever the appropriate pages are requested by the user.

The term *cookies* has no special significance. It is just a name in the same way Java is just a name for Sun's object-oriented programming language.

When a user requests a page, an HTTP request is sent to the server. The request includes a header that defines several pieces of information, including the page being requested.

The server returns an HTTP response that also includes a header. The header contains information about the document being returned, including its MIME type. These headers all contain one or more fields of information in a basic format.

### FieldName: Information

Cookie information is shared between the client browser and a server using fields in the HTTP headers. When the user requests a page for the first time, a cookie (or more than one cookie) can be stored in the browser by a **set-cookie** entry in the header of the response from the server. The set-cookie field includes the information to be stored in the cookie along with several optional pieces of information including an expiry date, path, and server information and if the cookie requires security

Then, when the user requests a page in the future, if a matching cookie is found among all the stored cookies, the browser sends a cookie field to the server in a request header. The header will contain the information stored in that cookie.

The set-cookie and cookie fields use a syntax to transfer significant information between client and server.

## SETTING A COOKIE

### Syntax:

**Set-cookie: NAME=value ; EXPIRES=date; PATH=path; DOMAIN=domain; SECURE**

The **NAME=value** is the only required piece of information that must be included in the set-cookie field. All other entries are optional.

Name	Description
NAME = value	Specifies the name of the cookie.
PATH = path	Specifies the path portion of the URLs for which the cookie is valid. If the URL matches both the PATH and the DOMAIN, then the cookie is sent to the server in the request header. (If left unset, the value of the PATH is the same as the document that set the cookie)

Table 11.1

Name	Description
EXPIRES = date	Specifies the expiry date of the cookie. After this date the cookie will no longer be stored by the client or sent to the server (DATE takes the form Weekday, DD-MON-YY HH:MM:SS GMT- dates are only stored in Greenwich Mean Time). By default the value of expires is set to the end of current Navigator session.
DOMAIN= domain	Specifies the domain portion of the URLs for which the cookie is valid. The default value for this attribute is the domain of the current document setting the cookie.
SECURE	Specifies that the cookie should only be transmitted over a secure link (i.e. to HTTP servers using the SSL protocol-known as HTTPS servers)

Table 11.1 (Continued)

**Example:**

This example illustrates the use of cookies. While the user traverses from page to page, the information entered by the user in any page is stored in cookies at the client side and made available to the user whenever the user returns to any page previously visited.

&lt;HTML&gt;

&lt;HEAD&gt;&lt;SCRIPT&gt;

```

function newcookie(cookieName, value) {
    document.cookie = cookieName + "=" + value; }
function getCookie(cookieName) {
    var cookiefound = false;
    var start = 0;
    var end = 0;
    var cstring = document.cookie;
    var clength = 0;
    while (clength <= cstring.length) {
        start = clength;
        end = start + cookieName.length;
        if (cstring.substring(start, end) == cookieName) {
            cookiefound = true;
            break;
        }
        clength++;
    }
    if (cookiefound) {
        start = end + 1;
        end = document.cookie.indexOf(";", start);
        if (end < start) {
            end = document.cookie.length;
            var contents = document.cookie.substring(start, end);
            return contents;
        }
        else {
            var contents = document.cookie.substring(start, end);
            return contents;
        }
    }
}
</SCRIPT></HEAD>

```

```

<BODY ><SCRIPT>
  var thisCookie = ((document.cookie !== "") && (document.cookie != null));
  var cfirst_name = (thisCookie) ? getCookie("first_name") : "";
  var cmiddle_name = (thisCookie) ? getCookie("middle_name") : "";
  var clast_name = (thisCookie) ? getCookie("last_name") : "";
  document.write('<FORM>');
  document.write('<TABLE Border="0" CellPadding="4" CellSpacing="4" Width = 100%>');
  document.write('<TR><TD ColSpan="2"><B> Personal Details (<FONT Color="Red"> *
    </Font> indicates Required)</B> </TD></TR>');
  document.write('<TR>');
  document.write('<TD Width="25%"> First Name <FONT Color="Red"> * </FONT></TD>');
  document.write('<TD Width="75%"><INPUT Name="first_name"
    onBlur="newcookie(this.name, this.value);" Size="20" Type="Text" Value="" +
    cfirst_name + "'></TD></TR>');
  document.write('<TR><TD Width="25%"> Middle Name </TD>');
  document.write('<TD Width="75%"><INPUT Name="middle_name"
    onBlur="newcookie(this.name, this.value);" Size="20" Type="Text" Value="" +
    cmiddle_name + "'></TD></TR>');
  document.write('<TR>');
  document.write('<TD Width="25%"> Last Name <FONT Color="Red"> * </FONT></TD>');
  document.write('<TD Width="75%"><INPUT Name="last_name"
    onBlur="newcookie(this.name, this.value);" Size="20" Type="Text" Value="" +
    clast_name + "'></TD>');
  document.write('</TR></TABLE><BR>');
  document.write('<CENTER>');
  document.write('<INPUT Type="Submit" Value="Done">');
  document.write('<INPUT Type="Reset" Value="Reset">');
  document.write('</CENTER>');
  document.write('</FORM>');
</SCRIPT></BODY>
</HTML>

```

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

- Information can be stored locally in the browser which can be sent to the server whenever required by using \_\_\_\_\_.
- Cookie information is shared between the client browser and the server using fields in the \_\_\_\_\_ headers.
- When the user requests a page for the first time a cookie can be stored in the browser by a \_\_\_\_\_ entry in the header of the response from the server.
- If a matching cookie is found among all the stored cookies, the browser sends a cookie field to the server in a \_\_\_\_\_.
- The \_\_\_\_\_ attribute specifies that the cookie should be transmitted only over a secure link.

### TRUE OR FALSE

- The name Cookie has a special significance with respect to object oriented programming language.
- The set-cookie field includes the information to be stored in the cookie along with several optional piece of information.

## B - PROJECTS IN JAVASCRIPT

### Project Specifications For The First Project In JavaScript – Guest Book

Since the learning of Java Script has now been completed, it is time to consolidate this learning by building a small guest book.

A Guest Book is used to store any comments that a visitor to the site may have regarding the site. The comments (if applicable) help improve the functionality of the site.

The structure of the guest book is given in the following pages along with html source code. This is a description of how the guest book will be used by the person signing-in. The java script code working with Html which provides for various client side data capture validations.

The guest book is a html document with:

- Simple visuals in the form of .gif or .jpeg files.
- Java Script embedded HTML code.

The files required to construct these pages is available on the accompanying CD-ROM for immediate use.

If required run the HTML files first in a web browser and get a look and feel of what the project could be like. Once this is done, code the web pages according to what you believe is appropriate.

For a visitor to leave comments about the site, or comments in general, the HTML form used in diagram 1.1 is used. This is the guest book page.

This form captures the following:

- Visitors Name (Not compulsory)
- Emailid (Mandatory)
- A visitors Comments (Mandatory)

Feel free to improvise and get a look and feel that satisfies you.

Each HTML file is really just a simple guideline to what the web page could look like. Java script is added to perform simple client side validations.

**For your guidance:**

- The source code is provided in the document file.

#### **Client Side Validations**

The Client side validations to be coded in Java script are as follows

- Emailid, General Comments, cannot be left empty
- The Emailid needs to be scanned for the presence of an '@' and '.' symbol.
- The length of the Name and Emailid fields cannot exceed 30 characters.

The screenshot shows a web form titled "OUR GUEST BOOK" under the heading "Silicon Chip Technologies". The form contains the following elements:  
- A header: "OUR GUEST BOOK" in large, bold, black letters.  
- A sub-header: "Please take a few moments to let us know you were here today."  
- A text input field: "Please Give Us Your Name" with a horizontal line below it.  
- Another text input field: "Please Give Us Your Email Address" with a horizontal line below it.  
- A large text area: "Bouquets Or Brickbats Are Welcome" with a scroll bar on the right.  
- A checkbox: "Can we contact you with information about our products or services." with radio buttons for "Yes" and "No, Thanks!".  
- Three buttons: "Submit", "Reset", and "Abort" in a row.  
- A footer: "Thank You For Stopping By Our Web Site".

**Diagram 1.1:** The User Interface For Guest book

&lt;HTML&gt;

&lt;HEAD&gt;&lt;TITLE&gt;SCT'S GUEST BOOK&lt;/TITLE&gt;

&lt;SCRIPT Language = "Javascript"&gt;

```
<!-- The function verify() checks whether appropriate information
is filled in all the elements. If any element is left empty, an
alert() box is displayed informing the user to fill in the empty
element The code also scans the Emailid for the presence of an
'@' and a '.' symbol. -->
```

```
function verify(form) {
    for (i=1; i<=2; i++) {
        if (document.forms[0].elements[i].value == "") {
            alert("Please fill in the " + document.forms[0].elements[i].name + " field");
            document.forms[0].elements[i].focus( );
            return (false);
        }
        if(document.forms[0].elements[1].value != "") {
            pass = document.forms[0].elements[1].value.indexOf('@',0);
            pass1 = document.forms[0].elements[1].value.indexOf(':',0);
            if((pass==-1) || (pass1==-1)) {
                alert("Not a valid Email address");
                document.forms[0].elements[1].focus();
                return (false);
            }
        }
    }
    return(true);
}
```

```
<!-- This function takes the user back to the 'Home Page' from the
current page -->
```

```
function abort(form) {
    history.back();
}
```

```
<!-- Sets the focus on the first element when the form is loaded -->
```

```
function set(form) {
    document.forms[0].elements[0].focus( );
}
```

```
<!-- The function checklen() checks that the length of name and
email addresses does not exceed 30 characters -->
```

```
function checklen(form) {
    for (i=0; i<=1; i++) {
        val=document.forms[0].elements[i].value;
        len=val.length;
        if (len > 30) {
            alert ("Value exceeds 30 characters");
            document.forms[0].elements[i].value="";
            document.forms[0].elements[i].focus();
        }
    }
}
```

&lt;/SCRIPT&gt;&lt;/HEAD&gt;

```

<BODY Background="images/grid1.gif" TEXT="green" onLoad="set(this.form)">
  <CENTER><IMG Src="images/Gestbk.gif" Alt="gestbk.gif"></CENTER>
  <P><P><CENTER>Please take a few moments to let us know you were here today.
  <P><P><FORM Action="" Method="POST" onSubmit="return verify(this.form)">
    <P><P>Please Give Us Your Name<BR>
    <INPUT Type="text" Name="name" Size="40" onBlur="checklen(this.form)"><BR>
    <P><P>Please Give Us Your Email Address<BR>
    <INPUT Type="text" Name="emailid" Size="40" onBlur="checklen(this.form)"><BR>
    <P><P>Bouquets Or Brickbats Are Welcome<BR>
    <TEXTAREA Name="request" Rows="8" Cols="65"></TEXTAREA>
    <P><P>Can we contact you with information about our products or services.<BR><BR>
    <INPUT TYPE="radio" NAME="moreinfo" VALUE="y">
    <FONT Color = "Blue">Yes</FONT><IMG Src="images/shim.gif" Width=20 Height=10>
    <INPUT TYPE="radio" NAME="moreinfo" VALUE="n" Checked="True">
    <FONT Color = "Red">No, Thanks!</FONT>
    <P><INPUT Type="Submit" Value="Submit"><INPUT Type="Reset" Value="Reset">
    <INPUT Type="Button" Value="Abort" onClick = "abort(this.form)">
  </FORM>
  <P><B>Thank You For Stopping By Our Web Site</B></CENTER>
</BODY>
</HTML>

```

## Project Specifications For The First Project In JavaScript – Pen Pals

Since the learning of Java Script has now been completed, it is time to consolidate this learning by building a page, which offers free hosting of pen pal information. Java Script will then be used to validate data captured by this page.

Using this service, visitors can post their Name, Emailid, Sex, Date of birth, Interests and Hobbies on the Web Site. Other visitors who find compatible information in the Pen pals list would email the registrant. This could become the start of a lasting friendship.

The structure of the pen pal page is given in the following pages along with the html source code. This is a description of how the Pen Pal page is to be used by the person visiting the page. The java code working within html provides for data capture validations.

Other visitors can then refer to these pages and email of individuals who's Age, Sex, Interests and other criteria are interesting. To post Pen Pal Information on a Web Site a simple form needs to be filled in with details as shown in diagram 2.1.

Diagram 2.1: The User Interface For Pen Pals

The Pen Pal page is a html document with:

- Simple visuals in the form of .gif or .jpeg files.
- Java script embedded HTML code.

The files required to construct these pages is available on the accompanying CD-ROM for immediate use.

If required run the HTML files first in a web browser and get a look and feel of what the project could be like. Once this is done, code the web pages according to what you believe is appropriate.

Pen Pal information is hosted for free on our website to visitors to the website.

For a visitor to leave his/her information on a site. The HTML form used in diagram 1 is used. This is the Pen Pal page.

The form captures the following:

- Visitors Name (Mandatory)
- Emailid (Mandatory)
- The visitors D O B in DD/MM/YYYY (Mandatory)
- Textbox to retrieve users hobbies and interests (Not Compulsory)

Feel free to improvise and get a look and feel that satisfies you.

Each HTML file is really just a simple guideline to what the web page could look like. Java script is added to perform simple client side validations.

#### For your guidance:

- The source code is provided in the document file.

#### Client Side Validations:

The Client Side validations to be coded in Java script are as follows:

- Name, Email id, Sex, DOB entries, cannot be left empty
- The Email id needs to be scanned for the presence of an '@' and '.' sign
- The length of the Name and Email id fields cannot exceed 30 characters
- Ensure that the date is keyed in as "DD/MM/YYYY"

<HTML>

```
<HEAD><TITLE>SCT's PEN PAL</TITLE>
```

```
<SCRIPT Language = "Javascript">
```

```
<!-- The function verify() checks whether appropriate intervention is
filled in all the form elements. If any element is left empty an
alert() box is displayed informing the user to fill in all the
empty elements. The code also searches the emailed for the
presence of an '@' and a '.' symbol -->
```

```
function verify() {
  for (i=0; i<=7; i++) {
    if (document.forms[0].elements[i].value == "") {
      alert("Please fill in the " + document.forms[0].elements[i].name + " field");
      document.forms[0].elements[i].focus();
      return (false);
    }
    if(document.forms[0].elements[1].value != "") {
      pass = document.forms[0].elements[1].value.indexOf('@',0);
      pass1 = document.forms[0].elements[1].value.indexOf('.',0);
      if((pass== -1) || (pass1== -1)) {
        alert("not a valid email address");
        document.forms[0].elements[1].focus();
      }
    }
  }
}
```



```

        return (false);
    }
}
return(true);
}
<!-- The function checkdate() checks whether the date is between
(1-31) whether the month is entered as a number and is between
(1-12) whether the year entered is within 1995 -->
function checkdate() {
    year=document.forms[0].elements[6].value;
    if (year>1995) {
        alert("Enter proper year (till 1995)");
        document.forms[0].elements[6].focus();
        return(false);
    }
    monval=document.forms[0].elements[5].value;
    dtval=document.forms[0].elements[4].value;
    if ((dtval > 31) || (dtval < 1)) {
        alert("Enter proper date");
        document.forms[0].elements[4].focus();
        return(false);
    }
    if (isNaN(monval) != true) {
        if ((monval > 12) || (monval < 1)) {
            alert("Enter proper month");
            document.forms[0].elements[5].focus();
            return(false);
        }
    }
    else {
        alert("Enter the month number");
        document.forms[0].elements[5].focus();
        return(false);
    }
    return (true);
}
}
<!-- The function checklen() checks whether the length of name and
email address does not exceed 30 characters -->
function checklen() {
    for (i=0; i<=1; i++) {
        val=document.forms[0].elements[i].value;
        len=val.length;
        if (len > 30) {
            alert ("Value exceeds 30 characters");
            document.forms[0].elements[i].value="";
            document.forms[0].elements[i].focus();
        }
    }
}
}

```

```
<!-- The function takes the user to the previous page -->
    function abort(form) {
        history.back();
    }
<!-- Sets the focus on the first field when the form is loaded -->
    function set(form) {
        document.forms[0].elements[0].focus();
    }
</SCRIPT></HEAD>
<BODY Background="images\Grid1.gif" onLoad="set(this.form)">
    <CENTER><IMG SRC="images\Penpals.gif"></CENTER>
    <P><P><CENTER>
    <P><FORM Action="" Method="POST" onSubmit="return verify()">
        <P>Please enter your name<BR>
        <INPUT Type="text" Name="name" Size="40" onBlur="checklen()" ><BR>
        <P>Please enter your E-Mail address<BR>
        <INPUT Type="Text" Name="emailid" Size="40" onBlur="checklen()"><BR>
        <P>Please indicate your Sex<BR>
        <INPUT Type="Radio" Name="sex" Value="m" checked>Male
        <IMG Src="images/Shim.gif" Height="1" Width="5">
        <INPUT Type="Radio" Name="sex" Value="f" >Female<BR>
        <P>Please enter your DOB as DD/MM/YYYY<BR>
        <INPUT Type="Text" Name="day" Size="2" >
        <IMG Src="images/Shim.gif" Height="1" Width="2">
        <INPUT Type="Text" Name="month" Size="2">
        <IMG Src="images/Shim.gif" Height="1" Width="2">
        <INPUT Type="Text" Name="year" Size="4" >
        <P><P>Tell us about your hobbies and interests in the 'Text-Box' below<BR>
        We'll keep your Penpal info posted for <B>One Year</B> from today<p>
        <TEXTAREA NAME="request" ROWS="8" COLS="65" onFocus="checkdate()">
        </TEXTAREA>
        <P><P>Can we contact you with information about our products or services.<BR>
        <INPUT Type="Radio" Name="moreinfo" Value="y">
        <FONT Color = "Blue">Yes</FONT><IMG Src="images\shim.gif" Width=20 Height=10 >
        <INPUT Type="Radio" Name="moreinfo" Value="n" Checked="True">
        <FONT Color = "Red">No, Thanks!</FONT>
        <P><CENTER><INPUT Type="Submit" Value="Submit" Name="Submit">
        <INPUT Type="Reset" Value="Reset" Name="Submit">
        <INPUT Type="Button" Value="Abort" Name="At" onClick = "abort(this.form)">
        </CENTER></FORM>
        <P><B>Thank You For Stopping By Our Web Site</B></CENTER>
    </BODY>
</HTML>
```

## Project Specifications For The First Project In JavaScript – Registration Form

Since the learning of Java Script has now been completed, it is time to consolidate this learning by building a Registration form page.

This Registration page is used to register visitors and provide access to a number of services provided by the Web Site. To register a simple form as show in diagram 3.1 is used.

The structure of the registration page is given in the following pages along with html source code. The Java script code working within html, which provides for various client side data capture validations.

This captures two different blocks of information. The first block of information is associated with the visitors Login ID and Password that will be used when access to the sites services will be required in future. The second block of information captured, will be the services that the visitor wishes to sign up for.

The Registration page is a html document with:

- Simple visuals in the form of .gif or .jpeg files.
- Java Script embedded HTML code.

The files required to construct these pages is available on the accompanying CD-ROM for immediate use.

If required run the HTML files first in a web browser and get a look and feel of what the project could be like. Once this is done, code the web pages according to what you believe is appropriate.

Feel free to improvise and get a look and feel that satisfies you.

Each HTML file is really just a simple guideline to what the web page could look like.

Java script is added to perform simple client side validations.

### For your guidance:

- The source code is provided in the document file.

### Client Side Validations

The Client side validations to be coded in Java script are as follows

- The Login name, Password, Confirm Password cannot be left empty.
- The Emailid needs to be scanned for the presence of an '@' and '.' symbol.
- The Password and the Confirm Password has to be the same.
- The length of the Name and Emailid fields cannot exceed 30 characters.

<HTML>

```
<HEAD><TITLE>SCT'S GOODIES SIGN UP FORM</TITLE>
<SCRIPT Language="JavaScript">
<!-- Declaration of Variables -->
    var valueofpass1="";
    var valueofpass2="";
    var whitespace = " \t\n\r";
```

Diagram 3.1: The User Interface For Registration Form

```
<!-- Function to check whether the value in a Text Field is Null -->
function isEmpty(s) {
    return ((s == null) || (s.length == 0))
}
<!-- Function to check whether the value in a Text Field is a
WhiteSpace -->
function isWhitespace (s) {
    var i;
    <!-- Is empty? -->
    if (isEmpty(s)) return true;
    <!-- Search through string's characters one by one until we find
a non-whitespace character. -->
    for (i=0; i<s.length; i++) {
        <-- Check that current character isn't whitespace.>
        var currchar = s.charAt(i);
        if (whitespace.indexOf(currchar) == -1)
            return false;
    }
    <-- All characters are whitespace. -->
    return true;
}
<!-- Function to ensure that the email address is in proper format.
-->
function isEmail (eadd) {
    if (isEmpty(eadd))
        <!-- Is s whitespace? -->
        if (isWhitespace(eadd)) return false;
        <!-- There must be >= 1 character before @, so we start
looking at character position 1 (i.e. second character)
-->
        var i = 1;
        var sLength = eadd.length;
        <!-- look for @ -->
        while ((i < sLength) && (s.charAt(i) != "@")) {
            i++;
        }
        if ((i >= sLength) || (s.charAt(i) != "@"))
            return false;
        else
            i += 2;
        <!-- look for period -->
        while ((i < sLength) && (s.charAt(i) != ".")) {
            i++;
        }
        <!-- There must be at least one character after the period-->
        if ((i >= sLength - 1) || (s.charAt(i) != "."))
            return false;
        else
            return true;
    }
}
```

```

<!-- Function to check whether the value in Password contains
  Alphabets and Characters-->
function isCharsInBag (string, bag) {
  var i;
  <!-- Search through string's characters one by one. If character
    is in bag, append to returnString. -->
  for (i=0; i < string.length; i++) {
    var charval = s.charAt(i);
    if (bag.indexOf(charval) == -1) return false;
  }
  return true;
}

<!-- Function to check whether the Password contains atleast one
  number -->
function isNumberInPass (string, bag) {
  var i, flag;
  flag=0;
  <!-- Search through string's characters one by one. If character
    is in bag, append to return String. -->
  for (i=0; i < string.length; i++) {
    var charval = s.charAt(i);
    if (bag.indexOf(charval) == -1) {
      continue;
    }
    else {
      flag=1;
      break;
    }
  }
  if(flag == 1) { return true; }
  else { return false; }
  return false;
}

<!-- Function to check the values entered in all the elements of the
  formCalled on the clicked event of the Submit button -->
function verify() {
  var flag =0;
  <!-- Check to see if any of the Text fields is left blank -->
  for (i=0; i<=3; i++) {
    if (document.forms[0].elements[i].value == "") {
      alert("Please fill in the " + document.forms[0].elements[i].Name + "field");
      document.forms[0].elements[i].focus();
      flag =1;
      break;
    }
  }
  if (flag == 1) {
    return( false);
  }
}

```

```
<!-- Begining the check for Email address, Password and Confirm
Password -->
if (flag == 0) {
    var email = document.forms[0].elements[1].value;
    <!-- Validate the email address -->
    if (!isEmail(email)) {
        alert("Please enter the Email address in the proper Format");
        document.forms[0].elements[1].focus( );
        return false;
    }
    var passwd = document.forms[0].elements[2].value;
    <-- Validate the Password -->
    if(!isCharsInBag(passwd,
        abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ
        Z0123456789")) {
        alert( "Password must only contain alphabets and number");
        return false;
    }
    <-- Check to ensure that the Password is not less than 5
    characters -->
    else if ( passwd.length < 5 ) {
        alert( "Password must be 5 or more characters." );
        return false;
    }
    <--Check to ensure that the Password is not greater than 8
    characters -->
    else if (passwd.length > 8 ) {
        alert( "Password must be 8 or more characters." );
        return false;
    }
    <-- Check to ensure that the Password contains atleast one
    number -->
    else if ( !isNumberInPass( passwd, "0123456789") ) {
        alert( "Password must have atleast one number." );
        return false;
    }
    <-- Check to ensure that the Password fields have the same
    value -->
    else if (document.forms[0].elements[2].value != document.forms[0].elements[3].value) {
        alert( "Your passwords do not match. Please retype and try again." );
        return false;
    }
}
<-- Check to ensure that atleast one of the Check Boxes is
checked -- >
for (j=4; j<=5; j++) {
    if(document.forms[0].elements[j].checked) {
        break;
    }
}
```

```

        else if (j>=5) {
            alert("Atleast Check on One of Our Services");
            document.forms[0].elements[j].focus();
            return (false);
        }
    }
    return(true);
}
}
<-- Function called on the Clicked event of the Abort button -->
function Abort() {
    history.back();
}
</SCRIPT></HEAD>
<BODY Background="images/Grid1.gif">
    <CENTER><IMG Src="images/RegGoodies.gif"></CENTER>
    <FORM Action="" Method="post" onSubmit="return verify()">
        <TABLE Align=center Border=0 CellPadding=0 CellSpacing=0 Size="800">
            <TBODY><TR>
                <TD Align=right><B>Login Name:</B></TD>
                <TD><INPUT Name=LoginName></TD>
            </TR><TR>
                <TD Align=right><B>E-Mail Address:</B></TD>
                <TD><INPUT Name=Email></TD>
            </TR><TR>
                <TD Align=right><B>Password:</B></TD>
                <TD><INPUT Name=Passwd1 Type=password></TD>
            </TR><TR>
                <TD Align=right><B>Confirm Password:</B></TD>
                <TD><INPUT Name=Passwd2 Type=password></TD>
            </TR></TBODY></TABLE>
            <P><TABLE Align=center Border=0 CellPadding=0 CellSpacing=0 Size="800">
                <TBODY><TR><TD>
                    <FONT Color=red Face=Verdana Size=1><B>PASSWORD RULES:</B></FONT>
                </TD></TR><TR><TD>
                    <FONT Color=red Face=Verdana Size=1>A Password's minimum length is five
                    characters</FONT>
                </TD></TR><TR><TD>
                    <FONT Color=red Face=Verdana Size=1>A Password's maximum length is eight
                    characters</FONT>
                </TD></TR><TR><TD>
                    <FONT Color=red Face=Verdana Size=1>A Password should have at least one digit
                    included</FONT>
                </TD></TR><TR><TD>
                    <FONT Color=red Face=Verdana Size=1>Other than Alphabets and Digits no other
                    values are allowed</FONT>
                </TD></TR></TBODY></TABLE>
            <CENTER><IMG Height=25 Src="images\Shim.gif" Width=1>
                <IMG Src="images/SeleGoodies.gif">
            <IMG Height=30 Src="images\Shim.gif" Width=1></CENTER>

```

```
<TABLE Align=center Border=0 CellPadding=0 CellSpacing=0 Size="800">
<TBODY><TR>
  <TD><IMG Height=5 Src="images/Shim.gif" width=10></TD>
  <TD><B>Pen Pals</B><IMG Height=1 Src="images/Shim.gif" Width=3>
    <INPUT type=checkbox Name=PenPals value="y"></TD>
  <TD><IMG Height=5 Src="images/Shim.gif" width=10></TD>
  <TD><B>News Letter</B><IMG Height=1 Src="images/Shim.gif" Width=3>
    <INPUT type=checkbox Name=NewsLtr value="y" ></TD>
  <TD><IMG Height=5 Src="images/Shim.gif" Width=10></TD>
</TR></TBODY></TABLE>
<CENTER><IMG Height=30 Src="images/Shim.gif" Width=1></CENTER>
<P><P><CENTER>
  <INPUT Type=submit Value=Submit>
  <INPUT Type=reset Value=Reset>
  <INPUT onClick=Abort() Type=button Value=Abort>
</CENTER>
</FORM>
</BODY>
</HTML>
```



# ANSWERS TO SELF REVIEW QUESTIONS

## 8. INTRODUCTION TO JAVASCRIPT

### FILL IN THE BLANKS

1. Form
2. Netscape
3. <SCRIPT></SCRIPT>
4. <FORM></FORM>
5. Variables
6. (\$) character
7. implicitly defined, literal
8. string
9. Functions
10. Arrays
11. dense
12. Objects
13. %(modulus)

### TRUE OR FALSE

14. False
15. True
16. False
17. True
18. False
19. True
20. True
21. True
22. True

## 9. OBJECTS IN JAVASCRIPT

### FILL IN THE BLANKS

1. Document Object Model
2. JavaScript Assisted Style Sheets
3. Method
4. Interactive, Non Interactive
5. Navigator

### TRUE OR FALSE

6. False
7. False
8. True

## 10. FORMS OF A WEB SITE

### FILL IN THE BLANKS

1. <FORM> and </FORM>
2. Method and Action
3. Get or Post
4. Action
5. <INPUT>
6. Submit
7. <MULTIPLE>
8. Length
9. Math

### TRUE OR FALSE

10. True
11. False
12. True
13. False
14. True
15. True

## 11. COOKIES

### FILL IN THE BLANKS

1. Cookie
2. HTTP
3. set-cookie
4. Request header
5. SECURE

### TRUE OR FALSE

6. False
7. True

## SOLUTIONS TO HANDS ON EXERCISES

### 8. INTRODUCTION TO JAVASCRIPT

1. Creating a JavaScript code block using arrays to generate the current date in words, (in the format: Saturday, January 01, 2000).

```
<HTML>
  <HEAD><TITLE> Date validations </TITLE><SCRIPT>
    var monthNames = new Array(12);
    monthNames[0]= "January";
    monthNames[1]= "February";
    monthNames[2]= "March";
    monthNames[3]= "April";
    monthNames[4]= "May";
    monthNames[5]= "June";
    monthNames[6]= "July";
    monthNames[7]= "August";
    monthNames[8]= "September";
    monthNames[9]= "October";
    monthNames[10]= "November";
    monthNames[11]= "December";
    var dayNames = new Array(7) ;
    dayNames[0]= "Sunday";
    dayNames[1]= "Monday";
    dayNames[2]= "Tuesday";
    dayNames[3]= "Wednesday";
    dayNames[4]= "Thursday";
    dayNames[5]= "Friday";
    dayNames[6]= "Saturday";
    function customDateString (m_date) {
      var daywords = dayNames[m_date.getDay()];
      var theday = m_date.getDate();
      var themonth = monthNames[m_date.getMonth()];
      var theyear = m_date.getYear() + 1900;
      return daywords + ", " + themonth + " " + theday + ", " + theyear;
    }
  </SCRIPT></HEAD>
  <BODY><H1>WELCOME!</H1>
    <SCRIPT>document.write(customDateString( new Date()))</SCRIPT>
  </BODY>
</HTML>
```

2. Creating a JavaScript code block, which checks the contents entered in a form's Text element. If the text entered is in the lower case, convert to upper case.

```
<HTML>
<HEAD><SCRIPT Language="JavaScript">
    function checkData(column_data) {
        if ( column_data != "" && column_data.value != column_data.value.toUpperCase() ) {
            column_data.value = column_data.value.toUpperCase()
        }
    }
</SCRIPT></HEAD>
<BODY><FORM><B>Field 1:</B>&nbsp;<input type="text" name="collector" size=10 onChange="checkData(this)"><BR>
<B>Field 2:</B>&nbsp;<input type="text" name="dummy" size=10>
</FORM></BODY>
</HTML>
```

3. Creating a JavaScript code block, which validates a username and password against hard coded values.

```
<HTML>
<HEAD><TITLE>Password Validation</TITLE>
<SCRIPT language="JavaScript">
<!--
    var userpassword = new Array(4);
    userpassword[0]= "Allwyn";
    userpassword[1]= "allwyn";
    userpassword[2]= "Rohan";
    userpassword[3]= "rohan";
    function checkOut() {
        var flag =0;
        var flag1 =0;
        var i =0;
        var j =0;
        for (x=0; x<document.survey.elements.length; x++) {
            if (document.survey.elements[x].value == "") {
                alert("You forgot one of the required fields. Please try again");
                return;
            }
        }
        var user= document.survey.elements[0].value;
        var password = document.survey.elements[1].value;
        while(i<=3) {
            if(userpassword[i] == user) {
                j=i;
                j++;
                if(userpassword[j] == password) {
                    flag=1;
                    break;
                }
            }
        }
    }
-->
```

```

        i+=2;
    }
    if(flag == 0) {
        alert("Please enter a valid user name and password");
        return;
    }
    else {
        alert("Welcome !!\n" + document.forms[0].Username.value);
    }
    return;
}
//-->
</SCRIPT></HEAD>
<BODY><FORM Action="" Method="POST" onSubmit="return checkOut(this.form)"
        Name="survey" >
    <INPUT Type="TEXT" Name="Username" Size="15" MaxLength="15">User Name<BR>
    <INPUT Type="PASSWORD" Name="Pasword" size="15">Password<BR>
    <INPUT Type="SUBMIT" Value="Submit">
    <INPUT Type="RESET" Value="Start Over">
</FORM></BODY>
</HTML>

```

## 9. OBJECTS IN JAVASCRIPT

1. Creating a Web page using two image files, which switch between one another as the mouse pointer moves over the images.

```

<HTML>
<HEAD><TITLE>CHANGING IMAGES..</TITLE><SCRIPT>
    if(document.images) rollover="yes";
    else rollover="no";
    if(rollover=="yes") {
        img1on = new Image();
        img1off = new Image();
        img1on.src="man1.gif";
        img1off.src="man2.gif";
    }
    function imageSwitchOn(imagename) {
        if(rollover=="yes") {
            imageOn=eval(imagename + "on.src");
            document [imagename].src = imageOn;
        }
    }
    function imageSwitchOff(imagename) {
        if (rollover=="yes") {
            imageOff=eval(imagename + "off.src");
            document [imagename].src = imageOff;
        }
    }
</SCRIPT></HEAD>

```

```

<BODY><CENTER>
  <H1>IMAGES.....</H1><BR>
  <H3><I>place your mouse pointer on the picture</I></H3>
  <A onmouseover="imageSwitchOn('img1')" onmouseout="imageSwitchOff('img1')">
    <IMG SRC="man2.gif" height=200 width=200 alt="space" name="img1"></A>
</CENTER></BODY>
</HTML>

```

## 10. FORMS OF A WEB SITE

1. Creating a Web page, which accepts user information and user comments on the web site to check if all the Text fields have being entered with data else display an alert.

```

<HTML>
<HEAD><TITLE>Elements of a Form</TITLE>
<SCRIPT language="JavaScript">
<!--
  function checkOut() {
    for (x=0; x<document.survey.elements.length; x++) {
      if (document.survey.elements[x].value == "") {
        alert("Sorry, you forgot one of the required fields. Please try again.");
        break;
      }
    }
    return false;
  }
  //-->
</SCRIPT></HEAD>
<BODY bgcolor="lightyellow"><H2><I>INFONET SERVICES</I></H2>
  <FORM NAME="survey" onSubmit="return checkOut(this.form)">
    <INPUT Name="firstname" Size="30" Type="TEXT" MaxLength="30">
    <B>First Name</B><BR><BR>
    <INPUT Type="TEXT" Name="lastname" Size="30"><B>Last Name</B><BR><BR>
    <INPUT Type="TEXT" Name="emailaddr" Size="30"><B>E-mail Address</B><BR><BR>
    <INPUT Type="TEXT" Name="address" Size="30"><B>Address</B><BR><BR>
    <INPUT TYPE="TEXT" NAME="city" SIZE="30"><B>City</B><BR><BR>
    <INPUT NAME="state" SIZE="6"><b>State</b>
    <INPUT NAME="zip" SIZE="10"><b>Postal Code</b>
    <INPUT NAME="country" SIZE="15"><b>Country</b>
    <P><B><I>Please choose the most appropriate statement</I></B><BR>
    <INPUT Type="RADIO" Name="buying" Value="regular">
      <B>I regularly purchase items online</B><BR>
    <INPUT Type="RADIO" Name="buying" Value="sometimes">
      <B>I have on occasion purchased items online</B><BR>
    <INPUT Type="RADIO" Name="buying" Value="might" CHECKED>
      <B>I have not purchased anything online, but I would consider it</B><BR>
    <INPUT Type="RADIO" Name="buying" Value="willnot">
      <B>I prefer to shop in real stores</B>
    <P><B><I>I'm interested in (choose all that apply)</I></B><BR>
    <INPUT TYPE="CHECKBOX" NAME="hiking" VALUE="hiking"><B>Hiking</B><BR>

```

```
<INPUT Type="CHECKBOX" Name="mbiking" Value="mbiking">
  <B>Mountain Biking</B><BR>
<INPUT Type="CHECKBOX" Name="camping" Value="camping">
  <B>Camping</B><BR>
<INPUT Type="CHECKBOX" NAME="rock" Value="rock">
  <B>Rock Climbing</B><BR>
<INPUT Type="CHECKBOX" Name="4wd" Value="4wd">
  <B>Off-Road 4WD</B><BR>
<INPUT Type="CHECKBOX" Name="ccskiing" Value="ccskiing">
  <B>Cross-country Skiing</B>
<P><B><I>I learned about this site from</I></B><BR>
<SELECT NAME="referral">
  <OPTION VALUE="print" SELECTED>Print Ads
  <OPTION VALUE="visit"> In-Store Visit
  <OPTION VALUE="rec"> Friend's Recommendation
  <OPTION VALUE="internet"> Sources on the Internet
  <OPTION VALUE="other"> Other
</SELECT>
<P><H4>Comments</H4><BR>
<TEXTAREA Name="comments" Cols="40" Rows="5">
  Please type any comments here</TEXTAREA><BR>
<INPUT Type="SUBMIT" Value="Submit">
<INPUT Type="RESET" Value="Start Over">
</FORM>
</BODY>
</HTML>
```

## SECTION - III: Dynamic Hyper Text Markup Language

### 12. DYNAMIC HTML

- First Impression* - Did the initial page grab attention?
- Interface Design* - Is the menu interface interactive enough and visually interesting?
- Corporate Mildew* - Is the site trapped in a web of corporate look, feel and canned marketing speak?
- Coriolis Effect* - Does the site generate enough currents of interest based on design and content for the user to comeback?

The above points emphasize the requirements of a good web site.

DHTML is a new and emerging technology that has evolved to meet the increasing demand for eye-catching and mind-catching web sites.

DHTML combines HTML with Cascading Style Sheets (CSSs) and Scripting Languages. HTML specifies a web page's elements like table, frame, paragraph, bulleted list, etc. Cascading Style Sheets can be used to determine an element's size, color, position and a number of other features. Scripting Languages (JavaScript and VBScript) can be used to manipulate the web page's elements so that styles assigned to them can change in response to a user's input.

### CASCADING STYLE SHEETS

Style Sheets are powerful mechanism for adding styles (e.g. fonts, colors, spacing) to Web documents. They enforce standards and uniformity throughout a web site and provide numerous attributes to create dynamic effects. With Style Sheets, text and image formatting properties can be predefined in a single list. HTML elements on a web page can then be bound to the style sheet. The advantages of a Style Sheet includes the ability to make global changes to all documents from a single location. Style Sheets are said to Cascade when they combine to specify the appearance of a page.

The Style assignment process is accomplished with the `<STYLE>...</STYLE>` tags. The syntax for making the assignment is simple. Between the `<STYLE>...</STYLE>` HTML tags, specific style attributes are listed. The `<STYLE>...</STYLE>` tags are written within the `<HEAD>...</HEAD>` tags.

#### Syntax:

```
<STYLE Type="text/css">  
  tag {attribute:value; attribute:value ... }  
  ...  
</STYLE>
```

#### Note



In the `<STYLE>` tag, the expression "Type = text/css" indicates that the style sheet conforms to CSS syntax

The attributes that can be specified to the <STYLE> tag are Font Attributes, Color and Background Attributes, Text Attributes, Border Attributes, Margin Attributes and List Attributes.

## Font Attributes

Attributes	Values
font-family	A comma-delimited sequence of font family names (serif, sans-serif, cursive)
font-style	Normal, italic or oblique
font-weight	Normal, bold, bolder, lighter, or one of the nine numerical values (100, 200, 300, 400, 500, 600, 700, 800, 900)
font-size	A term that denotes absolute size (xx-small, x-small, small, medium, large, x-large, xx-large), relative size (larger, smaller), a number (of pixels), percentage (of the parent element's size)

Table 12.1

### Use Of Font Attributes

Example 1: (Refer to diagram 12.1)

```
<HTML>
  <HEAD><TITLE> Working with
    Style Sheets using Font
    Attributes </TITLE>
  <STYLE Type = "text/css">
    H1{font-family:arial, helvetica}
    P{font-size:12pt; font-style:italic}
  </STYLE></HEAD>
  <BODY>
    <H1>Silicon Chip
      Technologies</H1>
    <P>Silicon Chip Technologies, a
      private limited company, was
      founded in December 1989.
    <P>The vision of this company is to provide any corporate client a single entity which addresses
      all their Software Development, Technical and User Documentation, Training and Manpower
      Recruitment needs.
  </BODY>
</HTML>
```

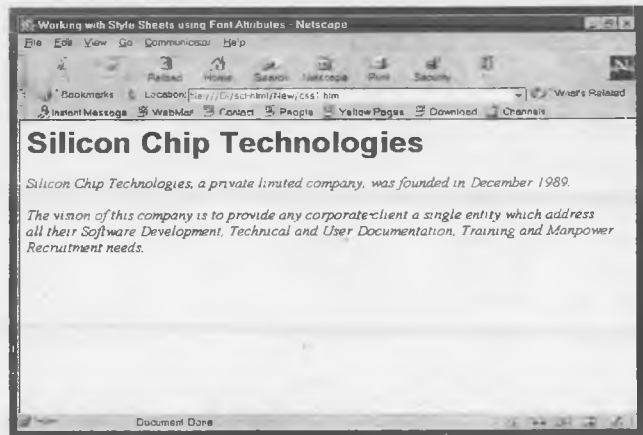


Diagram 12.1

## Color And Background Attributes

Attributes	Values
color	Sets an element's text-color. <i>A color name or a color code</i>
background-color	Specifies the color in an element's background. <i>A color name or a color code</i>
background-image	Sets the background image. <i>A URL or none</i>
background-repeat	With a background image specified, sets up how the image repeats throughout the page. <i>repeat-x</i> (repeats horizontally), <i>repeat-y</i> (repeats vertically), <i>repeat</i> (both), <i>no-repeat</i>

Table 12.2



Use Of Color And Background Attributes

Example 2: (Refer to diagram 12.2)

```

<HTML>
  <HEAD><TITLE> Working with Style
    Sheets using Color and
    Background
    Attributes</TITLE>
  <STYLE Type = "text/css">
    H1 {font-family:arial, helvetica;
      font-size:26pt; background-
      image:url(images/sct-logo.gif)}
    H2 {font-family:arial, helvetica;
      font-size:26pt; background-
      image:url(images/sct-logo.gif);
      background-repeat:no-repeat}
    P {font-size:12pt; font-style:italic;
      font-weight:bold;
      color:#23238e; background-color:red; background-position:bottom-left}
  </STYLE></HEAD>
  <BODY>
    <B><U>With background repeat</U></B>
    <H1> Silicon Chip Technologies</H1>
    <P>A private limited company, which was founded in December 1989. The vision of this
    company is to provide any corporate client a single entity which addresses all their Software
    Development, Technical and User Documentation, Training and Manpower Recruitment needs.
    </P><BR>
    <B><U>With background no-repeat</U></B>
    <H2> Silicon Chip Technologies</H2>
  </BODY>
</HTML>
  
```

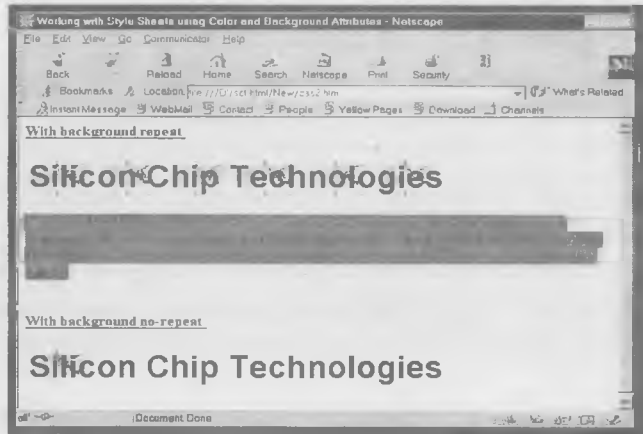


Diagram 12.2

**Text Attributes**

Attributes	Values
text-decoration	Adds decoration to an element's text. <i>None, underline, overline, line-through, blink</i>
vertical-align	Determines an element's vertical position, <i>Baseline, sub, super, top, text-top, middle, bottom, text-bottom</i> , also <i>percentage of the element's height</i>
text-transform	Applies a transformation to the text. <i>Capitalize</i> (puts the text into initial caps), <i>uppercase, lowercase</i> or <i>none</i>
text-align	Aligns text within an element. <i>Left, right, center, or justify</i>
text-indent	Indents the first line of text. <i>A percentage of the element's width, or a length</i>

Table 12.3

*Tip*



**Length values** can be positive or negative, have a numerical value, and are followed by a unit of measurement. Note that while length values can be either positive or negative, many properties cannot have negative length units.

There are two basic kinds of length unit, **relative** and **absolute**. As a rule of thumb, **absolute** units should be used only when the physical characteristics of the output device are known.

The units of measurement are:

Unit Name	Abbreviation	Explanation	Relative
Em	Em	the height of a font	yes
Ex	Ex	height of the letter x in a font	yes
Pica	Pc	1 pica is 12 points	no
Point	Pt	1/72 of an inch	no
Pixel	Px	one dot on a screen	yes
Millimeter	Mm	printing unit	no
Centimeter	Cm	printing unit	no
Inch	In	printing unit	no

Table 12.4

### Use of Text Attributes

Example 3: (Refer to diagram 12.3)

<HTML>

```
<HEAD><TITLE> Working with Style Sheets using Text Attributes
</TITLE>
```

```
<STYLE Type = "text/css">
```

```
H1 {font-family:arial, helvetica;
font-size:26pt; text-decoration:blink; color:red}
```

```
P {font-size:12pt; font-style:normal;
font-weight:bold; color:#23238e;}
```

```
H6 {font-size:12pt; font-style:italic;
font-weight:bold; color:#23238e; text-align:justify; text-indent:.5in}
```

```
</STYLE></HEAD>
```

```
<BODY>
```

```
<H1>Silicon Chip Technologies</H1><B><U>Without text align, first line indent</U></B>
```

```
<P>A private limited company, which was founded in December 1989. The vision of this
company is to provide any corporate client a single entity which addresses all their Software
Development, Technical and User Documentation, Training and Manpower Recruitment
needs.</P>
```

```
<B><U>With text align (justify), first line indent</U></B>
```

```
<H6>A private limited company, which was founded in December 1989. The vision of this
company is to provide any corporate client a single entity which addresses all their Software
Development, Technical and User Documentation, Training and Manpower Recruitment
needs.</H6>
```

```
</BODY>
```

```
</HTML>
```

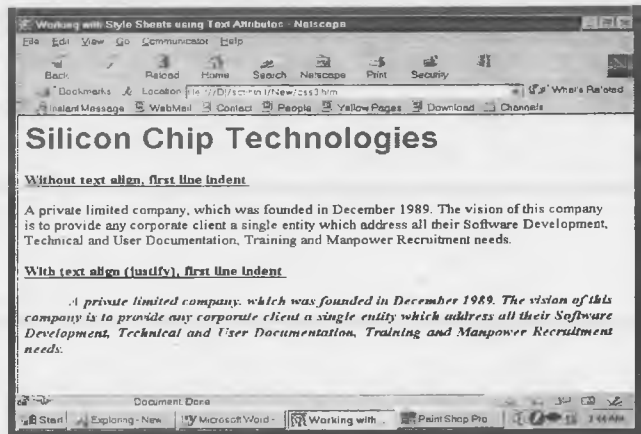


Diagram 12.3

**Border Attributes**

Attributes	Values
border-style	<i>Solid, double, groove, ridge, inset, outset</i>
border-color	<i>A color name or color code</i>
border-width	<i>Thin, medium, thick or length</i>
border-top-width	<i>Thin, medium, thick or length</i>
border-bottom-width	<i>Thin, medium, thick or length</i>
border-left-width	<i>Thin, medium, thick or length</i>
border-right-width	<i>Thin, medium, thick or length</i>
border-top	Specifies width, color and style
border-bottom	Specifies width, color and style
border-left	Specifies width, color and style
border-right	Specifies width, color and style
border	Sets all the properties at once

**Table 12.5**

**Use Of Border Attributes**

**Example 4:** (Refer to diagram 12.4)

<HTML>

```
<HEAD><TITLE>Working with Style
      Sheets using Border
      Attributes </TITLE>
```

```
<STYLE Type = "text/css">
```

```
  H1 {font-family:arial, helvetica;
      font-size:26pt; color:red}
```

```
  P {font-size:12pt; font-style:italic;
      font-weight:bold;
```

```
    color:#23238e; border-
    color:#23238e; border-
    style:groove; border-
    width:thick}
```

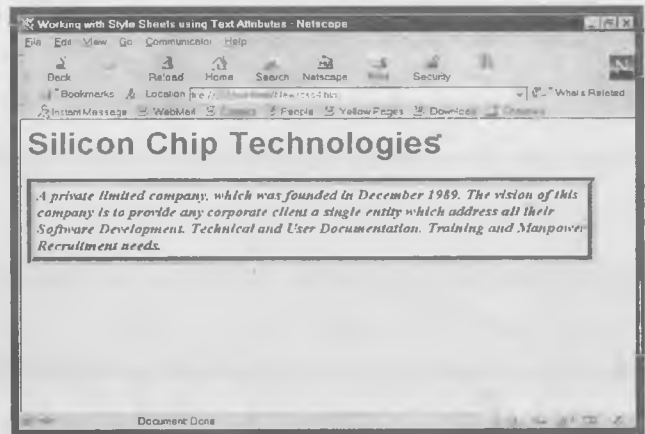
```
</STYLE></HEAD>
```

```
<BODY><H1> Silicon Chip Technologies</H1>
```

```
<P>A private limited company, which was founded in December 1989. The vision of this
  company is to provide any corporate client a single entity which addresses all their Software
  Development, Technical and User Documentation, Training and Manpower Recruitment
  needs.</P>
```

```
</BODY>
```

```
</HTML>
```



**Diagram 12.4**

**Margin Related Attributes**

Attributes	Values
margin-top	<i>Percent, length or auto</i>
margin-bottom	<i>Percent, length or auto</i>
margin-left	<i>Percent, length or auto</i>
margin-right	<i>Percent, length or auto</i>
margin	<i>Percent, length or auto</i>

**Table 12.6**

Use Of Margin Attributes

Example 5: (Refer to diagram 12.5)

```
<HTML>
  <HEAD><TITLE> Working with Style
  Sheets using Margin Attributes</TITLE>
  <STYLE Type = "text/css">
    BODY {margin-top:10%}
    H1 {font-family:arial, helvetica;
    font-size:26pt; color:red}
    P {font-size:12pt; font-style:italic;
    font-weight:bold;
    color:#23238e; margin-
    left:15%; margin-right:15%}
  </STYLE> </HEAD>
  <BODY><H1>Silicon Chip
  Technologies</H1>
  <P>A private limited company, which was founded in December 1989. The vision of this
  company is to provide any corporate client a single entity which addresses all their Software
  Development, Technical and User Documentation, Training and Manpower Recruitment
  needs.</P>
  </BODY>
</HTML>
```



Diagram 12.5

A private limited company, which was founded in December 1989. The vision of this company is to provide any corporate client a single entity which addresses all their Software Development, Technical and User Documentation, Training and Manpower Recruitment needs.

**List Attributes**

Attributes	Values
list-style	Disc, circle, square, decimal, lower-roman, upper-roman, lower-alpha, upper-alpha, none

Table 12.7

Use Of List Attributes

Example 6: (Refer to diagram 12.6)

```
<HTML>
  <HEAD><TITLE> Working with Style
  Sheets using List Attributes
  </TITLE>
  <STYLE Type = "text/css">
    BODY {margin-top:5%}
    H1 {font-family:arial, helvetica;
    font-size:26pt; color:red}
    UL {list-style-type:lower-roman}
  </STYLE></HEAD>
  <BODY><H1>Silicon Chip
  Technologies</H1>
  <H4>SCT Provides Corporate
  Training For The Following Products:</H4>
  <B><UL>
  <LI> HTML
  <LI> Javascript
  <LI> CGI
```



Diagram 12.6

```

<LI> Java
<LI> PowerBuilder
<LI> Oracle Developer 2000
<LI> Oracle DBA
</UL></B>
</BODY>
</HTML>

```

## CLASS

The control over page design that style sheet gives is exciting, but can be heavy handed. Its great to be able to change every paragraph, but what if only one paragraph, or a few paragraphs need to be changed?

A particular paragraph may need to look different from other paragraphs. It's probably because the content of the paragraph is in some way different from other paragraphs on the page. Think of a question and answer page. Questions in bold, while the answers are in plain text. The appearance of a paragraph is a function of the content of the paragraph.

Style sheets support classes or sets of style changes for a document. A class can be defined to change the style in a specific way for any element it is applied to, and classes can be used to identify logical sets of style changes that might be different for different HTML elements. The style changes can be applied directly to each HTML element or applied to part of a document with the `<SPAN>` `</SPAN>` tags. If any element is made a member of a class by inserting `Class = ClassName` into its opening tag, it conforms to that class specification.

In the example above, there are two classes of paragraph - question and answer. So in the style sheet there need to be two statements, one which affects only paragraphs of *class* question, and one which only affects paragraphs of *class* answer.

In a style sheet, class is defined by a dot followed by the name of the class i.e. `.class-name`.

### Use Of Class

**Example 7:** (Refer to diagram 12.7)

```

<HTML>
<HEAD><TITLE> Working with
Class </TITLE>
<STYLE Type = "text/css">
P {font-size:12pt; font-
weight:bold; text-align:justify;
margin-left:10%; margin-
right:10%}
.question {color:brown; font-
style:italic}
.answer {color:#3238e}
</STYLE></HEAD>
<BODY>
<P class="question"> How to
create style sheet to alter an
HTML element ? </P>

```

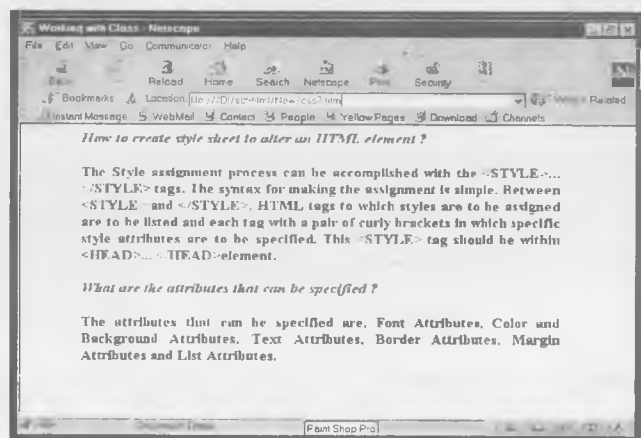


Diagram 12.7

<P class="answer"> The Style assignment process can be accomplished with the **&lt;STYLE&gt;...&lt;/STYLE&gt;** tags. The syntax for making the assignment is simple. Between **&lt;STYLE&gt;** and **&lt;/STYLE&gt;**, HTML tags to which styles are to be assigned are to be listed and each tag with a pair of curly brackets in which specific style attributes are to be specified. This **&lt;STYLE&gt;** tag should be within **&lt;HEAD&gt;...&lt;/HEAD&gt;**element.</P>

<P class="question"> What are the attributes that can be specified ? </P>

<P class="answer"> The attributes that can be specified are, Font Attributes, Color and Background Attributes, Text Attributes, Border Attributes, Margin Attributes and List Attributes.</P>

</BODY>

</HTML>

## USING THE <SPAN>...</SPAN> TAG

SPAN is an HTML element that plays a prominent role in style sheets. In the body of the document, **<SPAN>...</SPAN>** is used to set the boundaries of the rule's styling specifications.

### Use Of SPAN Tags

**Example 8:** (Refer to diagram 12.8)

<HTML>

```
<HEAD><TITLE>Working with
SPAN</TITLE>
<STYLE Type = "text/css">
P {font-size:12pt;font-
weight:bold;text-align:justify}
.question {color:brown;font-
style:italic}
.answer {color:#23238e}
.big {font-size:14pt;text-
decoration:underline;text-
transform:uppercase;color:red}
</STYLE></HEAD>
```

<BODY>

<P class="question"> How to create **<SPAN class="big">** style sheet **</SPAN>** to alter an HTML element ?</P>

<P class="answer"> The Style assignment process can be accomplished with the **&lt;STYLE&gt;...&lt;/STYLE&gt;** tags. The syntax for making the assignment is simple. Between **&lt;STYLE&gt;** and **&lt;/STYLE&gt;**, HTML tags to which styles are to be assigned are to be listed and each tag with a pair of curly brackets in which specific style attributes are to be specified. This **&lt;STYLE&gt;** tag should be within **&lt;HEAD&gt;...&lt;/HEAD&gt;**element.</P>

<P class="question"> What are the **<SPAN class="big">** attributes **</SPAN>** that can be specified?</P>

<P class="answer"> The attributes that can be specified are, Font Attributes, Color and Background Attributes, Text Attributes, Border Attributes, Margin Attributes and List Attributes.</P>

</BODY>

</HTML>

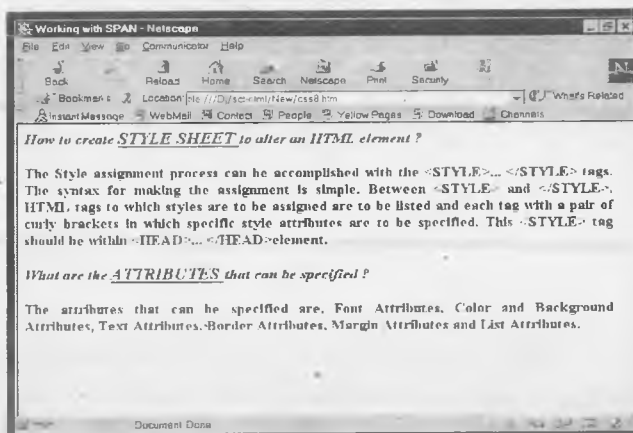


Diagram 12.8



To protect browsers that do not support `<STYLE>` element, insert comment tags around the declarations within the style element.

```
<STYLE Type = "text/css">
  <!--
    /* declaration */
  -->
</STYLE>
```

## EXTERNAL STYLE SHEETS

External Style Sheets are composed of standard text, which consists of a series of entries, each composed of a selector and a declaration. The selector indicates the HTML element(s) affected by the properties in the declaration. These external style sheets are then saved as a file with extension .css, which can be linked, to a web page via the `<LINK>` tag.

```
<LINK Rel=stylesheet HRef="<StyleSheet File Name>">
```

### Use Of External Style Sheet

Example 9: (Refer to diagram 12.9)

#### Code for mystyle.css:

```
P {font-size:12pt;font-weight:bold; text-align:justify}
.question {color:brown; font-style:italic}
.answer {color:#23238e}
.big {font-size:14pt; text-decoration:underline; text-transformation:uppercase; color:red}
```

#### Code for the HTML page:

```
<HTML>
  <HEAD><TITLE> Working with
    External Style Sheet
  </TITLE>
  <LINK Rel=stylesheet HREF="mystyle.css"></HEAD>
  <BODY>
    <P class="question">How to create <SPAN class="big"> style sheet </SPAN> to alter an HTML
    element?</P>
    <P class="answer"> The Style assignment process can be accomplished with the
    &lt;STYLE&gt;...&lt;/STYLE&gt; tags. The syntax for making the assignment is simple.
    Between &lt;STYLE&gt; and &lt;/STYLE&gt;, HTML tags to which styles are to be
    assigned are to be listed and each tag with a pair of curly brackets in which specific style
    attributes are to be specified. This &lt;STYLE&gt; tag should be within &lt;HEAD&gt;...
    &lt;/HEAD&gt;element.</P>
    <P class="question"> What are the <SPAN class="big"> attributes </SPAN> that can be
    specified?</P>
    <P class="answer"> The attributes that can be specified are, Font Attributes, Color and Background
    Attributes, Text Attributes, Border Attributes, Margin Attributes and List Attributes.</P>
  </BODY>
</HTML>
```

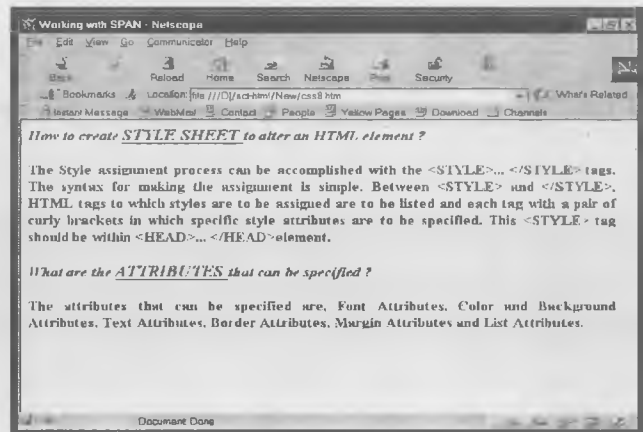


Diagram 12.9

*Tip*

Style information can be associated with the web page in several ways:

- By embedding the style information directly through a STYLE attribute
- `<H1 STYLE = text-align:center> Inline Style Sheet </H1>`
- By embedding the style information directly through a `<STYLE>` header
- By embedding the style information directly through `<LINK>` element

## WORKING WITH JAVASCRIPT STYLE SHEETS [JSSS]

In addition to supporting Cascading Style Sheets, Netscape Communicator also supports a JavaScript-based approach to style sheets, referred to JavaScript Style Sheets or JSSS for short. JSSS support the styles provided by CSS and has the advantage of making these styles available as JavaScript properties. This advantage enables style properties to be created, read and updated via JavaScript scripts.

A JavaScript Style Sheet code with the output shown in diagram 12.10 (a black-and-white representation) is explained in Example 10. The `<STYLE>` tags shown in the listing are surrounding tags. They surround four JavaScript statements that assign color values to different sub properties of the `document.tags` property. The `<STYLE>` tag uses the TYPE attribute to determine what type of style sheet is in effect. The `text/javascript` value is used to identify a JavaScript Style Sheet as shown in Example 10. The `text/css` value would be used to identify a Cascading Style Sheet as shown in Example 11.



Diagram 12.10

### Example 10:

```
<!-- This page only works in
Netscape -->
```

```
<HTML>
```

```
<HEAD><TITLE> Working with JSSS Style Sheets </TITLE>
```

```
<STYLE Type = "text/javascript">
```

```
document.tags.BODY.backgroundColor = 'cyan';
```

```
document.tags.H1.Color = 'red';
```

```
document.tags.H2.Color = 'blue';
```

```
document.tags.P.Color = 'green';
```

```
</STYLE></HEAD>
```

```
<BODY><BR><BR>
```

```
<CENTER><H1>Silicon Chip Technologies</H1><H2>Mumbai</H2></CENTER>
```

```
<P>Silicon Chip Technologies, a private limited company, was founded in December 1989.</P>
```

```
<P>The vision of this company is to provide any corporate client a single entity which address all
their Software Development, Technical and User Documentation, Training and Manpower
Recruitment needs.</P>
```

```
</BODY>
```

```
</HTML>
```



A CSS code equivalent to the above program is explained in Example 11.

**Example 11:** (Refer to diagram 12.10)

```
<HTML>
  <HEAD><TITLE> Working with Cascading Style Sheets </TITLE>
  <STYLE Type="text/css">
    BODY {background-Color: 'cyan';}
    H1 {Color: 'red';}
    H2 {Color: 'blue';}
    P {Color: 'green';}
  </STYLE></HEAD>
  <BODY><BR><BR>
    <CENTER><H1>Silicon Chip Technologies</H1><H2>Mumbai</H2></CENTER>
    <P>Silicon Chip Technologies, a private limited company, was founded in December 1989.</P>
    <P>The vision of this company is to provide any corporate client a single entity which address all
      their Software Development, Technical and User Documentation, Training and Manpower
      Recruitment needs.</P>
  </BODY>
</HTML>
```

## USING THE <DIV>...</DIV> TAG

A web page can be divided into segments or divisions called **DIVs**. Each segment starts with <DIV> and ends with </DIV>. These segments can be positioned anywhere on the page. The <DIV> tag has a 'position' attribute that can take one of the two values, **Absolute** or **Relative**. **Absolute** positions the segment with respect to the top/left edge of the browser window. In contrast with **Absolute**, **Relative** positions the segment in relation to other elements on the page.

### Use Of DIVs

**Example 12:** (Refer to diagram 12.11)

```
<HTML>
  <HEAD><TITLE> Working with DIVs
  </TITLE></HEAD>
  <BODY>
    <DIV ID=box1 Style="background-color:red; position:absolute; left:300; top:150; width:50">
      <IMG Src=images/SCT-LOGO.gif></DIV>
    <DIV ID=box2 Style="background-color:red; position:absolute; left:380; top:150; width:50">
      <IMG Src=images/SCT-LOGO.gif></DIV>
    <DIV ID=box3 Style="background-color:red; position:absolute; left:300; top:190; width:50">
      <IMG Src = images/SCT-LOGO.gif></DIV>
    <DIV ID=box4 Style="background-color:red; position:absolute; left:380; top:190; width:50">
      <IMG Src=images/SCT-LOGO.gif></DIV>
  </BODY>
</HTML>
```



**Diagram 12.11**

These images are actually put in four different segments.

This example shows when the mouse cursor is placed on the first picture, the second picture becomes invisible and viceversa. If the mouse is on neither of the pictures both the pictures are visible.

### Use Of The DIV Tag And Visibility Property

**Example 13:** (Refer to diagram 12.12.1)

```
<HTML>
<HEAD><TITLE>Using DIV
      Tag</TITLE></HEAD>
<BODY><SCRIPT Language="Javascript">
  function computer_onmouseover() {
    text1.style.visibility = "visible";
    computer.style.visibility = "visible";
    text2.style.visibility = "hidden";
    balloon.style.visibility = "hidden";
  }
  function everyone_onmouseout() {
    text1.style.visibility = "visible";
    computer.style.visibility = "visible";
    text2.style.visibility = "visible";
    balloon.style.visibility = "visible";
  }
  function balloon_onmouseover() {
    text1.style.visibility = "hidden";
    computer.style.visibility = "hidden";
    text2.style.visibility = "visible";
    balloon.style.visibility = "visible";
  }
</SCRIPT>
<DIV Style="visibility:visible" onmouseover="computer_onmouseover();"
      onmouseout="everyone_onmouseout();"><CENTER>
  <FONT ID=text1 Color=blue Font=sans-serif>For the Picture below to disappear move the mouse
    over this picture</FONT>
  <P><IMG ID=computer Src="Images/computer.gif" Height=200 Width=200></CENTER>
</DIV><DIV Style="visibility:visible" onmouseover="balloon_onmouseover();"
      onmouseout="everyone_onmouseout();"><CENTER>
  <FONT ID=text2 Color=blue Font=sans-serif>For the above Picture to disappear move the mouse
    over this picture</FONT>
  <P><IMG ID=balloon Src="Images/balloon.gif" Height=200 Width=200></CENTER></DIV>
</BODY>
</HTML>
```



Diagram 12.12.1

When the mouse is not on any of the pictures, the output of above example is as shown in diagram 12.12.1

When the mouse is on the first picture, the output of above example is as shown in diagram 12.12.2 and when the mouse is on the second picture, the output is as shown in diagram 12.12.3.

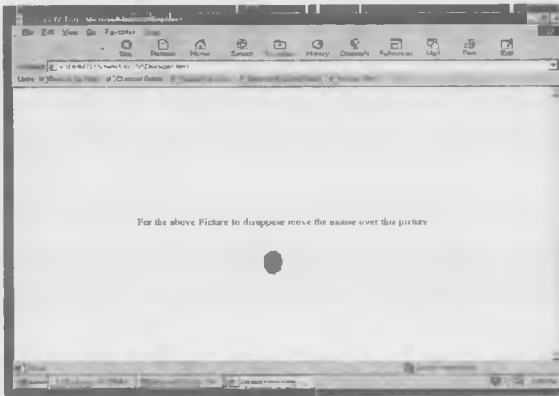


Diagram 12.12.2



Diagram 12.12.3

## LAYERS

To segment a Web page, there is also another pair of DHTML tags, `<LAYER>...</LAYER>`. These tags are designed to behave like a piece of transparent paper laid on top of a web page. Between `<LAYER>...</LAYER>`, HTML elements are inserted and the user is given precise control over the placement of these objects. Each `<LAYER>` is provided with an ID and with attributes and values that specify its appearance and position. JavaScript can be used to dynamically change these values in response to user events. A change to one of the layer's attributes affects all the content in the layer

### Layer Attributes

Attributes	Values
ID/NAME	<i>The name of the layer</i>
Left and Top	<i>The horizontal and vertical positions of the layer in pixels with the top-left corner of the screen being (0,0)</i>
PageX and PageY	<i>The horizontal and vertical positions of the layer relative to the document's window</i>
Src	<i>The pathname of a file that contains HTML formatted content</i>
z-index, above, below	<i>The stacking order of a layer. These three parameters are mutually exclusive. Only one is valid at any given time</i>
Width	<i>The width of the layer's display</i>
Height	<i>The height of a layer's display</i>
Clip	<i>The viewable area of a layer</i>
Visibility	<i>Whether a layer is visible or not. Valid values are SHOW and HIDE</i>
Bgcolor	<i>The background color to be used by the layer</i>
Background	<i>An image to be used as the background for the layer</i>

Table 12.8

### Layer Methods

Method	Description
CaptureEvents(eventType)	<i>Allows the layer to capture all events of the specified type.</i>
HandleEvents(event)	<i>Invokes the event handler for the specified event</i>
MoveAbove(layer)	<i>Moves the layer above the identified layer</i>

Table 12.9

Method	Description
MoveBelow(layer)	Moves the layer below the identified layer
MoveBy(x, y)	Moves the layer by the specified x and y pixel increments
MoveTo(x, y)	Moves the layer to the specified position within the container
ReleaseEvents(eventType)	Ends the capturing of the specified event type

Table 12.9: (Continued)

## Layer Event Handlers

Event Handler	Description
onMouseOver, onMouseOut	Event handler to use when the mouse enters or leaves the layer
onfocus, onBlur	Event handler to use when the layer receives or loses keyboard focus
onLoad	Event handler to use when the layer is first loaded

Table 12.10

## Use Of Layers

**Example 14:** (Refer to diagrams 12.13.1 and 12.13.2)

```
<HTML>
  <HEAD><TITLE>Working with Layers</TITLE></HEAD>
  <BODY>
    <LAYER ID=box1 Left=150 Top=150>
      <IMG Src=images/fence1.jpg Height=100 Width=170>Animal Fencing</LAYER>
    <LAYER ID=box2 Left=50 Top=200>
      <IMG Src=images/barbed.jpg Height=100 Width=140>Barbed Tapes</LAYER>
    This text is before layers.<HR>
    This text is after layers. Notice how the layers are positioned absolutely, independent of the order
    they appear in the document.
  </BODY>
</HTML>
```

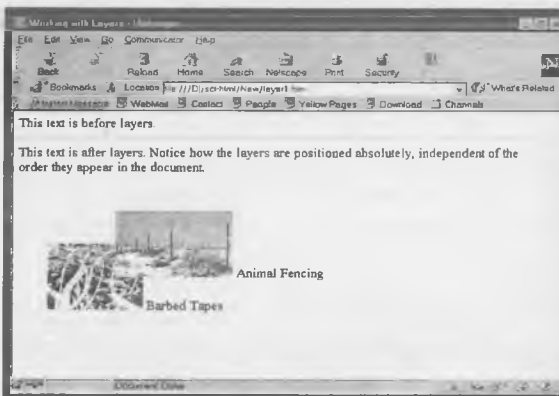


Diagram 12.13.1

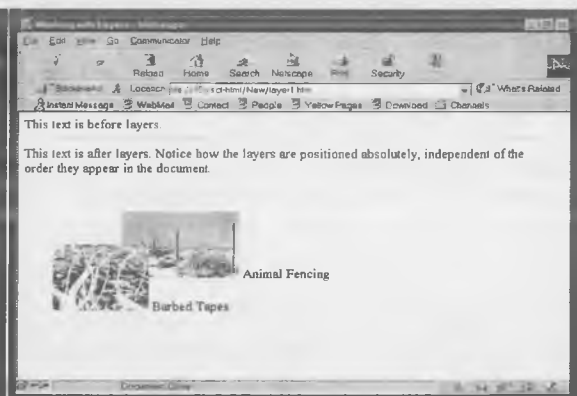


Diagram 12.13.2

In the above output, the second layer is on top of the first layer. The stacking order of a layer is controlled by its **Z-INDEX** attribute. The higher the Z-INDEX, the higher the layer will be stacked.

In the above example, set the Z-INDEX as follows:

```
<LAYER ID=box1 Left=150 Top=150 Z-INDEX=2>
  <IMG Src=images/fence1.jpg Height=100 Width=170>Animal Fencing</LAYER>
<LAYER ID=box2 Left=50 Top=200 Z-INDEX=1>
  <IMG Src=images/barbed.jpg Height=100 Width=140>Barbed Tapes</LAYER>
```

## Inflow Layers

To create a layer that occurs in its "natural" place in a document, an inflow layer with the `<ILAYER>...</ILAYER>` tag is used. This creates a layer that is positioned relatively in the document, much like a graphic is positioned using the `<SRC>` tag. The layer can be nudged into position by using the `LEFT` and `TOP` attributes. In an inflow layer, values passed to its `LEFT` and `TOP` attributes will move the layer away from its natural position in the document.

### Inflow Layers: Without Layers

**Example 15:** (Refer to diagram 12.14.1)

```
<HTML>
  <HEAD><TITLE> Without Layers </TITLE></HEAD>
  <BODY>
    <IMG Src=images/fence1.jpg Height=100 Width=170>Animal Fencing
    <IMG Src=images/barbed.jpg Height=100 Width=140>Barbed Tapes
  </BODY>
</HTML>
```

### Inflow Layers: Use Of Layers

**Example 16:** (Refer to diagram 12.14.2)

```
<HTML>
  <HEAD><TITLE>With Layers</TITLE></HEAD>
  <BODY>
    <ILAYER ID=box1 Left=0 Top=50>
      <IMG Src=images/fence1.jpg Height=100 Width=170>Animal Fencing</ILAYER>
    <ILAYER ID=box2 Left=50 Top=100>
      <IMG Src=images/barbed.jpg Height=100 Width=140>Barbed Tapes</ILAYER>
  </BODY>
</HTML>
```

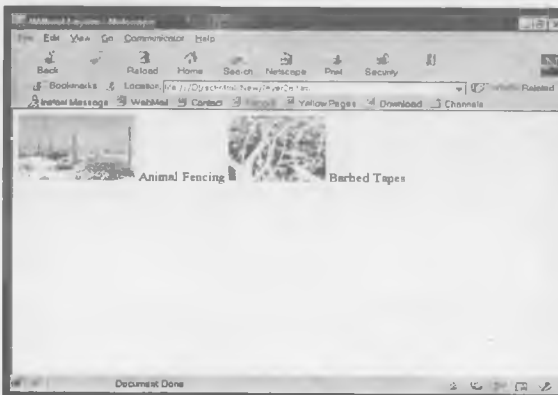


Diagram 12.14.1

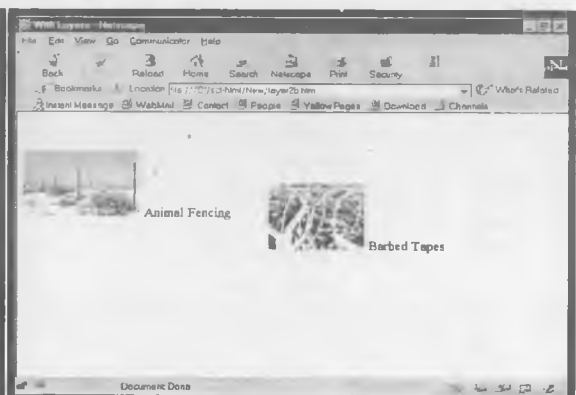


Diagram 12.14.2

*Tip*

Since <LAYER> tags are **not supported** by **Internet Explorer**, defining the layers in the <STYLE> tags and then displaying them and their content with <SPAN> or <DIV> tags works much like <LAYER> tag.



Diagram 12.15

**Example 17:** (Refer to diagram 12.15)

**Focus:** Create a page with dynamic effects. Write the code to include layers and basic animation.

<HTML>

```
<HEAD><TITLE> Working with Layers </TITLE>
<SCRIPT Language="Javascript">
  var glbfwdtimer, glbbacktimer, glbcounter;
  glbcounter = 0;
  function windowload() {
    glbfwdtimer = setInterval("fingermovement()", 200);
  }
  function fingermovement() {
    document.layers["finger1"].offset(2,0);
    glbcounter = glbcounter + 1;
    if (glbcounter == 50) {
      glbcounter = 0;
      clearInterval(glbfwdtimer);
      glbbacktimer = setInterval("backmovement()",100);
    }
  }
  function backmovement() {
    document.layers["finger1"].offset(-2,0);
    glbcounter = glbcounter + 1;
    if (glbcounter == 50) {
      glbcounter = 0;
      clearInterval(glbbacktimer);
      glbfwdtimer = setInterval("fingermovement()",100);
    }
  }
</SCRIPT></HEAD>
<BODY onLoad="windowload()">
  <LAYER ID=finger1 Left=10 Top=60><IMG Src=images/rhand.gif></LAYER>
  <LAYER ID=amconfus Left=150 Top=50 Visibility=show
    onMouseOver="amconfus.visibility='hide'; amidea.visibility='show';
    amvictor.moveTo(350,50)">
```

```

<IMG Src=images/confus.gif Height=100 Width=75> </LAYER>
<LAYER ID=amidea Left=150 Top=50 Visibility=hide onMouseOut="amidea.visibility='hide';
    amconfus.visibility='show'; amvictor.moveTo(250,50)">
    <IMG Src=images/idea.gif Height=100 Width=75></LAYER>
<LAYER ID=amvictor Left=250 Top=50 Visibility='show'>
    <IMG Src=images/victor.gif Height=100 Width=75></LAYER>
</BODY>
</HTML>

```

In the above example when the mouse cursor is over the 'confuse' image, the original image changes to another image 'idea' and the other image 'victor' moves forward. When the mouse cursor is moved out of the 'idea' image it changes back to the 'confuse' image and the 'victor' image moves back to its original position.

Additionally, animation has been introduced through the movement of the hand via JavaScript.

### Use Of Drag And Drop

**Example 18:** (Refer to diagram 12.16)

**Focus:** This program demonstrates the Drag and Drop feature using Layers. This code gets the position of the mouse cursor when it is clicked and checks whether the mouse cursor is over a drag-and-droppable object. If so, the new position of the mouse cursor is recorded as and when the selected object moves from one position to another. The entire information of the location of the selected object is displayed in the text area

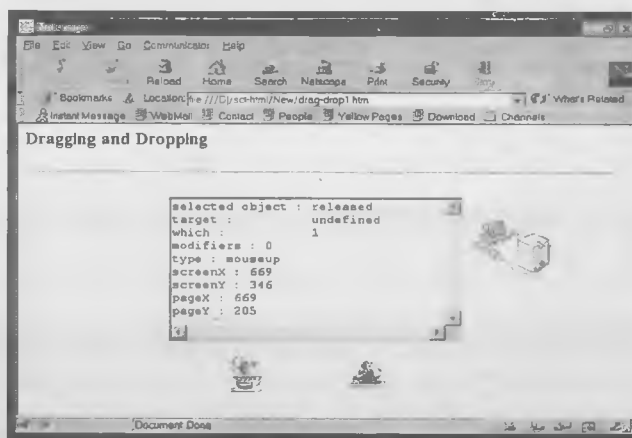


Diagram 12.16

```

<HTML>
<HEAD><SCRIPT
    Language="Javascript">
    var glbSelectedObject = null;
    glbCurrentX = 0;
    glbCurrentY = 0;
    document.captureEvents(Event.MOUSEDOWN | Event.MOUSEMOVE | Event.MOUSEUP);
    document.onmousedown = selectTheObject;
    document.onmousemove = dragTheObject;
    document.onmouseup = dropTheObject;
    function selectTheObject(e) {
        var workObject;
        cursorX = e.pageX;
        cursorY = e.pageY;
        for (i=0; i<document.layers.length; i++) {
            workObject = document.layers[i];
            if (workObject.id.indexOf("DnD") == -1) { continue }
            if ((cursorX > workObject.left) && (cursorX < (workObject.left +
                workObject.clip.width)) && (cursorY > workObject.top) && (cursorY <
                (workObject.top + workObject.clip.height))) {
                glbSelectedObject=workObject; }
        }
    }

```

```

    if (glbSelectedObject == null) {        . return    }
    glbCurrentX = e.pageX;
    glbCurrentY = e.pageY;
    info(e);
    document.captureEvents(Event.MOUSEMOVE);
    document.onmousemove = dragTheObject;
}
function dragTheObject(e) {
    if (glbSelectedObject == null) {        return    }
    distanceX = (e.pageX - glbCurrentX);
    distanceY = (e.pageY - glbCurrentY);
    glbCurrentX = e.pageX;
    glbCurrentY = e.pageY;
    lbSelectedObject.moveBy(distanceX, distanceY);
    info(e);
}
function dropTheObject(e) {
    glbSelectedObject = null;
    info(e);
    releaseEvents(Event.MOUSEMOVE);
}
function info(event) {
    var selection;
    if (glbSelectedObject != null) {        selection = glbSelectedObject.id    }
    else {    selection = "released"        }
    textArea = document.forms[0].elements["textareaEventInfo"];
    textArea.value = "selected object : " + selection + "\n" + "which : " + event.which + "\n".
        + "modifiers : " + event.modifiers + "\n" + "type : " + event.type + "\n"
        + "screenX : " + event.screenX + "\n" + "screenY : " + event.screenY + "\n"
        + "pageX : " + event.pageX + "\n" + "pageY : " + event.pageY;
}
}
</SCRIPT></HEAD>

```

```

<BODY><H1> Dragging and Dropping </H1><HR>

```

```

<FORM Name=frmText><CENTER>

```

```

    <TEXTAREA Name=textareaEventInfo Rows=9 Cols=35></TEXTAREA>

```

```

</CENTER></FORM>

```

```

<LAYER ID="layerPhoneDnD" Top=350 Left=125 Height=50 Width=50>

```

```

    <IMG Src="images/javacup.gif"></LAYER>

```

```

<LAYER ID="layerBallDnD" Top=350 Left=275 Height=50 Width=50>

```

```

    <IMG Src="images/computer.gif" Width=50 Height=50></LAYER>

```

```

<LAYER ID="layerBookDnD" Top=350 Left=425 Height=50 Width=50>

```

```

    <IMG Src="images/corp.gif"></LAYER>

```

```

</BODY>

```

```

</HTML>

```



## TO MOVE FORWARD

In the topics discussed so far, information flows only in one direction from the web server to the browser. Web pages can be created with well-formatted page contents, which would attract visitors to the web site. This would still be incomplete if a Web site cannot process the feedback form and store the information in a Database or some text file for some future reference. Feedback on a web site leads to constant improvement of the site. Based on the data stored, dynamic Web pages can be created by reading the data that is stored in the Database, which also enhances the Web site.

The next chapters describes how to create interactive web pages that obtain information from viewers through forms, validate this using JavaScript and then store it in Database or a File using Server side code. The Server side code can be developed using languages like PERL, ASP, JavaScript, Java and so on. Further chapters are going to take a long interesting look at Common Gateway Interface (CGI) scripting using PERL.

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. \_\_\_\_\_ are powerful mechanism for adding styles (e.g. fonts, colors, spacing etc) to web documents.
2. The advantages of style sheets include the ability to make \_\_\_\_\_ changes to all documents from a single location.
3. The Style assignment process can be accomplished with the \_\_\_\_\_ and \_\_\_\_\_ tag.
4. The <STYLE> tag should be within the \_\_\_\_\_ and \_\_\_\_\_ element.
5. \_\_\_\_\_ attribute sets an element's text-color.
6. \_\_\_\_\_ attribute aligns the text within an element to the right.
7. In the body of a document, \_\_\_\_\_ and \_\_\_\_\_ tags are used to set the boundaries of the rule's styling specifications.
8. \_\_\_\_\_ attribute is used to set the color of the border.
9. External Style Sheets can be saved as file using \_\_\_\_\_ extensions, which can be linked to web pages by the \_\_\_\_\_ tag.
10. Style Sheets support \_\_\_\_\_, which are sets of style changes for a document.
11. \_\_\_\_\_ tag is used to create an Inflow layer.
12. The layer's visibility can be controlled using the \_\_\_\_\_ attribute.

### TRUE OR FALSE

13. BACKGROUND-REPEAT: repeat-x will repeat the image specified vertically.
14. Text-indent is used to indent the first line of text encountered.
15. <DIV>...</DIV> are used to divide a web page into segments which can be positioned anywhere on the page.

## HANDS ON EXERCISES

- Design a web page for CYBERSHOP INC, using style sheets with the following specifications:
  - Define a style class 'Maxx' with the following attributes:
 

```
{font-size:120%; color:'green'; font-weight:bold; font-family:cursive}
```
  - Use the defined Style class wherever the text 'CYBERSHOP INC' appears on the web document.
  - Use unordered listing giving the list of services offered by CYBERSHOP INC.
  - Define three segments using <DIV>...</DIV> tags with background colors Blue, Green and Goldenrod positioned accordingly with the given text. Obtain an output as shown in the diagram 12.17.



Diagram 12.17: Output of Hands on Exercise 1.

- Design a web page for CYBERSHOP INC, using layers and layer attributes with the following specifications:
  - Layer 1 will display the Name CYBERSHOP INC which will be visible.
  - Layer 2 will display the services provided by CYBERSHOP INC. This layer will initially not be visible.
  - Layer 3 will contain two buttons Show and Reset, clicking on Show will make Layer 1 disappear and show Layer 2. Clicking on the Reset button will return to back first Layer.
  - Obtain an output as shown in the diagram 12.18.1 and 12.18.2 below.



Diagram 12.18.1: Hands on Exercise 2 screen 1.

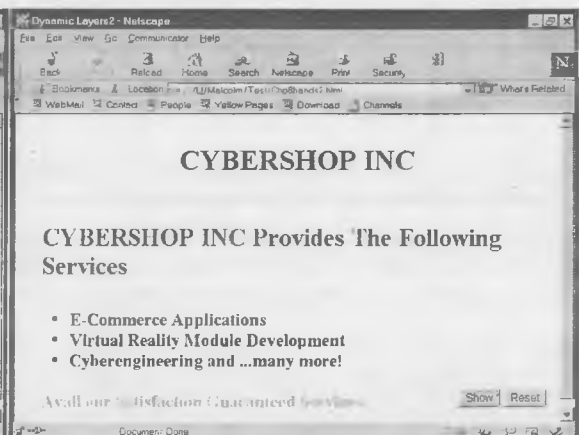


Diagram 12.18.2: Hands on Exercise 2 screen 2.

## ANSWERS TO SELF REVIEW QUESTIONS

### 12. DYNAMIC DHTML

#### FILL IN THE BLANKS

1. Style Sheets
2. Global
3. <STYLE> and </STYLE>
4. <HEAD> and </HEAD>
5. Color
6. Text-Align:Right
7. <SPAN> and </SPAN>
8. Border-color
9. .CSS extension and <LINK> tag
10. Classes
11. <ILAYER>
12. VISIBILITY

#### TRUE OR FALSE

13. False
14. True
15. True

## SOLUTIONS TO HANDS ON EXERCISES

### 12. DYNAMIC DHTML

#### 1. Designing a web page for CYBERSHOP INC using style sheets:

```
<HTML>
  <HEAD><STYLE Type="text/css">
    .Maxx{font-size:120%;color:'green';font-weight:bold;font-family:cursive}
  </STYLE><TITLE>Dynamic Layers 1</TITLE></HEAD>
  <BODY Style="background-color:#ffffcc">
    <H1 Style="text-align:center">Welcome to <A Class="Maxx">CYBERSHOP INC</A><BR>the
      only online Cyber Mall!</H1><HR>
    <P Style = "margin-right:40%">At <A Class="Maxx"> CYBERSHOP INC</A>, we are proud of
      our Clients who avail our services, and every personnel of the incorporation. Here are few the
      services we offer:<UL>
        <LI>E-Commerce Applications</LI>
        <LI>Virtual Reality Module Developments</LI>
        <LI>Cyberengineering</LI>
      </UL></P>
    <A Class="Maxx">CYBERSHOP INC</A>
    Products and services are known virtually all over the world.
    <DIV Style="position:absolute;top:40%;left:65%;background-color:'blue';
      color:'white';width:120;height:110;z-index:0">
      <P Style="font-weight:bold;font-size:20;text-align:center">Virtual Developments</P>
    </DIV>
    <DIV Style="position:absolute;top:60%;left:73%;background-
      color:'green';color:'white';width:120;height:110;z-index:1">
      <P Style="font-weight:bold;font-size:20;text-align:center">Number one since 1994</P>
    </DIV>
    <DIV Style="position:absolute;top:80%;left:81%;background-
      color:'goldenrod';color:'white';width:120;height:90;z-index:2">
      <P style="font-weight:bold;font-size:20;text-align:center">The Cyber Mall on the WEB</P>
    </DIV>
  </BODY>
</HTML>
```

#### 2. Designing a web page for CYBERSHOP INC, using layers and layer attributes:

```
<HTML>
  <HEAD><TITLE>Dynamic Layers 2</TITLE></HEAD>
  <BODY BgColor="Lavender">
    <LAYER Name=layerHeader Left=5% Top=10% Visibility="show">
      <CENTER><H1>CYBERSHOP INC</H1></CENTER><HR>
    </LAYER>
```

```
<LAYER Name=layerMessage Left=5% Top=10% Visibility="hide">
  <CENTER><H1 Style="color:'Crimson' ">CYBERSHOP INC</H1></CENTER><HR>
  <P Style="color:'Blue';font-size:35px;font-weight:bold">CYBERSHOP INC Provides The
    Following Services<UL Style="color:'Blue';font-size:25px;font-weight:bold">
      <LI>E-Commerce Applications
      <LI>Virtual Reality Module Development
      <LI>Cyberengineering and ... many more!
    </UL></P>
  <P Style="color:'Coral';font-family:cursive;font-weight:bold;font-size:15pt">Avail our
    Satisfaction Guaranteed Services.</P>
</LAYER>
<LAYER Name=layerButton Left=80% Top=90%><FORM>
  <INPUT onClick="layerMessage.visibility='show';layerHeader.visibility='hide'"
    Type=Button Value="Show" >
  <INPUT onClick="layerMessage.visibility='hide';layerHeader.visibility='show'"
    Type=Button Value="Reset">
</FORM></LAYER>
</BODY>
</HTML>
```

## SECTION - IV: Understanding The PERL Language

### 13. COMMON GATEWAY INTERFACE CONCEPTS

#### WHAT IS THE COMMON GATEWAY INTERFACE?

The Common Gateway Interface (CGI) is a specification defined by the World Wide Web Consortium (W3C), defining how a program interacts with a Hyper Text Transfer Protocol (HTTP) server. The Common Gateway Interface (CGI) provides the middleware between WWW servers and external databases and information sources. CGI applications perform specific information processing, retrieval, and formatting tasks on behalf of WWW servers.

#### Why is CGI used?

An interesting aspect of a CGI enabled Web server is that computer programs can be created and deployed that can accept user input and create a Web page on the fly. Unlike static Web pages that display some preset information, these *interactive* web pages enable a client to send information to the Web server and get back a response that depends on the input.

A Web search engine is a good example of an interactive web page. The client enters one or more keywords, and the Web index returns a list of Web pages that satisfy the search criteria entered. The Web page returned by the Web index is also *dynamic*, because the content of that page depends on what the client types in as search words – it's not a predefined static document.

To create an interactive Web page, HTML elements are used to display a form that accepts a client's input and passes this to special computer programs on the Web server. These computer programs process a client's input and return requested information, usually in the form of a web page constructed on the fly by the computer program. These programs are known as gateways because they typically act as a medium between the Web server and an external source of information, such as a database. Gateway programs exchange information with the Web server using a standard known as *The Common Gateway Interface*. This is the reason *CGI programming* is used to describe the task of writing computer programs that handle client requests for information.

The term *gateway* describes the relationship between the WWW server and external applications that handle data access and manipulation chores on its behalf. A gateway interface handles information requests in an orderly fashion, and then returns an appropriate response. For example, an HTML document generated on the fly, which contains the results of a query, applied against an external database.

In other words, CGI allows a WWW server to provide information to WWW clients that would otherwise not be available to those clients. This could, for example, allow a WWW client to issue a query to a database and receive an appropriate response in the form of a custom built Web document.

Some common uses of CGI include:

- Gathering user feedback about a product line through an HTML form
- Querying a database and rendering the result as an HTML document

## CGI – HOW IT WORKS

As shown in diagram 13.1, a Web browser running on a client machine exchanges information with a Web server using the Hyper Text Transfer Protocol or HTTP. The Web server and the CGI program, normally run on the same computer, on which the web server resides. Depending on the type of request from the browser, the web server either provides a document from a directory in the file system or executes a CGI program.

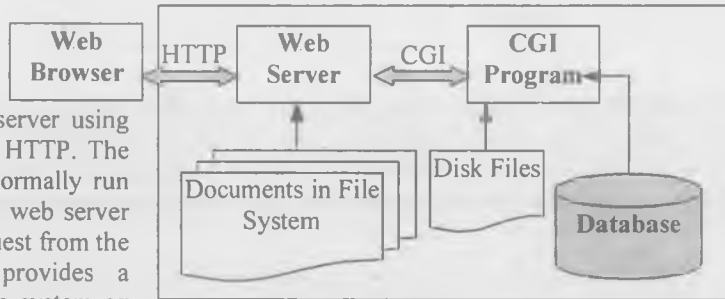


Diagram 13.1

The purpose of the CGI program (or CGI script as it is often called) is the creation of dynamic HTML on demand from a client browser. The sequence of events for creating a dynamic HTML document on the fly through CGI scripting is as follows:

- ❑ A client makes an HTTP request by means of a URL. This URL could be typed into the **Location** window of a browser, be a hyperlink or be specified in the **Action** attribute of an HTML `<form>` tag
- ❑ From the URL, the Web server determines that it should activate the CGI script referenced in the URL and send any parameters passed via the URL to that script
- ❑ The CGI script processes the parameters passed, then based on these parameters, returns HTML to the Web server. The web server, in turn, adds a MIME header and returns the HTML text to the Web browser. Hence fulfilling the concept of dynamic HTML being returned on demand from a Web browser
- ❑ The Web browser then renders and displays the HTML document received from the Web server

### How information is transferred from the Web Browser to a CGI program

The Web browser uses value coded with the **method** attribute of the `<form>` tag to determine how to send the HTML form's data to the Web server. There are two values that can be passed to the method attribute, they are:

Value	What the Web browser does
GET	The Web browser submits the HTML form's data <b>encoded into</b> the URL being dispatched to the Web Server
POST	The Web browser opens a text data stream and submits the HTMLS form's data to the Web Server

### The GET Method

The GET method is so called because the browser uses the HTTP GET command **to submit the data** to the Web Server. The GET command sends a URL to the Web server. If the HTML form's data is sent to the Web server using the HTTP GET command, the browser must encode all the forms data into the URL.

The key features of the GET method of **data submission** are as follows:

- ❑ The values of all the fields are concatenated and passed to the URL specified in the **action** attribute of the `<form>` tag. Each field's values appear in the **name-value** format.
- ❑ Any character with a special meaning in the form's data is encoded using a special encoding scheme commonly referred to as URL encoding. In this encoding scheme a **space** is replaced by a **plus sign** (+), **fields** are separated by an ampersand (&), and any **non-alphanumeric character** is replaced by a **%xx code** (where xx is a hexadecimal representation of the character).

Since data is returned to the Web server using an encoded URL the amount of data that can be streamed to the Web Server is limited to only 2Kbytes

## The POST Method

In the POST method of data submission, the Web browser uses the POST command to submit the data to the server and includes the form's data in the body of that command. The POST method can handle **any amount of data**, because the browser sends the data as a separate text data stream to the Web Server. The POST method **must** be used to send potentially large amounts of data to a Web server.

*Tip*



It is probably a good idea to standardize on the POST method to submit data to a Web Server. This will make Server side CGI coding a lot simpler.

## How a CGI URL is interpreted by the Web Server

The Web server must be configured to recognize an HTTP request for a CGI program. In short, configuring the web server involves informing it of the directory where the CGI programs reside. The URL specifying a CGI program looks like any other URL, but the Web server is intelligent enough to examine the directory name and determine whether the URL references a normal HTML document or a CGI program.

The Web server expects the CGI program's name to appear immediately following the CGI directory (e.g. /cgi-bin/).

A URL can also include **additional path information**, which can be used by the CGI program. This path information needs to be included in the URL immediately following the CGI program name.

## Environment Variables

In the case of CGI, environment variables are known to the server. CGI is used to pass data about an HTTP request from the Web server to a specific CGI program. These variables are accessible to both the Web server and any CGI program invoked. These variables may also be set or assigned their values when the web server actually executes a CGI program.

To communicate with the CGI program, the Web Server sets up a number of environment variables with useful information. e.g. the REQUEST\_METHOD environment variable indicates the **data submission method** used by the browser - whether it was GET or POST.

Thus, environment variables provide a convenient mechanism to transfer information to a CGI program received from a browser. The requirement of a specific Web server controls how CGI programs access variables. If a variable does not have a value, this indicates a zero-length value (NULL). If the variable is not defined in the server's systems, it is assumed to have a zero-length value and hence, it is also assumed to be NULL. A list of environment variables is described in table 13.1 below.

Variable Name	What it is used for
AUTH_TYPE	Access Authentication Type
CONTENT_LENGTH	Size in decimal number of octets of any attached entity
CONTENT_TYPE	The MIME type of an attached entity
GATEWAY_INTERFACE	Server's CGI spec version
HTTP (string)	Client header data
PATH_INFO	Path to be interpreted by CGI application
PATH_TRANSLATED	Virtual to physical mapping of file in system
QUERY_STRING	URL-encoded search string
REMOTE_ADDR	IP address of agent making request
REMOTE_HOST	Fully qualified domain name of agent making request

Table 13.1



Variable Name	What it is used for
REMOTE_IDENT	Identify data reported about agent connection to server
REMOTE_USER	User ID sent by client
REQUEST_METHOD	Request method by client
SCRIPT_NAME	URL path identifying a CGI application
SERVER_NAME	Server name; host part of URL; DNS alias
SERVER_PORT	Server port where request was received
SERVER_PROTOCOL	Name and version of request protocol
SERVER_SOFTWARE	Name and version of server software

Table 13.1 (Continued)

## How A CGI Program Returns Information To The Server

Regardless of how a Web server passes information to the CGI program, the CGI program **always returns information** to the Web server by writing to **standard output**. In other words, if a CGI program wants to return an HTML document, the program must write that document to standard output. The Web server then processes that output and sends the data back to the browser that had originally submitted the request. The CGI program adds **appropriate header information** to its output and sends this to the Web server so that the Web sever knows what kind of data its streaming back to a browser.

### Note



Standard Output is a default device, to which a program sends its output. The environment in which a program is being run defines standard Output.

For example, the standard output device for a PERL program running in Windows is the command window. Here it is the O/s (i.e. Windows) that determines Standard Output.

In the case of a **CGI program**, the standard output device is the Web Server software running in the computer's memory. This is because the PERL program requires a PERL environment to run in, and PERL determines Standard Output.

Hence the output of a CGI program is always passed to a Web Server, which is active in the computer's memory. Without a Web server being active in the computers memory a CGI program will not run.

## Processing HTML Form Information In A CGI Program

A CGI program, (written in PERL) needs to be able to access data returned by the browser, then process it some way before generating any meaningful output. When a browser submits data via the **GET method**, the CGI program obtains its information through the **QUERY\_STRING** environment variable.

If the browser submits data via the **POST method**, the CGI program obtains information through **Standard Input**.

The basic steps followed by a CGI program designed to handle data sent in by a browser either by the GET or POST methods are:

- Check the **REQUEST\_METHOD** environment variable to determine whether the request is GET or POST
- If the method is GET, use the value of the **QUERY\_STRING** environment variable as the input. Also, check for any path information in the **PATH\_INFO** environment variable
- If the method is POST, get the length of the input (in number of bytes) from the **CONTENT\_LENGTH** environment variable. Then read that many bytes from **Standard Input**
- Extract the **name-value** pairs for various fields by splitting the input data at the ampersand (&) character, which separates the name from the value

- In each **name-value** pair convert all + signs to spaces
- In each **name-value** pair, convert all %xx sequences to ASCII characters (xx, denotes a pair of hexadecimal digits)
- Save the **name-value** pairs of specific fields for use later

### What Is A CGI Program?

A CGI program is a computer program that is started and run by a Web server in response to an HTTP request.

A CGI program is generally used to process data submitted to the Web server by a browser. The HTML **<FORM>** tag's **action** attribute specifies the name of the CGI program (including the TCP/ip address server where the program resides).

For example, the following **<FORM>** tag specifies a program named **query.cgi** in the **/cgi-bin** directory on the Web server **digital.com**

```
<FORM method = GET action = http://www.digital.com/cgi-bin/query.cgi>
```

### PROGRAMMING LANGUAGES

CGI programs have been developed in C, C++, Visual Basic Script (VB Script), PERL, TCL, REXX, Python, Icon, AppleScript, Unix shell script, and even Dos Batch files.

### Why PERL For CGI?

A CGI program can be written in any of the above-mentioned languages, but PERL is especially suited for this because PERL programs are easy to learn and write. PERL has great text-processing capabilities (CGI programs have to process the URL-enclosed text data and print HTML text to standard output), which is why it was a natural choice for some of the first CGI sample programs provided by NCSA.

PERL can often accomplish the same task as a C or C++ program with far fewer lines of code, it has become the most widely used option for custom CGI scripts.

Besides this PERL is a scripting language, which means it does not have to be compiled. Instead, an interpreter executes the PERL script, this makes it easy to write and test PERL scripts, because they do not have to go through the typical **edit-compile-link** cycle or compiler based programs

### SELF REVIEW QUESTIONS

#### FILL IN THE BLANKS

1. CGI stands for \_\_\_\_\_.
2. A \_\_\_\_\_ acts as a conduit between a Web Server and an external source of information.
3. HTTP stands for \_\_\_\_\_.
4. The \_\_\_\_\_ method of submission submits the form data as part of a URL.
5. \_\_\_\_\_ are used to pass data about an HTTP request from a server to the CGI programs.
6. When a form submits data via the GET method the CGI program receives information through the \_\_\_\_\_ environment variable.
7. The \_\_\_\_\_ environment variable specifies the MIME type of data being sent to the CGI program.

#### TRUE OR FALSE

8. CGI programs always return information to the server by writing to standard output.
9. The REMOTE\_HOST environment variable indicates the data submission method.
10. PERL programs need to go through the edit-compile-link cycles before they are executed.

## 14. THE PERL LANGUAGE

### AN INTRODUCTION

**PERL** (Practical Extraction Report Language) was originally created to extract information from text files and then use that information to prepare reports. PERL is a scripting language, which means the programmer does not have to compile and link a PERL script. Instead, a PERL **interpreter** executes the PERL script.

PERL's popularity can be attributed to many things, one of which is the ease with which programs can be built. Because PERL is interpreted, it greatly reduces development time. This allows the programmer to build larger and more complex programs in a shorter period of time when compared to compiler based program creation.

Unfortunately, some programmers do not give themselves enough time to get to know the language, which leads to inferior and inefficient code. The more complex the programs become, the more important it is to understand the syntax and semantics of the language. An intimate knowledge of the syntax and semantics of any language empowers a programmer to write programs that run efficiently and consume the least CPU resources to get the job done.

### INSTALLING AND SETTING UP PERL

Download the latest stable version of PERL from <http://www.activestate.com/Products/ActivePerl/>

The file chosen for download was **ActivePerl-5.8.4.810-MSWin32-x86**.

#### Starting The Install Process OF PERL

Follow these steps to install PERL on a windows box:

1. Execute the downloaded file by double clicking it
2. A screen as seen in diagram 14.1 pops up
3. Click Next to proceed. This will bring up the next screen as seen in diagram 14.2

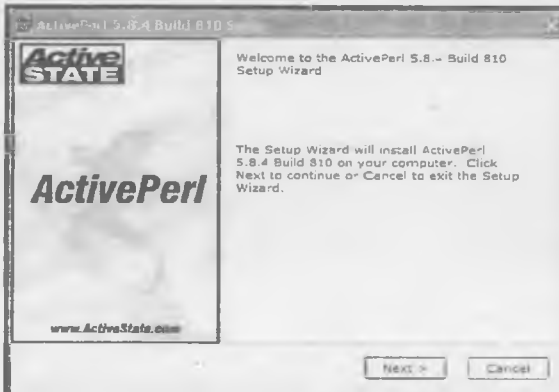


Diagram 14.1: Welcome Screen Of ActivePerl Install



Diagram 14.2: End-User License Agreement

4. Accept the End-User License Agreement and click **Next**

5. This will proceed to the screen as seen in diagram 14.3
6. Keep all the selections i.e. let all the products be installed. Click **Next**
7. This will display a screen as seen in diagram 14.4. Since PPM3 support is not required. Don't enable it. Just click **Next**

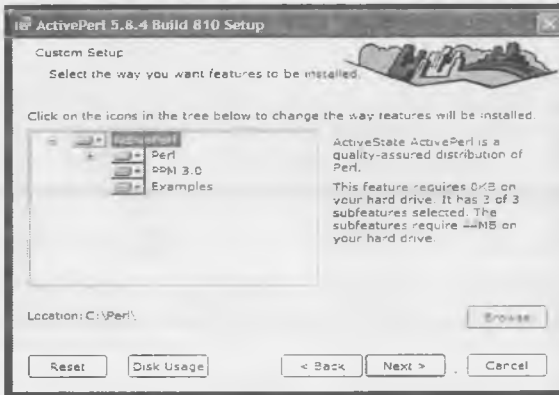


Diagram 14.3: The Custom Setup

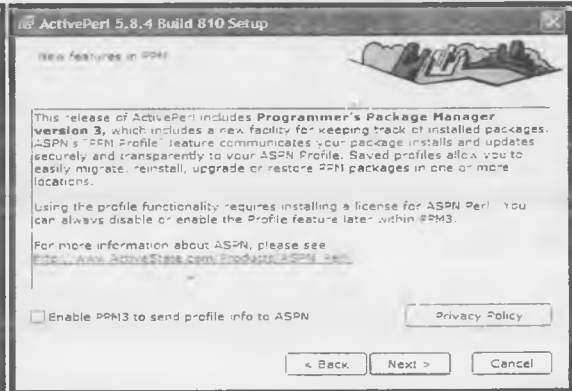


Diagram 14.4: The PPM3 support

This release of ActivePerl includes **Programmer's Package Manager version 3**, which includes a new facility for keeping track of installed packages. ASPN's "PPM Profile" feature communicates the package installs and updates securely and transparently to the ASPN Profile. Saved profiles allow the user to easily migrate, reinstall, upgrade or restore PPM packages in one or more locations.

8. This will now proceed to the setup options available as seen in diagram 14.5 such as:
  - a. Add Perl to the path environment variable
  - b. Create Perl file extension association

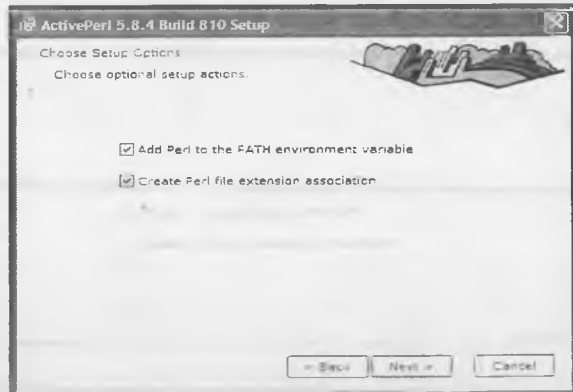


Diagram 14.5: The setup options

The last two will only appear if Microsoft's IIS is installed.

9. Select both the options and click **Next**
10. This will finally display the Ready To Install screen as seen in diagram 6.6. Click **Install** to start the actual install process

The Install process progresses as shown in diagram 14.7

11. Click **Finish** to complete the Perl Installation

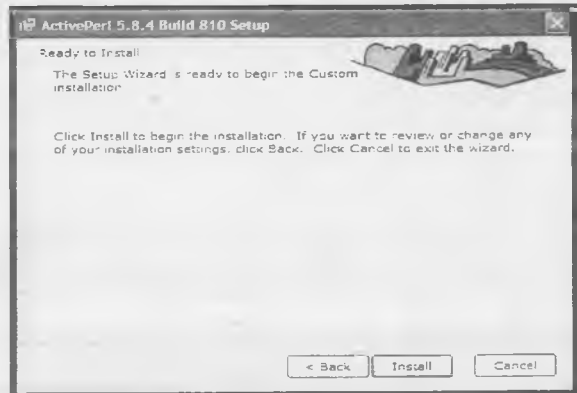


Diagram 14.6: The Ready to Install screen



Diagram 14.7: The Install process in progress

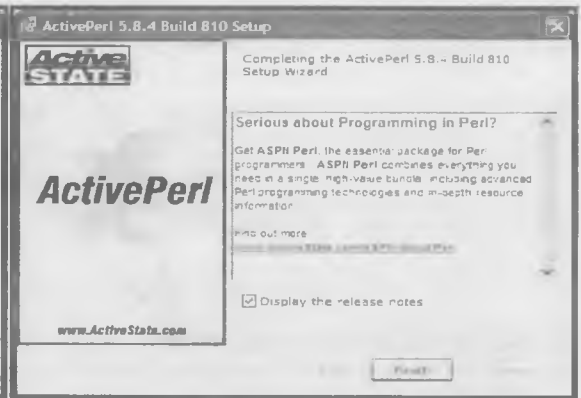


Diagram 14.8: The Install process completed

## Binding The PERL Installed With Apache2

The only thing left to do is to edit Apache's `httpd.conf` file and make an entry for the `cgi-bin` directory in the `ScriptAlias` tag of the virtual host created earlier as:

```
ServerName 172.16.9.66 (Here enter the actual IP of the host computer this IP is the one we used)
NameVirtualHost 172.16.9.66
<VirtualHost 172.16.9.66>
    ServerAdmin webmaster@sct.com
    DocumentRoot c:\sct\perltraining
    ServerName sct.perltraining.com
    ScriptAlias /cgi-bin/ "c:\sct\perltraining\cgi-bin\"
</VirtualHost>
```

## Registering The Changes Made In The httpd.conf With Apache2

After making any changes to the `httpd.conf` ensure that Apache is restarted to apply the new changes. This can be done by using the icon on the task bar: (Refer Diagram 14.9)



Diagram 14.9: Restarting Apache2

## Testing the PERL Setup

To test whether Perl has been successfully setup and integrated with Apache2 create a simple script named `testperl.pl` that contains the following code:

```
#!c:/perl/bin/perl.exe
# Print the Header
print "Content-type: text/html\n\n";
# Printing HTML Page
print "<HTML><HEAD>";
print "<TITLE>Testing PERL Setup and Integration With Apache2</TITLE></HEAD>";
print "<BODY BGColor=FFFFFF><TABLE Width='100%' BGColor=#ffcb32><TR><TD><H2>Hello
    World!</H2></TD></TR></TABLE>";
print "</BODY></HTML>";
exit(0);
```

Place this file under a directory named `cgi-bin` in the Web server's *virtual domain* created earlier under `c:\sct\perltraining` directory. Examine the output of this script in a Web browser by pointing to `http://sct.perltraining.com/cgi-bin/testperl.pl`. If Perl setup was successful then a screen similar to that shown in diagram 14.10.

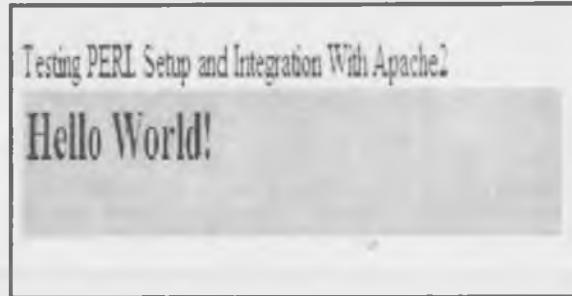


Diagram 14.10: The `testperl.pl` file in I.E.

## Changes in the `httpd.conf` File For The Framework

The following entries should be exists in the `httpd.conf` file available under `C:\Program Files\Apache Group\Apache2\conf` directory:

```
ServerName 172.16.9.66 (Here enter the actual IP of the host computer this IP is the one we used)
NameVirtualHost 172.16.9.66
<VirtualHost 172.16.9.66>
    ServerAdmin webmaster@sct.com
    DocumentRoot c:\sct\perltraining
    ServerName sct.perltraining.com
    ScriptAlias /cgi-bin/ "c:\sct\perltraining\cgi-bin\"
</VirtualHost>
```

## PERL BASICS

Like any computer program, a PERL program starts out as a file containing commands or statements - in this case, they are PERL commands or statements.

PERL is:

- Case sensitive
- Free form; there are no constraints on the placement of any keyword. There are no rules for the exact number of spaces or lines
- A semi-colon must terminate each PERL statement (;)
- Comments begin with a hash mark (#)
- A group of PERL statements can be treated as a block by enclosing them in curly braces ({...})

## PERL STRINGS

PERL has three types of strings, namely:

1. Double Quoted Strings
2. Single Quoted Strings
3. Back Quoted Strings

Each of these strings behave differently from the other and provide special functionality to the PERL language.

### Double Quoted Strings

Double-quoted strings are commonly used because of the following features:

- Scalar and array variables embedded in double-quoted strings are **replaced by their values**. Thus, PERL will treat the string `"Total = $total"` as `"Total = 10"` where `$total` is assumed to have a value `10`.
- In double-quoted strings, characters with a **backslash prefix** are **replaced** with a single special character. For example, `\n` becomes a **newline** and `\t` becomes a **tab** character.

**Note**

A single backslash followed by one or more characters is called an escape sequence.

**Single Quoted Strings**

PERL displays a **single quoted** string **without replacing** variable names with their corresponding values. Single quoted strings also **do not recognize** escape sequences. Single quoted strings are **simply treated as ordinary text**.

**Back Quoted Strings**

Strings are enclosed in **backquotes**(```) to display the output of some operating system command.

For example, the following block of code prints the directory listing of the directory from where it is executed.

**Example 1:**

```
#!c:/perl/bin/perl.exe -w
# Print the Header if output is required via a browser
print "Content-type: text/html\n\n";
$dirout = `dir`;
print "Directory Listing:\n";
print "$dirout";
```

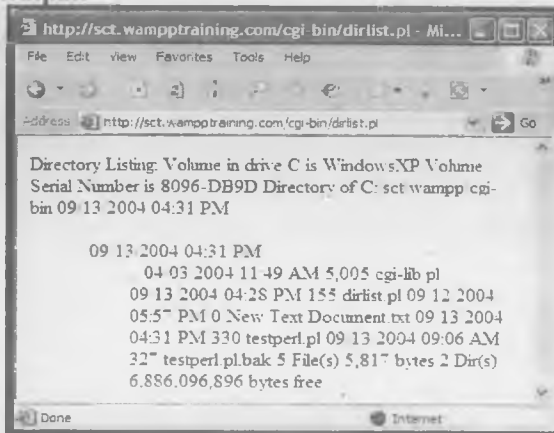
**Output:**

Diagram 14.11.1: The dirlist.pl output in I.E.

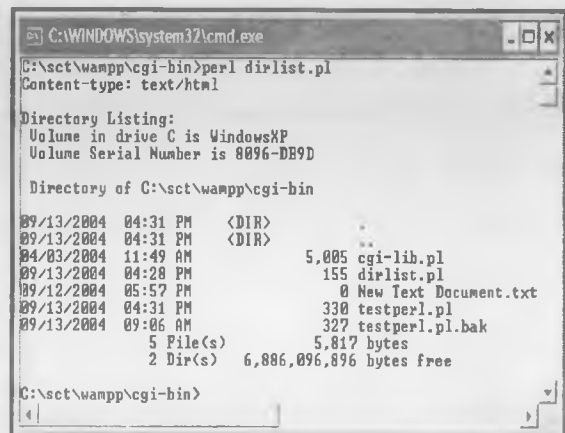


Diagram 14.11.2: The dirlist.pl output in cmd window

The output can be viewed via **command window** or a **web browser** i.e. Internet Explorer as seen in diagram 14.11.1 and diagram 14.11.2. However for the code to provide output in a web browser the content type has to be set. Refer the third line in the Codespec above.

**Tip**

When running any MySQL/mSQL Perl programs, the `-w` command line argument should always be included. With this present, DBI will redirect all MySQL and mSQL specific error messages to STDERR so that you can see any database errors without checking for them explicitly in the program.

## THE NEED FOR DATA STORAGE

In general, the goal of any commercial application program is the manipulation of relevant data. In order to achieve this, the programming environment should be able to store that data for use on demand.

### Variables

In PERL programs data is stored in **variables**. Each variable has a name and can store any type of data. Each variable name in a PERL program begins with a special character:

- A dollar (\$) sign
- A at the rate of (@) sign
- A percent (%) sign

The special character denotes the variable type. The three types of variables are:

- Scalar Variables
- Indexed Array Variables
- Associative Array Variables

### Note



In PERL, there is no data type for a variable. The sex or data type of the variable is dependent on the context in which it is used.

### Scalar Variables

#### What is a Scalar Variable?

**Scalar Variables** are the most basic form of data containers in PERL. A scalar variable can reference a string value or a numeric value. Even though PERL treats strings and numbers with almost the same regard, there is a definite visible difference between the two.

#### Example 2:

```
$num_pages = 256;  
$title = "Good Morning";
```

In the above example, the **string** is enclosed in **double quotes**.

#### Defining Scalar Variables

When a scalar variable is assigned a value, the syntax of the assignment assists the PERL interpreter in deciding the variable type. If the value of the variable is surrounded in single or double quotes, then PERL treats the variable as a string. If there are no quotes, then PERL has to decide if the value is a string or a numeric value.

#### Example 3:

```
$FirstName = Rajesh;  
$MiddleName = 'Rao';  
$LastName = "Guruswamy";  
$Age = "3";
```

If the above script were run, the first scalar variable assignment (`$FirstName = Rajesh`) would give a warning that the bare word `Rajesh` may be a future reserved word like a function name, which will change the context of the assignment if a function named `Rajesh` is ever added to PERL.

The assignment of `$Age` uses quotes, which means that PERL will treat this scalar variable as a string value instead of a numeric value.



## Arrays

For single values, such as the name of an employee or the travel expense of an employee, a scalar variable can be used. Sometimes multiple values (e.g. 30 employee names, 30 expense reports, or 20 e-mail addresses) need to be stored. In such a case an array can be used. An Array is a collection of scalars. e.g. An array of names, or array of marks and so on. In PERL, there are **two** types of arrays – Indexed Arrays & Associative/Hash Arrays.

### Indexed Arrays

PERL has a data structure that is strictly known as an **array** of **scalars**. This structure is more commonly known as a list. PERL's arrays can be used either as a simple list, a stack, or even the skeleton of a complex data structure. These arrays are known as INDEXED ARRAYS and are denoted by the (@) symbol. PERL's array of scalars can be defined by a number of methods.

#### **Creating An Indexed Array**

The example, mentioned below, will create an array named **myList** with no array elements. In other words an empty array.

```
@myList = ( );
```

#### **Populating An Indexed Array**

```
@months = ("Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec");
@months = qw (Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);
```

The above two examples describe a common method of using an array as an indexed list. The values of the array are assigned all at once. Arrays start at **index 0**, so in the above example, \$months[0] will have a value Jan, \$months[1] will have a value Feb and so on.

The **qw** keyword is a shortened form used to extract individual words from a string. In the above case, the individual words are the names of the months, and the result of running **qw** on them is stored in the array **@months**.

If the array elements cannot be assigned all at once, the same can be assigned on an individual basis.

For example, the elements for the **@months** array could be assigned in the following fashion:

```
$months[0] = "Jan";
$months[1] = "Feb";
$months[2] = "Mar";
```

Notice that in the above examples, when the array elements are directly assigned, the \$ character is used and not the @ character. The \$ character at the beginning of an array tells PERL that one individual element of the array, **not** the complete array, is to be assigned.

#### **Extracting Information From An Indexed Array**

The most common method of extracting information from an array is to index the array elements directly.

**Example 4:** (Refer diagram 14.12)

```
#!c:/perl/bin/perl.exe -w
```

```
@months = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);
```

```
for ($x = 0; $x <= $#months; $x++) {
    print "Index[$x] = $months[$x]\n";
```

```
}
```

In the above example, the statement `'@months = qw(Jan Feb Mar Apr May Jun Jul Aug Sep Oct Nov Dec);'` creates an array called `@months` and assigns the names of the months as elements of the array.

The next statement block, is a loop structure which iterates through each and every element of the array, printing the element's index number and its corresponding value. The word `$#months` is actually a PERL convention which returns the value of the largest subscript of the array. If `$#months` returns a value `-1`, the array is empty.

### Hash Arrays

Hash Arrays or Associative Arrays, are arrays that are **indexed by string values** instead of **integer index values**. Associative arrays, unlike scalar arrays, do not have a sense of order, there is **no true first addressable element**. This is because the indexes of the associative array are strings and the information is not stored in a predictable manner.

In PERL, the `%` special character at the head of the variable name identifies an associative array.

### **Creating An Associative Array**

The following example creates an empty associative array named `cities`:

```
%cities = ( );
```

### **Populating An Associative Array**

Just like a normal array of scalars, an associative array can have all its values assigned at once.

#### **Example 5:**

```
%cities = ("Chennai"=>"East", "Nagpur"=>"Central", "Mumbai"=>"West");
```

```
%cities = ("Chennai", "East", "Nagpur", "Central", "Mumbai", "West");
```

As seen in the two examples above, `=>` is equivalent to a comma. **Its main purpose is to create a visual association between pairs.**

Similar to a standard array of scalars, the associative array can be populated by individual elements as well. The above example can be rewritten as:

```
$cities{'Chennai'} = "East";
```

```
$cities{'Nagpur'} = "West";
```

```
$cities {'Mumbai'} = "Central";
```

### **Extracting Information From An Associative Array**

The contents of an associative array can be accessed using one of the following PERL functions:

- keys
- values
- each

#### **The keys Function**

The `keys` function returns a list of the keys of the given associative array when used in a **list context** and the number of keys when used in a **scalar context**.

```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl extractarray.pl
Content-type: text/html

Index[0] = Jan
Index[1] = Feb
Index[2] = Mar
Index[3] = Apr
Index[4] = May
Index[5] = Jun
Index[6] = Jul
Index[7] = Aug
Index[8] = Sep
Index[9] = Oct
Index[10] = Nov
Index[11] = Dec

C:\sct\wampp\cgi-bin>
```

**Diagram 14.12:** The `extractarray.pl` output in command window

```
@keyList=keys %hash
```

OR

```
@keyList=keys(%hash)
```

Since the keys function returns a list that can be stored into an array for later use, such as printing. For example, the statement '**print @keyList;**' will print the array in list format.

Usually, the list returned by the keys function is processed in some type of loop and never stored in array.

The following example gives an idea of retrieving the keys value by scalar Context:

```
foreach $index (keys(%myhash)) {
    print "The keys value is $myhash{$index} \n";
}
```

### Note



**Scalar Context** means that a function is being called and the return value of the function is a scalar variable.

**Array Context** means that a function is being called and the return value of the function is an array.

The keys returned from the **keys** function are merely indexes in the associative array. The **keys** function is the most common function used to extract information from an associative array.

**Example 6:** (Refer diagram 14.13)

```
#!/c:/perl/bin/perl.exe -w
%cities=("Chennai"=>"East", "Nagpur"=>"Central",
        "Mumbai"=>"West");
for $key(keys %cities) {
    print "Key = $key Value = $cities{$key}\n";
}
```

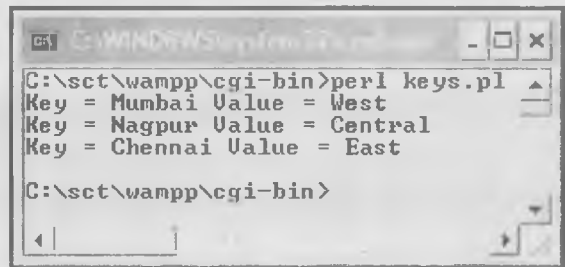


Diagram 14.13: The keys.pl output in command window

In the above example, the first statement, creates an associative array named **%cities** and initializes it with three pairs of elements, namely:

Chennai=>East, Nagpur=>Central and Mumbai=>West

The next block of code uses the **keys** function on the associative array **%cities**. Each of the keys in the **%cities** array is stored in the **scalar** variable **\$key**. By using the value contained in this variable, each and every key and its corresponding value in the array are printed by the **print()**.

### The values Function

In a scenario where the keys of an associative array are not required, the **values** function can be used to directly access the values in an associative array.

**Example 7:** (Refer diagram 14.14)

```
#!/c:/perl/bin/perl.exe -w
%cities=("Chennai"=>"East", "Nagpur"=>"Central",
        "Mumbai"=>"West");
for $value(values %cities)
{ print "Value = $value\n"; }
```

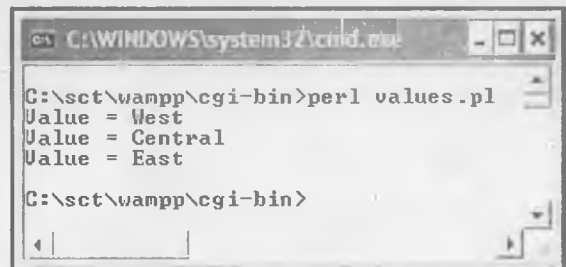


Diagram 14.14: The values.pl output in command window

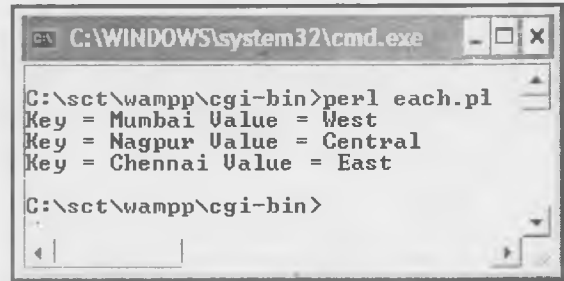
### The each Function

The **each** function returns a key-value pair from the associative array.

**Example 8:** (Refer diagram 14.15)

```
#!/c:/perl/bin/perl.exe -w
```

```
%cities = ("Chennai"=>"East", "Nagpur"=>"Central",
           "Mumbai"=>"West");
while (($key,$value) = each %cities) {
    print "Key = $key Value = $value \n";
}
```



```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl each.pl
Key = Mumbai Value = West
Key = Nagpur Value = Central
Key = Chennai Value = East
C:\sct\wampp\cgi-bin>
```

Diagram 14.15: The each.pl output in command window

**Example 9:** (Refer diagram 14.16)

To accept a name from the user and give a **Welcome message** if the name is in the array.

```
#!/c:/perl/bin/perl.exe -w
```

```
%names=("Name1"=>"Chhaya",
        "Name2"=>"Vaishali",
        "Name3"=>"Sharanam");
```

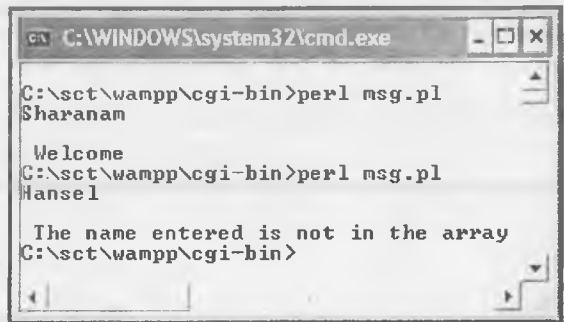
```
$new=<STDIN>;
```

```
chomp $new;
```

```
$a=0;
```

```
$x=0;
```

```
foreach $ni(keys%names) {
    $a=$a+1;
    if($new eq $names{$ni}) {
        $x=1;
        print "\n Welcome";
    }
    elsif (($x!=1) && ($a==3)) {
        print "\n The name entered is not in the array";
    }
}
```



```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl msg.pl
Sharanam

Welcome
C:\sct\wampp\cgi-bin>perl msg.pl
Hansel

The name entered is not in the array
C:\sct\wampp\cgi-bin>
```

Diagram 14.16: The msg.pl output in command window

## ENVIRONMENT VARIABLES AND THE %ENV SPECIAL HASH ARRAY

Environment variables are entities that exist within a specific computer's environment. Many of these variables attain their values whenever a user logs onto a computer.

Because environment variables persist even as execution threads come and go, environment variables sometimes function as placeholders, to pass data from one application to another **within the same user session**.

The environment variables are common in operating systems such as UNIX, Linux, Windows NT, and so on and fit the model of an associative array. An environment variable has a **name** and a **value**. Because each environment variable is a **name-value** pair, these can be easily stored in an associative array with the **name** of the variable as the **key**.

PERL automatically defines the %ENV associative array, which holds all the currently defined environment variables. Use the environment variable **name** as an index to access any environment variable from % ENV.

**Example 10:** (Refer diagram 14.17)

```
#!/c:/perl/bin/perl.exe -w
foreach $name (keys %ENV) {
    $value = $ENV{$name};
    print "$name = $value\n";
}
```

In the above example, the `keys` function accepts the `%ENV` associative array as an argument and returns all the keys in that array. Each key from `%ENV` returned by the `keys` function is stored in `$name`. `$name` is then used to pick up the associated value of that key in the `%ENV` associative array.

```
C:\WINDOWS\system32\cmd.exe
C:\set>perl c:\perl\bin\envvars.pl
USERPROFILE = C:\Documents and Settings\Sharanam Shah
HOMEDRIVE = C:
TEMP = I:\Temp
SYSTEMDRIVE = C:
PROCESSOR_ARCHITECTURE = x86
SYSTEMROOT = C:\WINDOWS
COMMONPROGRAMFILES = C:\Program Files\Common Files
COMSPEC = C:\WINDOWS\system32\cmd.exe
SESSIONNAME = Console
LOGONSERVER = \\SHARANAMS
APPDATA = C:\Documents and Settings\Sharanam Shah\Application Data
WINDIR = C:\WINDOWS
PROGRAMFILES = C:\Program Files
OS = Windows_NT
PROCESSOR_LEVEL = 6
PATH = COM;.EXE;.BAT;.CMD;.VBS;.UBE;.JS;.JSE;.ASP;.ASH
USERNAME = Sharanam Shah
PROMPT = $PS
NUMBER_OF_PROCESSORS = 1
APP_NO_HOST_CHECK = NO
HOMEPath = \Documents and Settings\Sharanam Shah
PATH = C:\Perl\bin;C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\msql\bin
PROCESSOR_IDENTIFIER = x86 Family 6 Model 11 Stepping 1, GenuineIntel
US Endpwin = SHARANAMS
COMPUTERNAME = SHARANAMS
ALLUSERSPROFILE = C:\Documents and Settings\All Users
PROCESSOR_ARCHITECTURE = x86
TEMP = I:\Temp

C:\set>perl c:\perl\bin>
```

**Diagram 14.17:** The `envvars.pl` output in command window

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. PERL stands for \_\_\_\_\_.
2. Scalar and array variables embedded in \_\_\_\_\_ strings are replaced by their values.
3. An array is a collection of \_\_\_\_\_.
4. \_\_\_\_\_ are indexed by string values.
5. The \_\_\_\_\_ function is the most common function used to extract information from a hash array.
6. Environment variables are stored in the array \_\_\_\_\_.

### TRUE OR FALSE

7. Perl is not case sensitive.
8. In double-quoted strings, characters with a backslash prefix are replaced with a single special character.
9. Indexed arrays are denoted by the symbol %.
10. `$#array` returns a value `-1` if the array is not empty.

## HANDS ON EXERCISES

1. Assign string values and numeric values to scalar variables.
2. Create an indexed array `@week` and populate it with the days of the week. Print the elements of the array `@week`.
3. Create an associative array `%expenses` for the following persons: John, Mary, Ed & Jane with their expenses respectively as \$250.50, \$195.00, \$345.25 and \$225.99  
Print the name and the expenses of each element of the `%expenses` array with the help of the `keys` function using: a) a for loop b) a foreach loop c) a while loop
4. Create an indexed array `@names` by assigning individual elements of that array as follows:  
`$names[0]="Name1";`  
Populate the array with 10 elements. Then print the value of each element using a `for` loop.

## 15. PERFORMING OPERATIONS AND CONTROLLING PROGRAM FLOW

Commercial Applications revolve around giving business manager information, which has **integrity** so that informed business decisions can be made. This involves capturing data and storing this data in temporary storage. Once done, this data is **processed** to ensure its **integrity**. Often done by the application of business rules to the data in temporary storage, then moving this data to permanent storage if it passes the business rules of the application.

The processing of this data could involve formulae, arithmetic operators as well as dealing with and manipulating text. Manipulating text may mean comparing strings, concatenation and so on. This chapter deals with various ways to process numerical and text data in PERL.

### BASIC ARITHMETIC OPERATIONS

Operator	Name	Example
+	Addition	<code>\$amount = \$price + \$sales_tax;</code>
-	Subtraction	<code>\$over_payment = \$payment - \$balance;</code>
*	Multiplication	<code>\$sales_tax = \$price * \$tax_rate;</code>
/	Division	<code>\$average = \$total_amount / \$number_of_items;</code>

#### Example 1:

```
$price = 66.80;
$sales_tax = $price * 0.05;
$amount = $price + $sales_tax;
$payment = 100;
$change = $payment - $amount;
```

### Note



When an arithmetic operation is performed and the result is a real number, that number is printed with 4 digits after the decimal point instead of the regular 2 digits. To print a value with a specified number of digits after the decimal point, the PERL **printf** function can be used.

### Auto-Increment And Auto-Decrement Operators

Incrementing or reducing the value of a variable by 1 is a common operation required in many commercial applications. PERL provides a shortcut to perform these tasks.

- The **Auto-Increment** operator denoted by double plus sign (++) increments the value of a variable by 1.
- The **Auto-Decrement** operator denoted by double minus sign (--) decrements the value of a variable by 1.

Post-Increment	The value of the variable is incremented <b>after</b> the variable has been used in the current expression	n++
Pre-Increment	The value of the variable is incremented <b>before</b> the variable is used in the current expression	++n
Post-Decrement	The value of the variable is decremented <b>after</b> the variable has been used in the current expression	n--
Pre-Decrement	The value of the variable is decremented <b>before</b> the variable is used in the current expression	--n

**Example 2:** (Refer diagram 15.1)

```
#!c:/perl/bin/perl.exe -w
$count = 100;
$count_now = $count++;
print "count_now = $count_now\n Count = $count\n";
$count = 100;
$count_now = ++$count;
print "count_now = $count_now\n Count = $count\n";
```

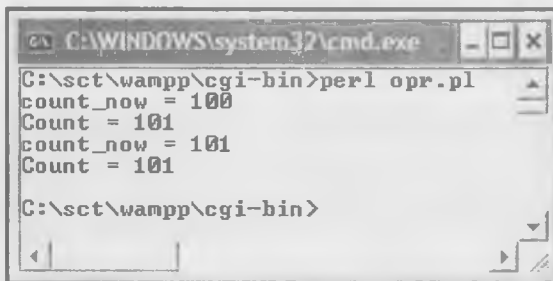


Diagram 15.1: The opr.pl output in command window

## OPERATOR SHORTCUTS

Sometimes, it is necessary to perform an arithmetic operation on the same variable. For example, while

calculating an Accounts Receivable cumulative total, it might be necessary to write the following PERL code:

```
$total = $total + $receivable;
```

PERL provides **shortcut operators** to perform an operation with a variable and then save the result in the same variable. Using these **shortcut operators**, the above code can be implemented in the following manner:

```
$total += $receivable;
```

Following is a summary of commonly used shortcut operators:

Operator	Example	Description
+=	\$x += \$y	Adds \$y to \$x and stores the result in \$x
-=	\$x -= \$y	Subtracts \$y from \$x and stores the result in \$x
*=	\$x *= \$y	Multiplies \$y and \$x and stores the result in \$x
/=	\$x /= \$y	Divides \$x by \$y and stores the result in \$x

## COMPARISON OPERATORS

PERL includes two separate sets of relational operators. One set for comparing numbers and the other for comparing text strings.

### What Is True (Or False) According To PERL?

PERL's numeric and string comparison operators return a **true** or **false** value. Also conditional statements test the value of an expression to see whether it is true or false. Essentially, in a PERL program, any nonzero number is considered true. All text strings are also true. Except for an empty string (defined as "") and the zero string "0", which are false. Any undefined variable is false (a variable is undefined if a value is not assigned to the variable).

### Comparing Numbers and Strings

Conditional statements, such as the **if** statement, checks the value of an expression and executes a block of statements if the expression is true.

Most of the time, the expression being tested in a conditional statement is a comparison that involves a relational operator. For example, an **if** statement might be used to print a message if the number of errors exceeds a preset threshold. Here's how to write such an **if** statement:

```
#!c:/perl/bin/perl.exe -w
$error_count = 30;
if ($error_count > 25) {
    print "Too many errors!\n";
}
```

The expression `$error_count > 25` is an example of a comparison of two numbers, the value of the variable `$error_count` and 25. The greater-than sign (`>`) is one of the relational operators in PERL.

The expression `$error_count > 25` means `$error_count` is greater than 25. The expression is true (that means the expression's value is nonzero), if `$error_count` exceeds 25, otherwise the expression is false (the expression's value is zero).

## Performing Numeric Comparisons

The greater than operator (`>`) is one of the relational operators used to compare numbers in a PERL program. Other numeric comparisons include checking if two numbers are equal or if one number is less than another. The following table lists PERL's relational operators for numeric comparison.

### Relational Operators For Comparing Numbers

Operator	Example	Description
<code>==</code>	<code>\$x == \$y</code>	Equal to operator; value is true if <code>\$x</code> equals <code>\$y</code> , else false.
<code>!=</code>	<code>\$x != \$y</code>	Not equal to operator; value is true if <code>\$x</code> and <code>\$y</code> are unequal, else false.
<code>&lt;</code>	<code>\$x &lt; \$y</code>	Less than operator; value is true if <code>\$x</code> is less than <code>\$y</code> , else false.
<code>&lt;=</code>	<code>\$x &lt;= \$y</code>	Less than or equal to operator; value is true if <code>\$x</code> is less than or equal to <code>\$y</code> , else false.
<code>&gt;</code>	<code>\$x &gt; \$y</code>	Greater than operator; value is true if <code>\$x</code> is greater than <code>\$y</code> , else false.
<code>&gt;=</code>	<code>\$x &gt;= \$y</code>	Greater than or equal to operator; value is true if <code>\$x</code> is greater than or equal to <code>\$y</code> , else false.
<code>&lt;=&gt;</code>	<code>\$x &lt;=&gt; \$y</code>	Three-way comparison operator; value is <code>-1</code> if <code>\$x</code> is less than <code>\$y</code> , <code>0</code> if <code>\$x</code> equals <code>\$y</code> , and <code>1</code> if <code>\$x</code> is greater than <code>\$y</code> .

## Performing String Comparisons

In some programming languages, such as C and C++, the relational operators for comparing numbers are the same ones used to compare text strings. Not so in PERL. PERL has a complete set of relational operators that must be used when comparing strings.

For example, in a PERL program, `==` operator cannot be used to find out whether two strings match. The `==` operator works only with numbers. To test the equality of strings, PERL includes the `eq` operator.

**Example 3:** (Refer diagram 15.2)

```
#!/c:/perl/bin/perl.exe -w
$name = "Sharanam";
if ($name eq "Sharanam") {
    print "$name";
}
```

```
C:\WINDOWS\system32\cmd.exe
C:\sct\wamp\cgi-bin>perl strcmp.pl
Sharanam
C:\sct\wamp\cgi-bin>
```

**Diagram 15.2:** The `strcmp.pl` output in command window

PERL includes other operators that can compare two strings in a variety of ways. The following table summarizes the string comparison operators.

### Relational Operators For Comparing Strings

Operator	Example	Description
<code>eq</code>	<code>\$x eq \$y</code>	Value is true if the strings <code>\$x</code> and <code>\$y</code> are equal, else false.
<code>ne</code>	<code>\$x ne \$y</code>	Value is true if the strings <code>\$x</code> and <code>\$y</code> are not equal, else false.
<code>gt</code>	<code>\$x gt \$y</code>	Value is true if <code>\$x</code> is greater than <code>\$y</code> , else false.
<code>ge</code>	<code>\$x ge \$y</code>	Value is true if <code>\$x</code> is greater than or equal to <code>\$y</code> , else false.
<code>lt</code>	<code>\$x lt \$y</code>	Value is true if <code>\$x</code> is less than <code>\$y</code> , else false.
<code>le</code>	<code>\$x le \$y</code>	Value is true if <code>\$x</code> is less than or equal to <code>\$y</code> , else false.
<code>cmp</code>	<code>\$x cmp \$y</code>	Value is <code>-1</code> if <code>\$x</code> is less than <code>\$y</code> , <code>0</code> if the strings are equal, and <code>1</code> if <code>\$x</code> is greater than <code>\$y</code> .



All string comparisons are done by comparing the numeric value of the characters. For example, the numeric value of uppercase **A** is 65 and that of uppercase **B** is 66. That means, in string comparison, **A** is less than **B**.

## Note



The numeric values of the characters come from the American Standard Code for Information Interchange (ASCII) code. In fact, the numeric value of a character is referred to as the ASCII code of that character.

## CONTROLLING PROGRAM FLOW IN PERL

Flow of execution, or program flow, refers to the sequence in which the PERL interpreter executes PERL script statements. The PERL interpreter always executes the statements sequentially, in the order of their appearance in a PERL program. In most PERL programs, however, the program needs to execute one set of statements if a specific condition is true and, possibly, another set of statements if the condition is false.

Suppose a PERL program is written to process a text file containing expense reports and prints out only those entries that exceed a specified amount. In such a program, one of PERL's **conditional statements**, the **if** statement, might be used, statements that are executed only when certain conditions are true.

### Making Decisions In PERL

A PERL program makes decisions by conditional processing. Based on the value of an expression, the program executes a block of statements that are enclosed in a pair of curly braces ({ . . . }).

#### Syntax:

```
keyword (expression) {      ... block of statements ... }
```

Here, keyword is a special word (such as **if** and **unless**) that denotes the type of decision the program makes. The expression within parentheses is tested before the block of statements within curly braces is executed.

### Using The 'if' Statement

The most common type of conditional statement is the **if** statement. The **if** statement executes a block of statements if a condition is true. The condition is usually an expression that compares numbers or strings.

#### Syntax:

```
if (expression) {          ... statements to execute if expression is true ... ;      }
else {                    ... statements to execute if expression is false ... ; }
```

Suppose a congratulatory message is to be printed if a student's test score is greater than or equal to 90. If the score is less than 90, however, a message needs to be printed urging the student to try harder the next time.

#### Example 4:

```
#!c:/perl/bin/perl.exe -w
# Assume $score has the test score
$score = 95;
if ($score >= 90) {
    print "Congratulations! You got an A in this test.\n";
}
else { print "Work harder for the next test!\n"; }
```

**Output:**

Congratulations! You got an A in this test.

The **if-else** pair enables the handling of both the true and false cases of a test.

**Example 5:**

```
#!/c:/perl/bin/perl.exe -w
```

```
# Assume that $version contains version number $version = 5;
# Check version number and take appropriate action
if ($version >= 10) {
    print "No upgrade necessary\n"; }
elsif ($version >= 6 && $version < 10) {
    print "Reinstall software\n"; }
else { print "Sorry, cannot upgrade\n"; }
```

**Output:**

Sorry, cannot upgrade

*Tip*

To write an **if** statement that executes a single PERL statement when an expression is true, the following shorthand form can be used:

**Example:**

```
$a = 10;
print "$a" if ($a = 10);
```

In this case, the PERL interpreter executes the statement only when the expression is true.

**Using The unless Statement**

Sometimes a block of statements needs to be executed if a condition is false. For example, instead of writing:

"if the user is not admin, don't run this program"

it maybe easier to write

"unless the user is admin, don't run this program"

PERL includes the **unless** statement precisely for this purpose.

The statement can be expressed as unless the user is root, don't run this program with the following PERL code:

```
unless ($user eq "admin") {
    print "you must be \"admin\" to run this program.\n";
    exit;
}
```

In this case, unless the variable \$user contains the string **root**, the program exits.

*Tip*

A common use of **unless** is to test whether a file has been opened successfully.

*Note*



The **unless** statement has the same form as **if**, including the use of **elsif** and **else** clauses. The difference is that **unless** executes its statement block only if the condition is false.

*Tip*



As with the *if* statement, the *unless* statement can be used in the following form:  
 statement unless (expression);

which executes the statement when the expression is false. For example, if a PERL program needs to be exited unless the variable \$username is equal to "root", you could write:

```
exit unless ($username eq "root");
```

**LOOPS**

**Repeating A Task By Looping**

Sometimes it is necessary to execute a block of statements repeatedly until some condition becomes false. The operation of repeating a block of statements is called looping.

**Using An while Loop**

Diagram 15.3 illustrates the flow control of a loop which might be implemented with the while statement.

**Syntax**

```
while (expression) {
    ... statements for execution as long as expression is true; }

```

The **expression** could be anything the value of a variable or the comparison of two strings or numbers. If an array is used as a condition, for example, the while loop (mentioned below) executes until the array has no elements left.

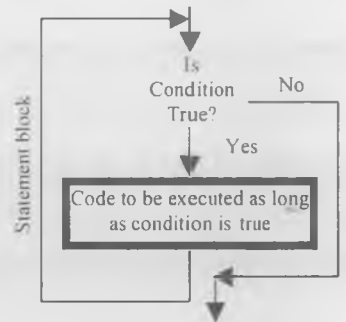


Diagram 15.3

**Example 6:** (Refer diagram 15.4)

```
#!/c:/perl/bin/perl.exe -w
@names = qw(Chhaya Hansel Sharanam);
while (@names) {
    $name = shift @names;
    print "$name\n";
}

```

In the above example, each time through the loop, an array element is extracted into the variable \$name using **shift** function.

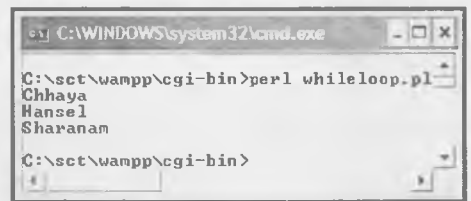


Diagram 15.4: The whileloop.pl output in command window

**Using An until Loop**

The **until** statement is just like **while** statement, but it repeats a block of statements until a specified condition becomes true. In other words, the **until** statement executes the block as long as that condition is false. The **until** statement will be used whenever it is found most natural to think of repeating a loop until an expression becomes true.

**Syntax:**

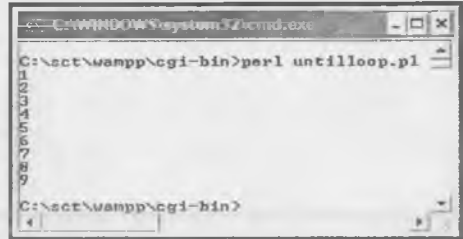
```
until (expression)
{ ... block of statements to execute as long as expression is false;}
```

**Example 7:** (Refer diagram 15.5)

```
#!c:/perl/bin/perl.exe -w
```

```
$n = 1;
until ($n == 10) {
    print "$n \n";
    $n = $n + 1;
}
```

The above example prints the numbers 1 to 9. Statements in the block get executed as long as the condition is false i.e., first it prints \$n and then increments it by 1. Once \$n gets the value 10, the condition becomes true and the execution of the block stops.



```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl untilloop.pl
1
2
3
4
5
6
7
8
9
C:\sct\wampp\cgi-bin>
```

**Diagram 15.5:** The untilloop.pl output in command window

**Using A for Loop**

The **for** statement is another way that a PERL program can execute a block of statements any number of times, based on the value of an expression. The syntax of a for loop is quite a different from that of *while* and *until* loops.

**Syntax:**

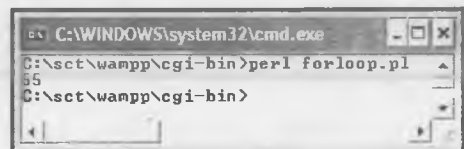
```
for (expr_1; expr_2; expr_3) { ... statement block; }
```

The first expression **expr\_1** is evaluated once, at the beginning of the loop, and the statement block is executed as long as the second expression **expr\_2** is true. The third expression **expr\_3** is evaluated after each execution of the statement block. Any of these expressions can be omitted, but the semicolons must be included. Also, the braces around the statement block are required.

**Example 8:** (Refer diagram 15.6)

Here is a loop to add the numbers from 1 to 10:

```
#!c:/perl/bin/perl.exe -w
for ($i=1; $i<= 10; $i++) {
    $sum += $i;
}
print "$sum";
```



```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl forloop.pl
55
C:\sct\wampp\cgi-bin>
```

**Diagram 15.6:** The forloop.pl output in command window

In this example, the actual work of adding the numbers is done in the statement controlled by the **for** loop the statement that appears within curly braces ({}). The following steps describe what this for loop does:

1. \$i is initialized to 1 before the loop starts.
2. The loop runs as long as, \$i is less than or equal to 10.
3. Each time through the loop, \$i is added to \$sum.
4. After each iteration of the loop, \$i is incremented.

**Note**

This is the most common form of **for** loop. The variable \$i is used as a counter. It is initialized to a value at the beginning of the loop, incremented each time through the loop, and tested to determine when to end the loop. The \$i variable is also known as the loop variable.

## Using A foreach Loop

Besides while, until, and for, PERL provides the **foreach** statement to execute a block of code for each element in an array. Of course, a for loop could be used to access each element of an array by its index, but it's much simpler to access each array element with the **foreach** statement.

### Syntax:

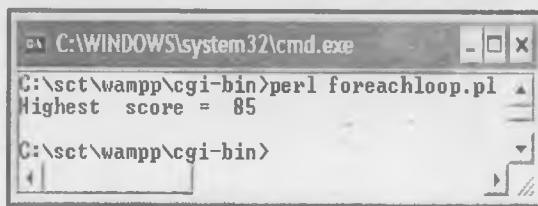
```
foreach $Variable (@Array) {
    ... statement block;
}
```

This **foreach** loop can be read as follows: for each variable in the array, execute the statement block. The **foreach** statement assigns to **\$Variable** an element from the **@Array** and executes the statement block. The **foreach** statement repeats this procedure until no array elements are left.

### Example 9: (Refer diagram 15.7)

The goal is to find the largest number in an array of numbers. Here's how such a program might be written using a **foreach** loop.

```
#!/c:/perl/bin/perl.exe -w
# Define an array of numbers
@scores = (65, 49, 76, 69, 85, 82, 64, 70, 72);
$max = 0;
# Loop through array and find highest score
foreach $score (@scores) {
    if ($score > $max) { $max = $score; }
}
# Print the highest score
print "Highest score = $max\n";
```



```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl foreachloop.pl
Highest score = 85
C:\sct\wampp\cgi-bin>
```

Diagram 15.7: The **foreachloop.pl** output in command window

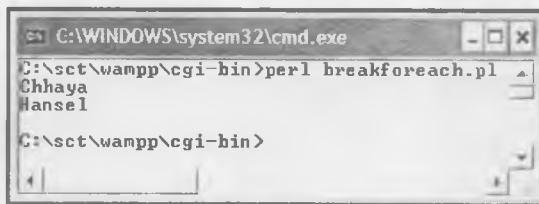
If the **foreach** loop is compared with the **for** loop for processing elements of an array, it will be seen that the **foreach** loop looks simpler because no array index has to be used to access the elements of the array. This eliminates all occurrences of expressions such as **\$scores[\$i]**.

## Breaking Out Of A Loop

Sometimes a loop needs to be broken out of **before** it has gone through all of its iterations. For example, a name in an array of names is searched for and the loop needs end as soon as a match is found.

### Example 10: (Refer diagram 15.8)

```
#!/c:/perl/bin/perl.exe -w
@names = qw (Chhaya Hansel Sharanam);
$customer = "Hansel";
foreach $name (@names) {
# Break the loop if $customer is found
    print "$name \n";
    last if($name eq $customer);
}
```



```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl breakforeach.pl
Chhaya
Hansel
C:\sct\wampp\cgi-bin>
```

Diagram 15.8: The **breakforeach.pl** output in command window

In the above example, customer names are stored in the **@names** array. The program looks for the name stored in the variable **\$customer**. Once the match is found, the loop is broken using the **last** command.

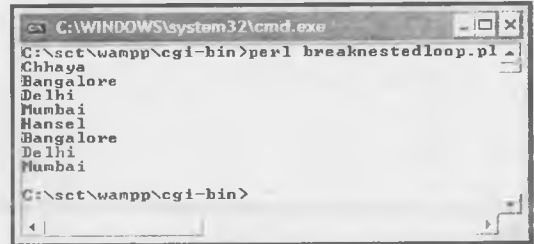
To end the **foreach** loop, the **last** command has to be put in the statement block of the **foreach** loop.

Last can be used in all types of loops: while, until, for, and foreach.

It's not unusual to have one loop embedded inside another. For example, there might be a **foreach** loop inside a **while** loop and the **while** loop needs end when \$line matches one of the items in the @names array.

**Example 11:** (Refer diagram 15.9)

```
#!c:/perl/bin/perl.exe -w
@names = qw (Chhaya Hansel Sharanam);
@locations = qw (Bangalore Delhi Mumbai);
while (@names) {
    $name = shift @names;
    print "$name \n";
    foreach $loc (@locations) {
        print "$loc \n"; }
    last if($name eq 'Hansel');
# Ends the while loop
}
```



```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl breaknestedloop.pl
Chhaya
Bangalore
Delhi
Mumbai
Hansel
Bangalore
Delhi
Mumbai
C:\sct\wampp\cgi-bin>
```

Diagram 15.9: The breaknestedloop.pl output in command window

### Skipping An Iteration Of A Loop

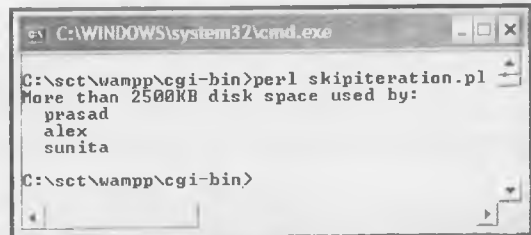
When loops are used in a PERL program, it may not be necessary to perform computations for all iterations of the loop. Typically some iterations of the loop may require to be skipped. For example, a PERL program might be written that generates a message if a user has used more than 2500KB of disk space.

The user names and the corresponding disk usage are stored in an associative array. In this case, an iteration of the loop needs to be skipped whenever a user's disk usage is less than 2500KB.

The following program shows how to skip an iteration of a loop by using the **next** command.

**Example 12:** (Refer diagram 15.10)

```
#!c:/perl/bin/perl.exe -w
# Define an associative array of usernames and disk usage
%disk_usage = ("prasad" => 3500, "alex" => 4800, "john" => 1950, "sunita" => 3100, "susan" => 2100);
$limit = 2500; # Usage must be less than this
# Process the disk usage log and list the users
# who exceed the current limit
print "More than ${limit}KB disk space used by:\n";
foreach $user (keys(%disk_usage)) {
# Skip this iteration if disk use is below limit
    next if ($disk_usage{$user} < $limit);
# Otherwise, print the username
    print " $user\n";
}
```



```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl skipiteration.pl
More than 2500KB disk space used by:
prasad
alex
sunita
C:\sct\wampp\cgi-bin>
```

Diagram 15.10: The skipiteration.pl output in command window

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. The auto-increment operator is denoted by the sign \_\_\_\_\_
2. An \_\_\_ statement is a Conditional statement.
3. The \_\_\_\_\_ operator is an operator for three-way comparison of numbers in PERL.
4. All string comparisons are done by comparing the \_\_\_\_\_ values of the characters.
5. The if statement executes a block of statements if a condition is \_\_\_\_\_.
6. To execute a block of statements repeatedly a \_\_\_\_\_ is used.
7. The \_\_\_\_\_ keyword is used to break out of a loop.

### TRUE OR FALSE

8. The (--) operator is an auto-decrement operator.
9. The (+=) operator adds the value of the variable on the right hand side to that on the left hand side and stores the result in the variable on the right hand side.
10. The (==) string can be used to compare two strings.
11. The numeric value of a character is nothing but the ASCII code of that character.
12. The unless statement executes its block only if the condition is true.

## HANDS ON EXERCISES

1. If \$ctr=10, print the value of:
  - \$ctr2 where \$ctr2=\$ctr+1
  - \$ctr3 where \$ctr3=\$ctr++
  - \$ctr4 where \$ctr4=++\$ctr
2. If \$a=10 and \$b=2 then print the value of \$a after each of the following operations:
  - \$a+=\$b
  - \$a-=\$b
  - \$a\*=\$b
  - \$a/=\$b
3. Write a loop to add numbers from 50 to 75.
4. Use the **if ... else** statement to categorize the marks of a student as Distinction, First Class, Second Class, Pass Class or Fail.
5. @numbers=(78, 87, 34, 45, 56, 23,64, 12, 32, 21, 54, 43);  
Write a foreach loop to print the lowest number in the above array.

## 16. PERL FUNCTIONS

A significant part of PERL's capability is its built-in **functions**. These functions are nothing but subroutines that are predefined and are available for use anywhere in a PERL program. The creators of PERL recognized that there were certain common tasks, which every programmer implementing PERL, would have to perform. PERL developers wrote the code to perform these tasks and embedded them into the PERL language using functions.

### STRING FUNCTIONS

String functions are used to manipulate text. The following are a few of PERL's string functions.

#### Chop The Last Character Of A String

Many PERL programs process the contents of a text file one line at a time. When a PERL program reads a line of text into a scalar variable, that text string includes all the characters on the line plus a **newline** character at the very end of each line.

PERL provides the **chop()** function, to remove the last character from the string.

##### Syntax:

```
chop(<variable name>);
```

**Example 1:** (Refer diagram 16.1)

```
#!c:/perl/bin/perl.exe -w
$line = "I Like PERL.";
chop($line);
print "$line";
```

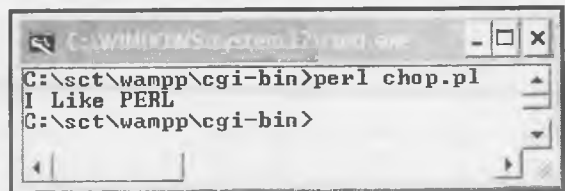


Diagram 16.1: The chop.pl output in command window

In the above example, the **chop(\$line)** function, will remove the full stop contained at the end of the statement in the scalar variable \$line. So now, the contents of \$line will be **I Like PERL** without the period at the end.

#### Chomp A Newline Character

Typically, the last character in a string needs to be taken out only if it were a newline character. The **chop()** function, will remove any character which is at the end of the string.

To overcome this, PERL provides the **chomp()** function. The **chomp()** function will remove the last character of the string only if it is a newline character.

##### Syntax:

```
chomp(<variable name>);
```

**Example 2:** (Refer diagram 16.2)

```
#!c:/perl/bin/perl.exe -w
$line = "I Like PERL.\n";
chomp($line);
print "$line";
```

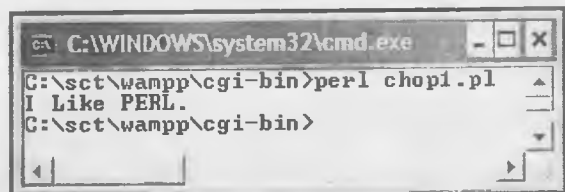


Diagram 16.2: The chop1.pl output in command window



In the above example, the `chomp($line)` function, will remove the newline character from the variable `$line`. So now, the contents of `$line` will be `I Like PERL`. Notice the existence of the period at the end of the line in this case.

## Concatenating Strings

A common text-processing step is to concatenate, or join, strings. Suppose the first and last names of a customer are stored in two separate scalar variables (`$firstname` and `$lastname`) the two parts need to be joined into a full name. The string concatenation operator can be used to accomplish this task, as illustrated by the following PERL code.

**Example 3:** (Refer diagram 16.3)

```
#!/c:/perl/bin/perl.exe -w
$firstname = "Vaishali";
$lastname = "Shah";

# Concatenate the two strings
$fullname = $firstname . " " . $lastname;
print "$fullname\n";
```

```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl concat.pl
Vaishali Shah
C:\sct\wampp\cgi-bin>
```

Diagram 16.3: The concat.pl output in command window

The contents the variable `$fullname` is the result of concatenating three strings:

- `$firstname`
- A blank space (written as a blank space enclosed in double quotes)
- `$lastname`

As seen from this example, the string concatenation operator is represented by a period (`.`). Concatenation of two or more strings can be accomplished by simply putting a period in-between the strings.

*Tip*



The assignment form of the concatenation operator (`.=`) can be used to concatenate two strings and store the result back in the first string.

For example, if `ing` needs to be appended to a string, the following should be written:

```
$str = "load";
$str .= "ing";
# Now $str is "loading"
```

```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl marker.pl
*****
C:\sct\wampp\cgi-bin>
```

Diagram 16.4: The marker.pl output in command window

## Repeating Strings

A useful operator is the repetition operator, denoted by `x`. The `x` operator can be used to repeat a string a specified number of times. Suppose a string needs to be initialized to 15 asterisks (`*`) and then use that line of asterisks as a separator in a text file (or the output printed by the PERL program).

**Example 4:** (Refer diagram 16.4)

The string can be initialized with a repeated number of asterisks by using the `x` operator.

```
#!/c:/perl/bin/perl.exe -w
# Define $marker as a string of 15 asterisks
$marker = "*" x 15;
print $marker;
```

The same result can be achieved by repeating a string of 3 asterisks 5 times, as follows:

```
$marker = "***" x 5;
```

The string repetition operator can be actually used in many useful programs. Suppose that a simple text plot of some data needs to be generated, replicating asterisks where the number of asterisks is proportional to the value being plotted.

### Example 5: (Refer diagram 16.5)

The following PERL program illustrates the creation of a simple plot showing the relative costs for a number of cars:

```
#!c:/perl/bin/perl.exe -w
# Initialize an associative array of costs
%costs = ("Mercedes" => 90000, "Esteem" => 82000, "Toyota" => 37500);
foreach $car (keys(%costs)) {
# Initialize the marker
$marker = " * ";
# Retrieve the cost for this car
$amount = $costs{$car};
# Compute how many thousands (divide by 10000)
$amount /= 10000;
# Set up that many asterisks
$marker x= $amount;
print "$car: \t$marker\n";
}
$marker = "-" x 40;
print "$marker\nNote: * = \$10,000\n";
```

This program's output is quite usable, at least as a way to compare the costs of these cars.

If this program is studied, it will be noticed that the string of asterisks is prepared in the variable named \$marker in the following manner:

\$marker is set to a single asterisk.

The cost of the car is divided by 10000 to convert it to the number of 10000 blocks that exist in the cost of the car.

\$marker is repeated by the cost of the car divided into ten thousand blocks by using the x= operator (this is the assignment form of the x operator) with the following statement:

\$marker x= \$amount;

### Extracting A Substring

Another type of text manipulation is to extract a part of a string. The substr ( ) function can be used to extract a substring from a string.

#### Syntax:

substr(string, offset, length);

#### Example 6: (Refer diagram 16.6)

```
#!c:/perl/bin/perl.exe -w
$line1 = "I like PERL";
$line2 = "From: Ivan Bayross";
$part = substr($line1, 7, 4);
$from = substr($line2, 6);
print "$part \n";
print "$from \n";
```

```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl plot.pl
Toyota:      * * *
Mercedes:    * * * * * * * * *
Esteem:      * * * * * * * *
-----
Note: * = $10,000
C:\sct\wampp\cgi-bin>
```

Diagram 16.5: The plot.pl output in command window

```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl substr.pl
PERL
Ivan Bayross
C:\sct\wampp\cgi-bin>
```

Diagram 16.6: The plot.pl output in command window

The first `substr()` call in this example skips the first 7 characters of `$line1` and return the next 4 characters. The second `substr()` call skips the first 6 characters of `$line2` and returns the remainder of that text string.

## The Index Function

If the word **PERL** needs to be extracted from the sentence **I like PERL.**, without counting the number of characters to be skipped, the `index()` function can be used. Typically indexes are used to find the start position of a string within a string. This can be achieved as follows:

```
$line = "I like PERL.";
$pos = index($line, "PERL");      # $pos = 7
$part = substr($line, $pos, 4);   # $part = "PERL"
```

## Finding The Length Of A String

In many situations, it is necessary to find the length of a string. The length of a string is the number of characters in the string.

Consider a situation where the length function can be used. Suppose a text needs to be printed and then print another line with a number of dashes that essentially underlines the text. The following example shows how the length function can be used to solve this problem:

**Example 7:** (Refer diagram 16.7)

```
#!/c:/perl/bin/perl.exe -w
$title = "The PERL Zone";
# Find the length of $title
$length = length($title);
# Print the title
print "$title\n";
# Prepare a line of dashes and
  print it
$uline = "-" x $length;
print "$uline\n";
```

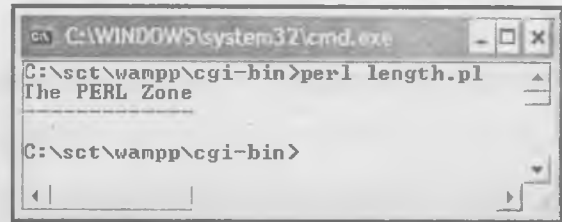


Diagram 16.7: The `length.pl` output in command window

Consider another problem of extracting a substring from a line of text. Suppose a variable named `$find` contains the substring which is required to be extracted from a string named `$line`. The `index()` can be used to find the start of the substring. Then use `substr()` to extract the substring. However, to use `substr()`, the number of characters in `$find` need to be specified. This is a situation where the `length()` has to be used.

Following example shows how the length function can be used when locating and extracting a substring:

**Example 8:** (Refer diagram 16.8)

```
#!/c:/perl/bin/perl.exe -w
$line = "I like PERL";
$find = "PERL";
# Find start of $find
$pos = index($line, $find);
# Length of $find
$ncchar = length($find);
# Extract substring by calling
  substr
$part = substr($line, $pos, $ncchar);
print $part;
```

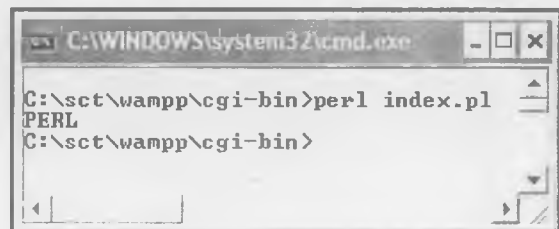


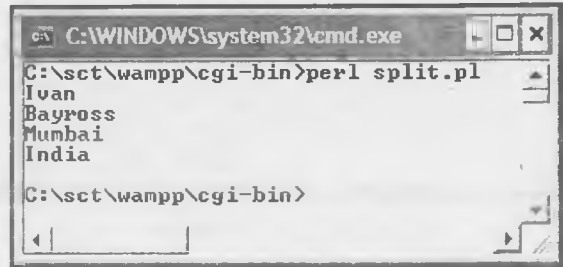
Diagram 16.8: The `index.pl` output in command window

## Splitting A String Into Several Parts

The `split()` is mainly used to split a string into several parts. For example, a string having multiple fields separated by a comma (,) can be split into several parts.

**Example 9:** (Refer diagram 16.9)

```
#!/c:/perl/bin/perl.exe -w
$record1 = "Ivan,Bayross,Mumbai,India";
@fields = split(",", $record1);
while (@fields) {
    $field = shift @fields;
    print "$field \n";
}
```



```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl split.pl
Ivan
Bayross
Mumbai
India
C:\sct\wampp\cgi-bin>
```

Diagram 16.9: The `split.pl` output in command window

## ARRAY FUNCTIONS

PERL includes a number of built-in functions that enable the manipulation of arrays.

### Adding An Array Element

To add an array element PERL provides few built-in functions.

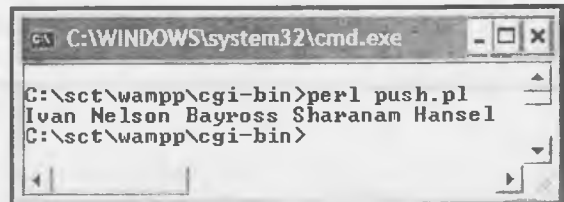
The `push()` function adds one or more elements to the end of an array.

**Example 10:** (Refer diagram 16.10)

```
#!/c:/perl/bin/perl.exe -w
@names = qw (Ivan Nelson Bayross);
push(@names, "Sharanam", "Hansel");
print "@names";
```

Now, the `@names` array contains five elements in the following order.

("Ivan", "Nelson", "Bayross", "Sharanam", "Hansel")



```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl push.pl
Ivan Nelson Bayross Sharanam Hansel
C:\sct\wampp\cgi-bin>
```

Diagram 16.10: The `push.pl` output in command window

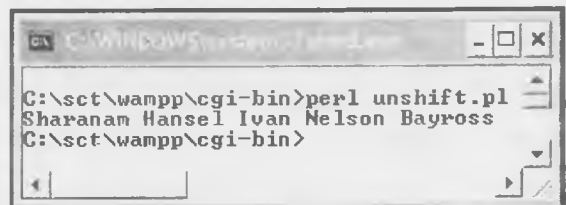
The `unshift()` adds one or more elements to the beginning of an array.

**Example 11:** (Refer diagram 16.11)

```
#!/c:/perl/bin/perl.exe -w
@names = qw (Ivan Nelson Bayross);
unshift(@names, "Sharanam", "Hansel");
print "@names";
```

Now, the `@names` array contains five elements in the following order.

("Sharanam", "Hansel", "Ivan", "Nelson", "Bayross")



```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl unshift.pl
Sharanam Hansel Ivan Nelson Bayross
C:\sct\wampp\cgi-bin>
```

Diagram 16.11: The `unshift.pl` output in command window

### Removing an Array Elements

To remove an array element PERL provides few built-in functions.

The `pop()` returns the last element of an array and removes the same from the array and decreases the size of the array by one.

**Example 12:** (Refer diagram 16.12)

```
#!c:/perl/bin/perl.exe -w
@names = qw (Ivan Nelson Bayross);
pop(@names);
print "@names";
```

Now, the @names array contains only two elements in the following order. ("Ivan", "Nelson")

The `shift()` function returns the first element of an array and removes the same from the array and decreases the size of the array by one.

**Example 13:** (Refer diagram 16.13)

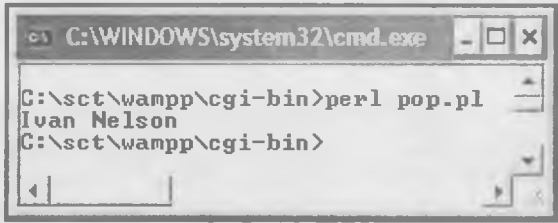
```
#!c:/perl/bin/perl.exe -w
@names = qw (Ivan Nelson Bayross);
shift(@names);
print "@names";
```

Now, the @names array contains only two elements in the following order. ("Nelson", "Bayross")

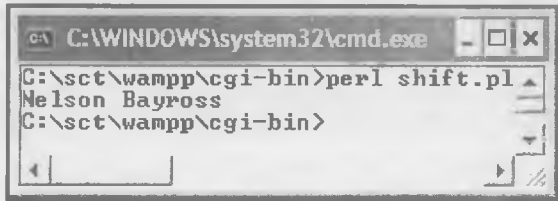
**Example 14:** (Refer diagram 16.14)

The following example describes implementing a stack using a menu with options to Push, Pop and Quit.

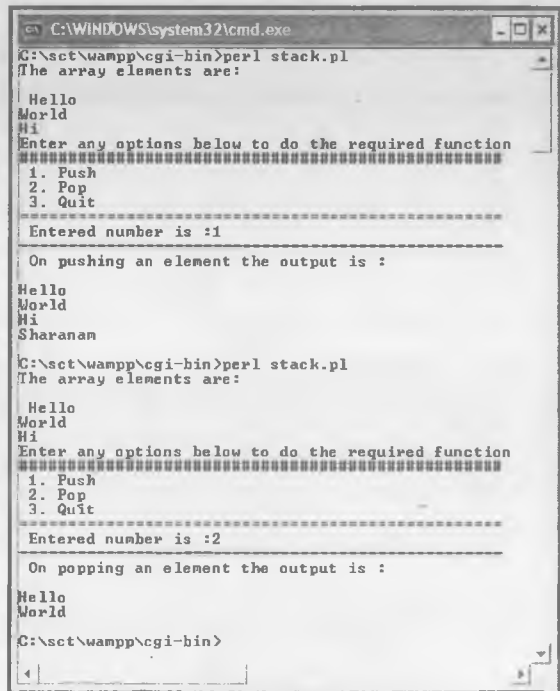
```
#!c:/perl/bin/perl.exe -w
@mylist = qw(Hello World Hi);
print "The array elements are: \n \n ";
for($i=0;$i<=$#mylist;$i++) {
    print "$mylist[$i]\n"; }
print "Enter any options below to do the required function\n";
print "# " x 50;
print "\n";
print " 1. Push \n";
print " 2. Pop \n";
print " 3. Quit \n";
print "=" x 50;
print "\n Entered number is :";
$x=<STDIN>;
print "*" x 50;
if ($x==1) {
    push(@mylist, "Sharanam");
    print "\n On pushing an element the output is :\n \n";
    for($i=0;$i<=$#mylist;$i++) {
        print "$mylist[$i]\n"; }
}
elseif ($x==2) {
    pop(@mylist);
    print "\n On popping an element the output is :\n \n";
```



**Diagram 16.12:** The pop.pl output in command window



**Diagram 16.13:** The shift.pl output in command window



**Diagram 16.14:** The stack.pl output in command window

```

for($i=0;$i<=$#mylist;$i++) {
    print "$mylist[$i]\n"; }
}
elseif ($x==3) {
    exit; }

```

## Sorting An Array

An array can be sorted by using the PERL `sort()` function. The `sort()` function returns a sorted array but **does not alter** the contents of the original array.

**Example 15:** (Refer diagram 16.15)

```

#!c:/perl/bin/perl.exe -w
@chars = qw(D G C H A F E B);
@sorted_chars = sort(@chars);
for ($x=0; $x <= $#sorted_chars; $x++) {
    print "Character $x = $sorted_chars[$x]\n";
}

```

```

C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl sort.pl
Character 0 = A
Character 1 = B
Character 2 = C
Character 3 = D
Character 4 = E
Character 5 = F
Character 6 = G
Character 7 = H
C:\sct\wampp\cgi-bin>

```

Diagram 16.15: The sort.pl output in command window

The following example reads a Password file (`password1.txt`) and prints all the user names and passwords in a sorted manner.

**Example 16:** (Refer diagram 16.16)

```

#!c:/perl/bin/perl.exe -w
unless(open("passwd","passwd.txt")) {
    print "cannot open file\n"; }
while(($line=<passwd>)) {
    { @users=split(":",$line); }
    @sort_users=sort@users;
}
foreach $user(@sort_users) {
    print "$user\n"; }
close("passwd");

```

```

C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl sortpass.pl
sharanam
sshah
vaishali
vshah
C:\sct\wampp\cgi-bin>

```

Diagram 16.16: The sortpass.pl output in command window

## Reversing Array Elements

The PERL `reverse()` function can be used to reverse the order of elements in an array. The `reverse()` does not alter the contents of the original array in any way. The `reverse()` can be used to change the order of a sorted index from ascending to descending or to simply reverse the order of elements in an array.

**Example 17:** (Refer diagram 16.17)

```

#!c:/perl/bin/perl.exe -w
@chars = qw(T E N);
@reverse_chars = reverse(@chars);
print "@reverse_chars";

```

```

C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl reverse.pl
N E T
C:\sct\wampp\cgi-bin>

```

Diagram 16.17: The reverse.pl output in command window

## Splicing An Array

The PERL `splice()` function can be used to either append one array to another or to replace a portion of one array with another array.

**Syntax:**

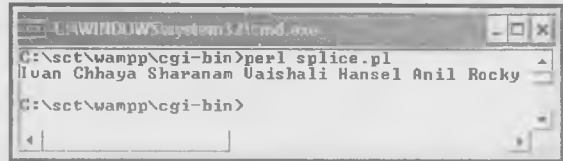
```
splice(@oldarray, offset, length, @newarray);
```

The `splice()` function first removes `length` items from the `@oldarray` starting with the item specified by `offset`. `splice()` then inserts the `@newarray` array in place of the elements that were removed.

**Example 18:** (Refer diagram 16.18)

```
#!/c:/perl/bin/perl.exe -w
@names = qw(Ivan Chhaya Sharanam Vaishali Hansel);
@new = qw(Anil Rocky);
splice(@names, $#names+1, 0, @new);
print "@names\n";
```

In the above example, `splice()` will add the elements from the `@new` array to the `@names` array. Thus, `@names` will now contain the following elements:



**Diagram 16.18:** The `splice.pl` output in command window

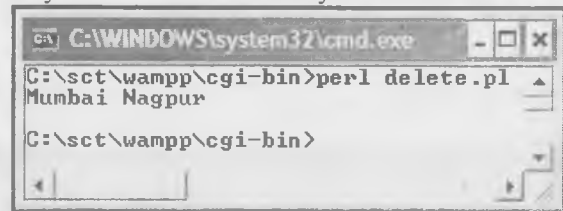
### Deleting an Associative Array Element

PERL provides the `delete()` function for removing an entry from an associative array.

**Example 19:** (Refer diagram 16.19)

```
#!/c:/perl/bin/perl.exe -w
%cities = ("Chennai"=>"East",
          "Nagpur"=>"Central",
          "Mumbai"=>"West");
```

```
delete($cities{Chennai});
@names = keys %cities;
print "@names\n";
```



**Diagram 16.19:** The `delete.pl` output in command window

## MATHEMATICAL FUNCTIONS

PERL's some of the built-in math functions are:

Function Call	Description
<code>Abs(value)</code>	Returns the absolute value of the argument
<code>Atan2(Y,X)</code>	Returns the arctangent of Y/X
<code>Cos(expr)</code>	Returns the cosine of the angle <code>expr</code> (radians)
<code>Exp(expr)</code>	Returns e raised to the power <code>expr</code>
<code>log(expr)</code>	Returns the logarithm (to base e) of <code>expr</code>
<code>Rand(expr)</code>	Returns random value between 0 and <code>expr</code>
<code>Sqrt(expr)</code>	Returns the square root of <code>expr</code>
<code>sin(expr)</code>	Returns the sine of the angle specified by <code>expr</code> (in radians)
<code>srand(expr)</code>	Sets the seed for random number generation

The problem is to measure the height of some object, in this case, a lamppost. Suppose a person is standing at a distance `L` from the post. Angle `A` is the angle of elevation of the lamp from the perspective of the person. If the height of the person is `h`, then the height of the lamppost `H` is given by the following expression:

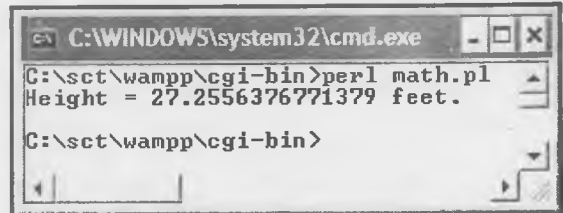
$$H = h + L \times (\sin(A)/\cos(A))$$

Assuming the person is 6ft tall at distance of 100 feet, with an angle of elevation of 12 degrees.

The following block of PERL code, illustrates the use of the `sin()` and `cos()` functions to calculate the height of the lamppost:

**Example 20:** (Refer diagram 16.20)

```
#!c:/perl/bin/perl.exe -w
$A = 12;
$h = 6;
$L = 100;
# Convert A into radians
$A *= 3.14159/180;
# Compute the height of the lamp
  post
$H = $h + $L * sin($A) / cos($A);
print "Height = $H feet. \n";
```



**Diagram 16.20:** The math.pl output in command window

## TIME FUNCTIONS

PERL provides the following **time()** functions.

When **time()** is called, it returns the seconds elapsed since 00:00:00 hours GMT on January 1, 1970. The **gmtime** and **localtime** functions then convert this to the binary time as required.

Greenwich Mean Time (GMT) is a standard reference time that all computers use. GMT refers to the local time in Greenwich, England).

Local time refers to the time at a specific locality. e.g. when its 10 a.m. in Washington D.C., it is already 10 p.m. in India.

**Example 21:**

```
$gmt = gmtime(time);
$local = localtime(time);
```

Thus \$gmt contains the GMT time and \$local contains the local time.

Task	Functions
Get current date and time in a binary form	Time
Convert time to GMT	Gmtime
Convert time to Local time	Localtime

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

- \_\_\_\_\_ are subroutines that are predefined and are available for use anywhere in PERL program
- PERL provides the \_\_\_\_\_ function to remove the last character from a string.
- The assignment form of the concatenation operator that appends a new string to the left operand is \_\_\_\_\_
- The \_\_\_\_\_ operator can be used to repeat a string a specified number of times.
- The \_\_\_\_\_ function can be used to extract a part of the string.
- The PERL function \_\_\_\_\_ is used to either append one array to another or to replace a portion of one array with another array.

### TRUE OR FALSE

- The **chomp()** function will remove the last character only if it is not a new line character.
- The **pop()** function is used to append an array element to the beginning of the array.
- The **shift()** function removes the first element from an array.



## HANDS ON EXERCISES

1. `$line = "Keep Your City Clean.";`  
Write a program to remove the last character using `chop()` function and store it in the variable. Then print the value of `$line` and the variable.
2. `$line = "Keep Your City Clean.\n.";`  
Write a program using `chomp()` function to remove the newline character.
3. Write a PERL program to concatenate two strings with space between them and append the third string using the concatenation operator storing the result in the same string.
  - "Jodie"
  - "Foster"
  - "is blonde. "
4. Use the **substr** function to extract first five characters of the string "Play Light Music". Then print those five characters.
5. Write a program to print the total length of the string given below, then extract a string and count its length. Example: "Jodie Foster is blonde." Extract "Foster" and count its length.
6. Write the program to add elements to the beginning of the array and sort it.

## 17. FILEHANDLING

PERL programs are usually designed to read input or get their input from the **standard input** and print or send their output to **standard output**. By default, standard input is the **keyboard** and standard output is the **VDU**.

PERL programs often access and use information held in files, which means often a PERL script needs to read data from a file and write data to a file. For example, a Web page that accepts reader feedback might use a CGI program that saves reader feedback in a text file. The PERL script has to open the file, store the reader's comments in the file and close the file. This material shows describes to work with standard input, standard output, files and directories.

### UNDERSTANDING STDIN AND STDOUT

PERL programs get their input from the **standard input** and send their output to the **standard output**. In PERL programs, files are referenced through identifiers known as **filehandles**. A filehandle is a pointer to a location in memory where the file is currently opened. To obtain a valid filehandle, **open()** has to be used with a filename. To use standard input and standard output a file does not have to be explicitly opened. Instead, PERL automatically defines and opens **STDIN** and **STDOUT** as filehandles for standard input and standard output, respectively.

By default, **STDIN** is tied to the keyboard and **STDOUT** is tied to the VDU. That means when a PERL program reads from **STDIN**, it receives its input from the keyboard. Similarly, when PERL sends its output using **print()**, the output appears on the VDU.

#### The <STDIN> Filehandle

<STDIN> is PERL defined filehandle, which points to the Standard Input device for the program. <STDIN> can be used to read either single lines of text or multiple lines of text from standard input.

To read a single line of text and assign it to a variable:

**Syntax:** #Scalar Context (1 line read)  
`$var = <STDIN>;`

where, \$var is the scalar variable.

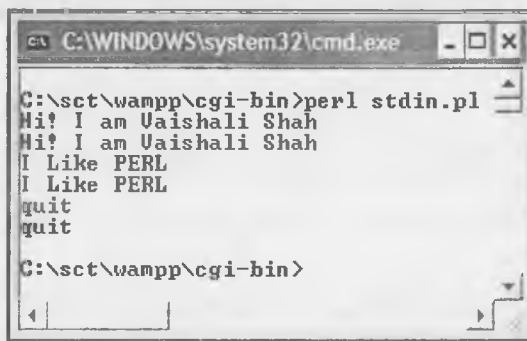
**Example 1:** (Refer diagram 17.1)

```
#!/c:/perl/bin/perl.exe -w
while ($line = <STDIN>) {
    print "$line";
    last if($line eq "quit\n");
}
```

When the above program is run, it echoes back each line of text that has been typed and exits when quit is typed in and the Enter key is pressed.

Here, the expression `$line = <STDIN>` reads a line from the standard input (keyboard) into the variable \$line. The **print()** sends its output to the standard output, which is the VDU.

If the line read from **STDIN** is not assigned to a specific variable, then PERL assigns that line to a special variable called `$_`.



```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl stdin.pl
Hi! I am Vaishali Shah
Hi! I am Vaishali Shah
I Like PERL
I Like PERL
quit
quit
C:\sct\wampp\cgi-bin>
```

Diagram 17.1: The `stdin.pl` output in command window

**Example 2:**

```
#!c:/perl/bin/perl.exe -w
while (<STDIN>) {
    print "$_"; }
```

To read multiple lines of text into an array:

**Syntax:** #List Context (many lines read)  
**@array** = <STDIN>;

where, @array is an indexed array to store the lines read. In this case, <STDIN> returns a **list** consisting of all the remaining lines from standard input upto the end of the file.

**Example 3:**

```
@lines = <STDIN>;
```

**The <STDOUT> Filehandle**

<STDOUT> is a PERL defined filehandle, which points to Standard Output for all PERL programs. This could be the VDU or at times the printer. <STDOUT> can be used to write output from a program.

To send output to STDOUT:

**Syntax:**

```
print STDOUT "<Whatever you want to print>;"
```

**Example 4:**

```
print "Good Morning , Jack!\n";
```

or

```
print STDOUT "Good Morning , Jack!\n";
```

**Using The printf Function**

The C programming language includes a function called **printf** that serves as an all-purpose output function. PERL also includes a printf function that's similar to its namesake in C. The printf function is more complex than print. It offers greater control over how the output will be formatted.

Consider the following **printf** function:

```
printf "%- 12s %10s %12s %8s\n", "Item", "Pounds", "Price/lb", "Total";
```

**Output:**

```
Item Pounds Price/lb Total
```

**This Is How It Works**

The first argument of the printf() is a string which describes the format of the output. In this case, the format string is:

```
"%-12s %8s %8s %8s\n"
```

Each % sign in the format string marks the beginning of a format specification. printf() expects a format specification for each argument following the format string. In this case, there are four specifications:

- The first format specification (%-12s) is applied to the first string ("Item"),
- the second one (%8s) to ("Pounds") and so on.

Each format specification begins with a percent sign followed by some optional numbers and a letter. That letter determines the type of data being printed. For example, the %-12s format specification letter s, which specifies format for strings.

The -12 part says that the string should occupy 12 spaces and the string should be left justified. The **minus sign** (-) specifies left justification.

Besides the format specifications, `printf()` prints all the other characters verbatim such as any spaces or text added in the format string (i.e. the first argument to `printf()`).

## Redirecting STDIN And STDOUT

By using redirection operators (`<`, `>`) in the command line STDIN and STDOUT can be bound to files instead of the default keyboard and VDU.

Suppose a PERL program `prog.pl` has been written that reads input from STDIN and prints output to STDOUT. This program can be run with its input being taken from a file named `infile` and output going to a file named `outfile`. The following syntax must be used at command line:

```
<System Prompt> perl prog.pl < infile > outfile
```

The less-than sign (`<`) is the input redirection operator. The greater-than sign (`>`) is the output redirection operator.

## UNDERSTANDING FILES AND DIRECTORIES

Files and directories on a computer are for storing information in an organized manner, just like filing cabinet. Operating systems such as Linux, Windows NT and so on organize files and directories similar to a filing cabinet. Of course, in this case the storage medium is not paper, but a device such as a hard disk drive, floppy disk drive, Pen drive, or a CD-ROM drive. A computer file system has a hierarchical structure with individual files stored in directories. A directory, in turn, may contain other directories.

## OPENING AND CLOSING FILES

### Using the open function

PERL provides the `open()` function, to open a file in memory.

#### Syntax:

```
open(FILEHANDLE, "<filename>");
```

The first argument is an identifier called a **filehandle**. The second argument is the filename, which may be a scalar variable or a string in double-quotes.

#### Example 5:

```
open(STORAGE, "data.txt");
```

When a filename such as `data.txt` is used, the PERL interpreter expects the file to be in the current directory. However, a path name can always be specified to any directory on the hard disk.

The syntax for the path name would depend on the operating system. For example, to open the file `c:\passwd` in Window:

```
open(PASSWD, "c:\passwd");
```

When using the `open()` function always check whether the `open()` function succeeded before using other PERL functions to perform other input/output operations on the file. A common technique is to use the logical OR (denoted by `||`) and combine the open function call with a call to the PERL keyword `die` as follows:

```
open(FILEHANDLE, "filename") || die "Error opening file!";
```

If `open()` succeeds it returns a value `true` and the `die` keyword is never called. When the `open()` fails, the `die` keyword is called and it exits the program with the error message being displayed. The same objective can be achieved by using the `unless` keyword as follows:

```
die "Error opening a file!" unless open(FILEHANDLE, "filename");
```

The program quits with an error message unless `open()` function returns a value `true`.

It's not absolutely necessary to quit a program when `open()` fails. Simply check the return value of `open()` then call and perform some tasks, such as reading from the file, if the return value is true, as follows:

```
if (open(FILEHANDLE, "filename")) {
    # Open succeeded, perform read/write operations
    # Close file
    close FILEHANDLE;
}
```

### Using The close Function

When the filehandle is no longer needed, the file can be closed by calling the `close()` function as follows:

```
close FILEHANDLE;
```

#### Example 6:

```
close STORAGE;
```

### File Open Modes

When a file is opened in a PERL program, the user has to decide what is to be done with the file:

- Read from the file (input)
- Write to the file (output)
- Read from and write to the file (input and output).

Intent can be indicated by the way that the file's name is specified when the file is opened.

### Reading From The File

Suppose a PERL program needs to work with a file named `error-log.txt` that presumably contains a log of any errors that may have occurred. When the error log needs to be read, the file can be opened with `open()` as shown below:

```
$errfile = "error-log.txt";
open(ERROR_LOG, "$errfile") || die("Cannot open $errfile");
```

The `open()` function expects an existing file with specified filename and it opens that file **for reading only**. After opening a file, it can be read as follows:

```
$line = <ERROR_LOG>;
```

### Writing To The file

If an error message is to be written to the error log, the file has to be opened such that it can be written to. In this case, `open()` takes the following form:

```
open(ERROR_LOG, ">$errfile") || die("Cannot open $errfile");
```

As it is seen from this example, simply adding a greater-than sign (`>`) prefix to the filename opens the file in **write mode**. When a greater-than sign (`>`) is used, the `open()` function **creates** the file, if it does not already exist. If the file exists, **it deletes everything** in the file and **then writes** to the file.

To keep the existing data intact and append new information to a file, two greater-than signs (`>>`) are used as shown below:

```
open (ERROR_LOG, ">>$errfile") || die ("Cannot open $errfile");
```

### Reading from and Writing to a File

There are two types of data files:

- Text Files:** They contain data that is organized into lines of characters. The characters are printable characters. For example, World Wide Web documents in HTML and system configuration files in Windows.

- **Binary Files:** They contain data that does not have lines ending with the newline character, nor does the data consist of printable characters. For example, GIF (Graphics Interchange Format) images used in HTML documents, spreadsheets and so on.

### Reading And Writing Text Files

To read a line of text from an open file:

```
$line = <FILEHANDLE>; #Read a line of text
```

```
@lines = <FILEHANDLE>; #Read multiple lines
```

where, FILEHANDLE is the filehandle used when the file was opened with open().

To write a line of text to a file, use **print()** with the FILEHANDLE as the argument followed by the list of scalar variables or quoted strings as follows:

```
print FILEHANDLE "The result is : $result, \n";
```

### Reading and Writing Binary Files

Binary files are read and written to in chunks, which are bigger than the typical lines of text. Each chunk can typically contain around 80 characters. The functions for binary read are:

The *read* and *sysread* functions read a specific number of bytes from a file.

#### Syntax:

```
$bytes_read = read(FILEHANDLE, $buffer, $length, $offset);
```

```
$bytes_read = sysread(FILEHANDLE, $buffer, $length, $offset);
```

The syntax used is identical. The arguments and return values have the same meaning:

- *FILEHANDLE* is the filehandle of an open file from which data is read
- *\$buffer* is the string into which the data is read
- *\$length* is an integer value that specifies how many bytes of data are to be read from *FILEHANDLE*
- *\$offset* is an integer values that specifies where in the *\$buffer* variable to put the bytes (thus, you can read into the middle of a string). The *\$offset* can be *omitted* if the data is to be put at the beginning of *\$buffer*
- *\$bytes\_read* will be the number of bytes actually read from *FILEHANDLE*

#### Example 7:

```
$len = read(IMAGE, $buf, 8192);
```

or

```
$len = sysread(IMAGE, $buf, 8192);
```

The read operation calls read / sysread function and reads the image data into the \$buf variable. The \$len variable holds the actual number of bytes. The **syswrite()** function writes a specific number of bytes to a file which has a syntax similar to that of sysread (and read).

#### Syntax:

```
$bytes_written = syswrite(FILEHANDLE, $buffer, $length, $offset);
```

Thus, this function writes **\$length** bytes from the string **\$buffer** to *FILEHANDLE*. To begin writing from a place other than the beginning of the \$buffer string, use the **\$offset** variable.

#### Example 8:

```
$write = syswrite(IMAGE1, $buf, $len);
```

In this case, the syswrite function writes \$len bytes of data from \$buf to STDOUT.

### Binary Mode File Access In MS-DOS

To set the file access to **binary mode** call the **binmode()** function as follows:

```
binmode(FILEHANDLE);
```

where FILEHANDLE is the filehandle of the open file.

The following example reads from a file and directs it to Standard Output.

**Example 9:** (Refer diagram 17.2)

```
#!/c:/perl/bin/perl.exe -w
unless(open(IMAGE,"javacup.gif")) {
    exit; }
binmode IMAGE;
print "Content-type: image\gif\n\n";
while($len=sysread(IMAGE,$buf,8192)) {
    syswrite(STDOUT,$buf,$len); }
close IMAGE;
exit;
```



Diagram 17.2: The gifread.pl output in command window

### WORKING WITH DIRECTORIES

Directories are special files that contain information about the other directories and files. PERL provides several functions to access and use information in a directory

#### To Open A Directory

Just as filehandle is initialized to open a file, similarly, directory handle is set up to open a directory using **opendir()** function.

**Syntax:**

```
opendir(DIRHANDLE, "<dirname>") || die("Cannot open directory");
```

In this case, DIRHANDLE is the directory handle and dirname is the name of the directory to be opened. If the directory is not opened successfully then an error message will be displayed.

**Example 10:**

```
opendir(NEWDIR, "dept") || die("Cannot open directory");
```

#### To Close A Directory

The **close()** can be called to close the directory when it is no longer in use.

**Syntax:**

```
close(DIRHANDLE);
```

**Example 11:**

```
close(NEWDIR);
```

#### Creating a directory

To create a directory call the **mkdir()** with an argument. It can also include file permissions as can be seen from the example given below.

**Syntax:**

```
mkdir("<directory name>");
```

**Example 12:**

To create a directory named **temp** that is accessible to all only for reading, but only the valid user can read & write, call **mkdir** as follows:

```
mkdir("temp");
```

**To remove a directory**

To remove a directory call the **rmdir()** with an argument. The **rmdir()** does not remove the directory if it contains any files.

**Syntax**

```
rmdir("<directory name>");
```

**Example 13:**

```
rmdir("test");
```

**To Go To The Beginning Of The Directory**

Sometimes, one needs to go back to the beginning of the directory after having read some of the entries. To do so, call the **rewinddir()**

**Syntax:**

```
rewinddir(DIRHANDLE);
```

**Example 14:**

```
rewinddir(NEWDIR);
```

**Reading the contents of the directory**

After the directory is open, its contents can be read by using the **readdir()**. For example, to read all the directory entries into an array with a single call to the **readdir** function:

**Syntax:**

```
@filelist = readdir(DIRHANDLE);
```

**Example 15:**

```
@deptfiles = readdir(NEWDIR);
```

where the array **@deptfiles** contains the directory entries of the directory **NEWDIR**.

**TESTING FILES AND DIRECTORIES**

Sometimes its necessary to check whether a file or directory exists before trying to access it. Even if a file exists, a check may be made whether the file can be read, written to or executed. PERL includes a handy set of file operators to test files and directories.

To test whether a file named **data.txt** exists.

This is how the **-e** operator is used:

```
if (-e "data.txt") {  
    print "File exists. \n";    }
```

All the other file and directory test operators have similar syntax. Each operator is denoted by a single letter with a hyphen (-) prefix. To test a file or a directory, use the operator with an **if** statement, as follows.



**Syntax:**

```
if(-O $name) {          # Test succeeded }
```

where, -O represents the operator and \$name is the name of the file or directory.

The following table lists file operators in PERL

Operator	Returns
-r \$filename	True if file is readable.
-x \$filename	True if file is executable.
-z \$filename	True if file has zero size.
-f \$filename	True if file is a plain file.
-l \$filename	True if file is a symbolic link (applies to UNIX). A symbolic link is a file that refers to another file.
-S \$filename	True if file is a socket.

Operator	Returns
-w \$filename	True if file is writable.
-e \$filename	True if file exists.
-s \$filename	True if file has non-zero size.
-d \$filename	True if file is a directory.
-p \$filename	True if file is a named pipe.
-B \$filename	True if file is a binary file (opposite of -T).
-T \$filename	True if file is a text file.

**SELF REVIEW QUESTIONS****FILL IN THE BLANKS**

- \_\_\_\_\_ and \_\_\_\_\_ are used for storing the information in an organized manner.
- A file is opened in memory using the \_\_\_\_\_ function.
- To open a file in output mode \_\_\_\_\_ prefix is used.
- The two types of data files are \_\_\_\_\_ and \_\_\_\_\_.
- The \_\_\_\_\_ and \_\_\_\_\_ functions read a specific number of bytes from a file.
- \_\_\_\_\_ function is used to create the directory.
- \_\_\_\_\_ function is used to open the directory.
- After the directory is open it can be read by using the \_\_\_\_\_ function.
- To test whether the file exists or not \_\_\_\_\_ operator is used.

**TRUE OR FALSE**

- To close the file, the following command must be given: close(FILEHANDLE).
- Binary files contain data that do not have lines ending with newline character, nor does the data consist of printable characters.
- To remove the directory rmdir() function is used.
- To read the directory back from the beginning call the rewinddir(DIRHANDLE) function.

**HANDS ON EXERCISES**

- Write a PERL program to open the file test-file1.txt. If the file is opened successfully then print the message ("You are Successful in opening file") otherwise show an error message("cannot find file") if the file is not found.
- Write a PERL program to write to the text file opened above. The text is "This is good progress".
- Write a PERL program to create a file test-file2.txt. Write and append to it accepting input from the user.
- Write a PERL program to create a directory called "Student\_Dir".

## 18. REGULAR EXPRESSIONS

Many programming tasks involve processing text data. For example, locating lines that contain a specific text pattern, replacing one string with another, and so on. **Regular expression** is a term for a **text pattern**.

It is a sequence of characters that use concise syntax to describe a large number of fixed patterns. For example, A.E indicates any 3-character sequence that begins with A and ends with E. Thus, A.E matches ACE, AcE, APE and ALE. A word processing search-and-replace operation is analogous to pattern matching.

### LEARNING BASIC REGULAR EXPRESSIONS

#### Understanding The Basic Form Of A Regular Expression

A **pattern** is a sequence of characters such as **s i o n**. The exact sequence of characters is all that is needed to find a fixed pattern. Often, however, there is need to specify a set of patterns, such as all words ending with the character sequence of **s i o n**.

A Regular expression enables describing such pattern sets in concise manner. Special characters (such as **\*** and **\**) are used to extend the meaning of a pattern. These characters are also called Meta-characters because they modify the meaning of other characters.

To define a pattern that matches words ending with **s i o n**, the following components have to be concatenated:

- The **\b** sequence that matches beginning of a word
- The **\w** sequence that denotes any **word** character (in PERL, a **word** character is any alphanumeric character and the underscore)
- A **\*** to specify that the pattern has one or more word characters
- The sequence **s i o n**, which must be matched exactly
- Another **\b** sequence that matches the end of a word (**\b** matches any **word boundary** – both the **beginning** and the **end** of a word)

Thus **\b\w\*sion\b** is the regular expression that represents all words that end with the sequence **s i o n**. To use the regular expression, one must enclose the regular expression **inside** a pair of **slashes**. A regular expression is used as follows:

```
^/b\w*sion\b/
```

As can be seen from the above example, a typical regular expression has three types of elements:

- Anchors** that specify the location of the pattern in a line of text. e.g. the caret (^) is an anchor that matches the beginning of a line
- Character sets** that match one or more characters e.g. **The** is a character set that matches the exact sequence of characters **T, h** and **e**
- Modifiers** that specify how many times the preceding character (or character sets) are repeated e.g. the asterisk (\*) is a modifier that indicates that the preceding character should be repeated zero or more times

#### Anchoring Patterns

##### Matching The Start And End Of A Line

The caret (^) is the starting anchor and the dollar sign (\$) is the end anchor.

A regular expression, such as `^The`, matches any line that begins with the sequence `The`. Therefore `^The` matches any of the following lines:

```
The book you want is Using Open Source Products On Windows.
Then you should buy the book.
Therefore, I recommend Using Open Source Products On Windows.
```

It matches because all the three lines have the **first three characters = The**.

On, the other hand, if one wants to search for `Using Open Source Products On Windows`, at the **end** of a line. It takes the following form:

```
/"Using Open Source Products On Windows \." $ /
```

In this case,

- The first and the last slashes are the delimiters for the pattern match
- The dollar sign (\$) at the end anchors the pattern to the end of a line
- The rest of the pattern is simply the text pattern. It looks complicated because **backslashes** are added in front of characters that have special meaning. This is why the **period** and the **double quote** are preceded by backslashes

## Note



The `^` and `$` act as anchors only when they occur at the proper end of a regular expression, i.e. `^` is an anchor only if it is the first character of a regular expression. Similarly, `$` is an anchor marking the end of a line only when it is the last character of the expression.

If the `^` or `$` appear elsewhere in the regular expression, these characters have the same meanings as when they appear in a double-quoted string. Thus, the `$` is treated as a prefix for a scalar variable's name and `^` means a control character (viz. `^a` is `Ctrl+A`, and so on).

## Tip



If the characters `^` and `$` must be matched without interpreting them in any special way (even when they occur at the proper ends of a pattern), add a backslash (`\`) **prefix** to the characters. Thus to match `^`, use `^\^` and to match a `$` use `\\$`.

## Matching A Word Boundary

In, addition to anchoring patterns at the end of a line, one can specify patterns that begin at a **word boundary** i.e. at the beginning or end of a word. In PERL, word boundaries have the following meaning:

- A word begins when a word character occurs. In PERL, a word character is any alphanumeric character or an underscore
- A word ends when a non-word character appears following the word. A non-word character is anything but an alphanumeric character or an underscore

Thus, to look for the word **able** (but not words such as `capable` and `reliable`) the following expression can be used.

```
/\b able \b /
```

## Using Character Sets In Regular Expressions

The simplest character set is a single character. For example, the regular expression `/The/` contains three character sets, `T`, `h` and `e`. This pattern will match any occurrence of these three characters in a line of text. Thus, it matches lines containing the words, `The`, `There` and `Therefore`. To limit the match to the word `The`, add the word boundary anchor (`\b`) in the following manner:

```
\bThe\b/
```

## Specifying A Range Of Characters

In addition, to single characters, meta-characters can be used to specify a more complex character set. One such character set is a range of characters, such as the set of all lowercase letter, from **a** to **z**. Indicate this as follows:

- Place a hyphen between the first and the last characters to write the range. Thus lowercase letters become **a-z**.
- Place this range of characters inside a pair of square brackets. Thus the character set of lowercase letters is **[a-z]**.

To match any letter regardless of case, simply concatenate the two character ranges in square brackets as follows:

**[a-zA-Z]**

Another range of characters is **[0-9]**, which denotes a single digit.

Inside, the square brackets, ranges of characters can be intermixed with explicit characters.

For example, PERL's word characters belong to the set **[A-Za-z0-9\_]**, which means a letter, a digit or an underscore.

The following are some examples of regular expressions that use character sets.

Regular Expression	Matches
/a-zA-Z/	Any letter
/b[aeiou]g/	Any of the sequences bag, beg, big, bog and bug
/[A-Z][A-Z]/	Any two uppercase letters next to each other
/[a-z]*/	Zero or more lowercase letters
/[567]/	One of the numbers 5,6 or 7

## Excluding Some Characters

So far, character sets have been specified with characters to be matched. Character sets that **exclude** certain characters can also be defined. All characters will be matched except those inside the square brackets by putting a caret (^) as the first character after the left bracket. For example, to match any character other than a digit, one can use the following regular expression:

**/[^0-9]/**

Thus, to pick out lines of text that do not contain any digits, use this pattern:

**/[^0-9]\*\$/**

Here, the first caret (^) matches the beginning of the line. Then, **[^0-9]\*** matches zero or more non-numeric characters. Finally, the dollar sign (\$) matches the end of a line. The net effect is that the pattern matches any line that does not contain a digit.

## Note



The caret (^) has a different meaning when it appears as the first character inside a square bracket, as opposed to when it appears as the first character of a regular expression.

## Matching Any Character

One can use the **dot** or the **period** (**.**), to match any *single* character. For example, a pattern such as **/A ./** matches **A** followed by any character.

### Repeating A Character Set

After anchors and character sets in regular expressions, the third type of element is the *modifier*. The *asterisk* (\*) is a modifier that means zero or more copies of the preceding characters. For example to match zero or more occurrences of the letter *x* the regular expression is as follows: `/x*/`

Similarly, to match one or more occurrences of a character set there is a modifier the plus sign (+). For example to match one or more digits the regular expression is as follows:

```
/[0-9]+/
```

### Matching A Specific Number Of Character Sets

Sometimes specific number of characters needs to be matched. For example, to match a pin code in India exactly six digits must be matched. The \* modifier cannot be used since it cannot specify a maximum number of characters. In this case, use another special pattern that allows the specification of the minimum and maximum number of characters to match. To do this, the pattern takes the form of two comma separated numbers enclosed in curly braces. For example, to match at least five and at most nine digits, the expression is as follows:

```
^b[0-9]{5,9}\b/
```

The {5,9} sequence means that the digit must occur at least 5 times but no more than 9 times.

Match exactly 5 digits in the following manner:

```
/[0-9]{5}/
```

### Matching One Fixed Sequence Or Another

Sometimes, one of several fixed text sequences may have to be matched, To do this simply list the text sequences separated by vertical bars (|). The following regular expression matches any of the U.S. state abbreviations MA, MD, VA or CA:

```
/MA|MD|VA|CA/
```

### Matching Special Characters

In addition to matching specific characters and character sets, special characters such as tab or whitespace may have to be matched. PERL considers space, tab, newline, carriage return and form feed whitespace characters.

Following is the list of backslashed letters with special meaning in regular expressions

Character	Meaning
<code>\d</code>	Any digit
<code>\t</code>	Tab
<code>\e</code>	Escape
<code>\w</code>	Word character
<code>\s</code>	A white space character (space, tab, newline, carriage return or form feed)

Character	Meaning
<code>\D</code>	Anything other than a digit
<code>\f</code>	Form feed
<code>\cX</code>	Ctrl+X, where X is any character
<code>\W</code>	Anything but a word character
<code>\S</code>	Anything but a white space character

#### Example 1: (Refer diagram 18.1)

To find the number of times the word PERL exists in a file then prints this as output. First create a text file with the word PERL mentioned in it a couple of times.

```
#!/c:/perl/bin/perl.exe -w
unless(open(passwd1,"PERLwd.txt")) {
    print"Cannot open file \n";    }
while(($line=<passwd1>)) {
    @users=split(" ",$line);    }
```

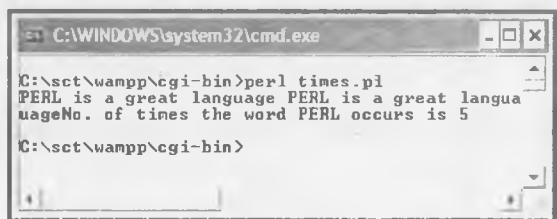


Diagram 18.1: The times.pl output in command window

```
print "@users";
$cnt=0;
for($x=0; $x<=$#users; $x++) {
    if(($users[$x] eq "PERL") || ($users[$x] eq "PERL ") || ($users[$x] eq "PERL.")) {
        $cnt=$cnt+1;    }
    }
print "No. of times the word PERL occurs is $cnt\n";
```

### Finding Lines That Do Not Match

To print lines that do not contain a pattern, all that is required is to add the logical NOT operator (denoted by !) as follows:

```
while (<>) {
    print if ![J]java; }
```

### The while (<>) File-Processing Code

The simple syntax for processing your command-line variables as a file list is: While(<>)

This code opens, reads, and closes files. It works like this:

- The @ARGV array is shifted one element at a time into the \$ARGV variable
- The \$ARGV variable is used to open the file handle ARGV
- Each line of the file associated with ARGV is read, one line at a time, into the special variable \$\_
- When the entire file has been read, the ARGV file handle is closed and the first step is repeated.

### MATCHING PATTERNS IN ANY STRING

When a regular expression is placed inside a pair of forward slashes (/ . . /), PERL matches the pattern with the current contents of the special variable \$\_. Sometimes however, a match has to be looked for in another string variable. For this PERL provides the pattern binding operators =~ and !~

#### Example 2:

To look for a match, write the following:

```
if ($string =~ /regex/) {
    print " Valid Id \n";    # Pattern match    }
```

where **regex** is the regular expression for the pattern and \$string is the variable in which PERL should look for the match of the pattern

Similarly, a variable can be checked whether it does not match a pattern, use the !~ operator, as follows:

```
if ($string !~ /regex/) {
    print " Invalid id \n";    # No pattern match    }
```

### Modifying The Pattern-Matching Criteria

#### Making The Pattern Match Case-Insensitive

Often, a string may be searched for without regard to case. This can be done by appending the **i** modifier to the pattern as follows;

```
^b image \b/i
```

This pattern will match all variations of the word image e.g image, IMAGE or even iMaGe.

#### Finding All Occurrences Of A Pattern

Just as by appending **i** to make a pattern match case-insensitive, a pattern match can be made global i.e. the match finds all occurrences, by appending a **g** to the pattern matching expression.

Thus to match all occurrences of the word Java or java the expression is as follows:

```
/\b[Jj]ava\b/g
```

## Note



To count all the matches in each line, one can use the global pattern matching expression occurrences `/\b[Jj]ava\b/g` in a list context. Simply assign the pattern to an array variable as follows:

```
@matches = /\b[Jj]ava\b/g;
```

Then the array `@matches` will contain all substrings matched by the pattern. To count the occurrences, all that is to be done is to add to `$count` the number of items in `@matches` as follows:

```
$count += @matches;
```

## Replacing A Pattern

Sometimes, after searching for a pattern, it must be replaced with a different string. PERL provides a **pattern substitution** operator that performs this task.

## Referring To A Previous Match

Sometimes, when substituting a pattern, there is a need to refer to the exact substring that had just matched a pattern specified. This need arises specifically when writing PERL programs to handle form input on the Web. When the PERL program receives information, many special characters are provided in an encoded form. For example, a tilde `~` appears as `%7e` and a space is encoded as `%20`. Each special character is replaced with a percent sign (`%`) followed by a two digit hexadecimal number representing the ASCII for that symbol.

The PERL program has to locate each occurrence of `%` followed by a two digit hexadecimal digits and replace those three characters with a single character whose ASCII code matches the hexadecimal digits. Given the hexadecimal digits, the character can be found by using `pack()`. This is done as follows:

**Example 3:** (Refer diagram 18.2)

```
#!/c:/perl/bin/perl.exe -w
$hexdigits = "7e";
$char = pack("c", hex($hexdigits));
print "ASCII code $hexdigits correspond to $char\n";
```

To find two hexadecimal digits that follow the `%` and use them in the argument of `pack()`. Do the following:

```
s/%( . .)/pack("c", hex($1))/ge;
```

As can be seen the pattern to match is written as `%( ..)`, which means `%` followed by two digits. Because this pattern is in parenthesis, after a match one can refer to the two digits with the special variable `$1`. Then the substitution string uses the `pack` function within which `$1` appears.

```

C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl pack.pl
ASCII code 7e correspond to ~
C:\sct\wampp\cgi-bin>

```

Diagram 18.2: The `pack.pl` output in command window

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. A \_\_\_\_\_ is used to match a sequence of characters.
2. The three types of elements of a regular expression are \_\_\_\_\_, \_\_\_\_\_ and \_\_\_\_\_.
3. The \_\_\_\_\_ symbol matches the start of a line.

4. The \_\_\_ character is added before characters that have special meaning.
5. To match one or more occurrences of a character set the \_\_\_ modifier is used.
6. To match a list of fixed text sequences, they must be separated by a \_\_\_ symbol.
7. The logical NOT operator is represented by the \_\_\_ symbol.
8. \_\_\_\_\_ is known as the pattern-binding operator.
9. The \_\_\_ character matches any single character.

### TRUE OR FALSE

10. To match the characters ^ and \$ one must add the "/" prefix to these characters.
11. To make a global substitution one must append a "g" to the pattern.
12. The "\w" character means any white space character.
13. The "\*" modifier means one or more copies of the preceding characters.
14. "\b" is known as a word boundary.

### **HANDS ON EXERCISES**

1. Write a program, which will take in a string and check for the pattern 'Hello'. If found display an appropriate message for Uppercase, Lowercase or Proper case and also an appropriate message when match not found.
2. Define a variable containing a string. Match the pattern, print a message when there is a:
  - Case sensitive match
  - Case insensitive match



## 19. CREATING STRUCTURED PROGRAMS

### PACKING CODE IN SUBROUTINES

#### What Are Subroutines?

A **subroutine** is a block of PERL statements with a name. A subroutine groups together PERL statements to perform a specific task. That task can then be performed anywhere in a PERL program by invoking the subroutine by its name.

#### Creating A Simple Subroutine

The structure of a subroutine is simple. The **sub** keyword has to be used to define a subroutine. The **name** of the subroutine comes after the sub. The body of the subroutine is a block of PERL statements enclosed in curly braces ({ ... }).

#### Syntax:

```
sub <subroutine_name> {  
    ...block of statements...; }  
}
```

For example, the following is a subroutine that prints a message:

```
sub message { print "Hello, there. \n"; }
```

#### Invoking A Simple Subroutine

To invoke the **message** subroutine, the subroutine's name has to be written with an ampersand (&) prefixed to it.

```
&message;
```

This causes the subroutine's statement block to execute, and the resultant output is:

```
Hello, there.
```

### PASSING ARGUMENTS TO SUBROUTINES

#### Subroutine Arguments

When a subroutine is invoked, a list of values can be placed inside the parentheses following the subroutine's name. These are the subroutine's **arguments**.

Most subroutines accept arguments and produce a result based on the argument provided. In most programming languages, the syntax for subroutine definition requires the specification of arguments that a subroutine accepts. In PERL, the arguments are **not defined** in the subroutine. Instead, PERL passes arguments to a **special array variable** named `@_`.

For example, call the **message** subroutine as follows:

```
&message ("Ivan", "Bayross");
```

The result is still the same "Hello, there." because the message subroutine is not written to handle arguments.

When a subroutine is called with an argument list, PERL places this list in a special array named `@_`. Thus, arguments can be handled in a subroutine by accessing and using the contents of the `@_` array. To use the `@_` array, the message subroutine must be re-written as follows:

```
sub message {      print "Hello, there. @_.\n";      }
```

The print statement now includes the `@_` array as a parameter and the resulting output will be the following:

```
Hello, there. Ivan Bayross.
```

## Initializing Variables From Arguments

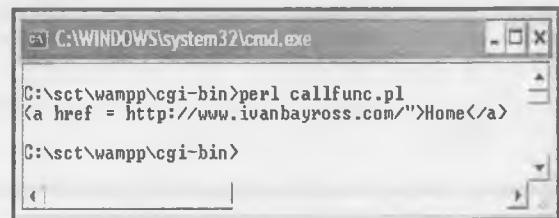
Instead of working with the elements of the `@_` array, it's more readable if variables can be defined with meaningful names and initialized from an argument, then used in a subroutine. Suppose a subroutine expects a single argument a scalar variable with the web server's name. A variable can then be declared and initialized from the `@_` array as follows:

```
my($hostname) = @_;
```

This initializes the `$hostname` variable with the first element in the array `@_`.

**Example 1:** (Refer diagram 19.1)

```
#!/c:/perl/bin/perl.exe -w
sub MakeLink {
# Expects hostname of server
    my($hostname) = @_;
    print "<A HRef = http://$hostname/>Home</A>\n";
}
# Sample call
MakeLink ("www.ivanbayross.com");
```



**Diagram 19.1:** The `callfunc.pl` output in command window

If the subroutine expects **several scalar** values, they can be initialized in a similar manner. Suppose a subroutine is written that expects **two** arguments, a last name and an identifier, the variables might then be declared and initialized as follows:

```
my($lastname, $id) = @_;
```

## Declaring Local Variables

As variables are **global** in PERL, whatever is done to a variable inside a subroutine affects the variable with the same name anywhere else in the program. To prevent this, a **local** variable should be used inside a subroutine.

As illustrated in the above example, `my()` enables the declaration of variable names local to the subroutine. Such local variables can be used inside the subroutine without worrying whether the name conflicts with other variables already in use outside the subroutine with the same name.

PERL had one way to declare a local variable i.e. using `local()` as shown below:

```
local($arg);
```

The net effect of `local()` is the same as `my()`. The `$arg` variable becomes local to the subroutine, and anything done to the variable inside the subroutine does not affect any variable named `$arg` that exists outside the subroutine.

### Understanding Argument Passing

PERL passes arguments to a special array. PERL subroutines can by design, accept any number of arguments. It's up to the subroutine to use these arguments appropriately. For example, if a subroutine does not expect arguments, it **simply ignores** any argument one provides when the subroutine is called. One should know what a subroutine does and what arguments it expects before one calls it in the program.

**Example 2:** (Refer diagram 19.2)

```
#!c:/perl/bin/perl.exe -w
sub printarg {
# Declare a variable local to the
subroutine using my function
    my ($arg);
    if (@_) {
        print "Here are the arguments:\n";
        foreach $arg (@_) {
            print "$arg\n";
        }
    }
    else {
        print "No arguments\n";
    }
    print "_____ \n";
}
```

```
# Call the subroutine with various arguments
&printarg;
&printarg(1, 2, 3);
&printarg("Chhaya", "Hansel", "Mamta", "Sharaman");
# If the subroutine is defined above,
# the subroutine can be call without the &
printarg "price", 24.99;
printarg 100, 200, 300;
```

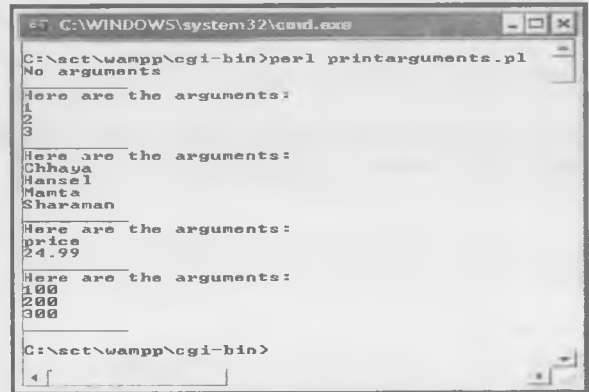


Diagram 19.2: The printarguments.pl output in command window

In above example, the **printarg** subroutine prints the contents of the **@\_** array that contains the arguments passed to the subroutine.

### Modifying The Argument Value

If the **@\_** array is directly manipulated inside the subroutine, any changes made to the values in **@\_** do affect the arguments passed during the subroutine call. Sometimes this may be a desirable way to implement a subroutine. An example might be a subroutine that converts its arguments into lowercase. The conversion is performed in-place, which means, the subroutine makes the change directly in the **@\_** array.

**Example 3:** (Refer diagram 19.3)

The **MakeLC** subroutine in the following program demonstrates how this can be done:

```
#!c:/perl/bin/perl.exe -w
sub MakeLC {
    foreach (@_) {
        # Converts $_ into lowercase
        tr/A-Z/a-z;
    }
}
$string = "I WANT THIS IN LOWERCASE!";
```

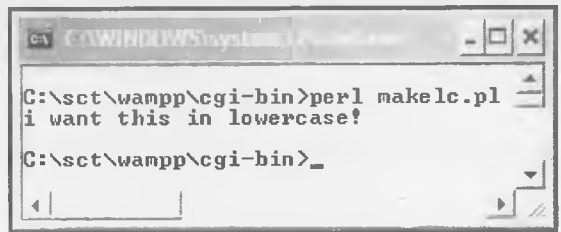


Diagram 19.3: The makelc.pl output in command window

```
&MakeLC($string);
print "$string\n";
```

The `MakeLC` subroutine modifies the argument that was passed to it.

**Example 4:** (Refer diagram 19.4)

Using a special type of local variable called **typeglob** with the `MakeLC_New` subroutine.

```
#!/c:/perl/bin/perl.exe -w
sub MakeLC_New {
# Use a special form of local variable called "typeglob"
    local(*input) = @_;
    $input =~ tr/A-Z/a-z;
}
$newstring = "CHANGE THIS TO LOWERCASE!";
&MakeLC_New(*newstring);
print "$newstring\n";
```

The revised subroutine `MakeLC_New` converts its argument to lowercase. This version, however, has the following key differences from the preceding subroutine, which directly manipulated the `@_` array.

- ❑ The **local** variable is declared with a **\*** prefix, as: `local (*input) = @_;`  
This variable is known as a **typeglob** and it works only with `local()`. The net effect of the typeglob is that this local variable matches whatever actual variables are passed in an argument, be it a scalar, an index array or an associative array.
- ❑ Inside the subroutine, the local variables have to be used in its expected form. Thus, if the subroutine expects a scalar variable typeglob `*input` is used, it has to be referred to with the symbol `$input`.
- ❑ When the subroutine is called, pass the argument in typeglob form. Replace the letter at the beginning of the variable name (such as `$`, `@`, or `%`) with an asterisk (`*`), like: `& MakeLC_New(*newstring);`  
Here, `$newstring` is the variable actually passed to the subroutine.

When any subroutine call is seen with arguments that have an asterisk prefix, it means the subroutine directly modifies the argument that's being passed.

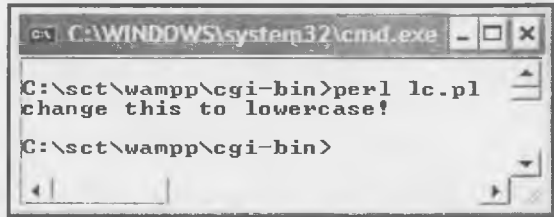
## Passing A Filehandle To A Subroutine

Another interesting use for the **typeglob** variable format is to pass a **filehandle** to a subroutine. In fact filehandles cannot be passed to PERL subroutines **without using** this special approach, because PERL treats filehandles differently from other variables.

Suppose a filehandle needs to be passed to a subroutine, which will read from the file (or write to it). Here is a very simple program that passes the `STDIN` filehandle to a subroutine, and the subroutine reads a line from its filehandle argument:

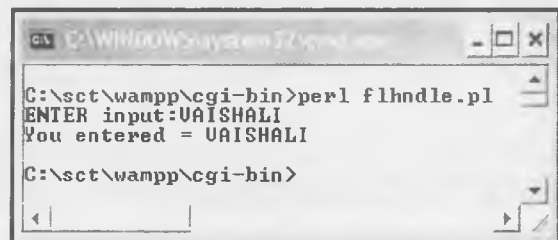
**Example 5:** (Refer diagram 19.5)

```
#!/c:/perl/bin/perl.exe -w
sub readline {
# Use typeglob to get the filehandle, where $_[0] is simply
# the first element of the @_array that contains the arguments
    local(*FHANDLE) = $_[ 0 ];
    local $line;
```



```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl lc.pl
change this to lowercase!
C:\sct\wampp\cgi-bin>
```

Diagram 19.4: The `lc.pl` output in command window



```
C:\WINDOWS\system32\cmd.exe
C:\sct\wampp\cgi-bin>perl flhndle.pl
ENTER input:VAISHALI
You entered = VAISHALI
C:\sct\wampp\cgi-bin>
```

Diagram 19.5: The `flhndle.pl` output in command window

```
# Read a line from the filehandle assuming it don't have any
# special characters such as $ or @ as prefix
$line = <FHANDLE>;
}
print "ENTER input:";
$line = &readline(*STDIN); # Note the * prefix
print "You entered = $line";
```

The key features of the **readline** subroutine and how it's used:

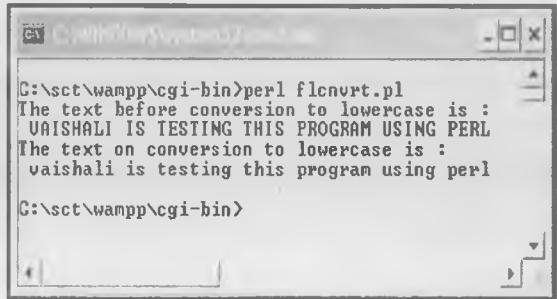
- The local filehandle is declared as a **typeglob** consisting of an asterisk followed by the name:  
local(\*FHANDLE) = \$\_[0];
- To use the filehandle in the subroutine, use the plain name without any special character prefix, as follows:  
\$line = <FHANDLE>;
- In the subroutine call, pass the filehandle argument with an asterisk prefix. Thus, to pass STDIN to the subroutine, write:  
\$line = readline(\*STDIN);

This program also returns a value by placing the return value as the last expression in the subroutine's body. The next section covers the subject of returning values from subroutines.

In the following example, the subroutine **conlc** converts the contents of a text file from Uppercase to Lowercase. A text file (**password1.txt**) is created with contents in Uppercase and then passed to the subroutine for conversion.

**Example 6:** (Refer diagram 19.6)

```
#!c:/perl/bin/perl.exe -w
sub readline {
    local (*FHND)=$_[0];
    local $line;
    $line=<FHND>;
}
sub conlc {
    foreach (@_) {
        tr/A-Z/a-z;
    }
}
unless (open(FILE,"password.txt")) {
    exit;
}
$string= &readline(*FILE);
print "The text before conversion to lowercase is :\n $string\n";
&conlc($string);
print "The text on conversion to lowercase is :\n $string\n";
```



**Diagram 19.6:** The **flcnvrt.pl** output in command window

### Returning Values From Subroutines

PERL has a simple scheme for returning values from a subroutine. PERL returns the value of the last expression in the subroutine. To explicitly return a value from a PERL subroutine the **return()** function can be used.

### Returning The Last Expression

PERL subroutines return values by making the last expression in the subroutine as a return value.

Example 7: (Refer diagram 19.7)

```
#!/c:/perl/bin/perl.exe -w
sub doubleit {
# Returns 2 times the argument
    my ($x) = $_[0];
    $x *= 2;
}
$y = doubleit (2);
print "Result = $y\n";
```

The last line in `doubleit()` simply computes the value. This becomes the return value of the subroutine.

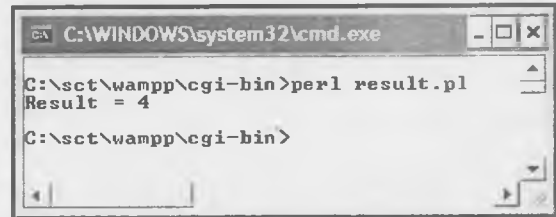


Diagram 19.7: The `result.pl` output in command window

## Using The Return Function

PERL includes a built-in function called `return()` that explicitly returns a value from any subroutine. Program code in C, C++, or Java uses the `return()` statement extensively.

### Syntax:

```
return(EXPRESSION);
```

where, `EXPRESSION` is a PERL expression containing the value to be returned. As in all PERL function calls, the parentheses are optional.

### Example 8:

```
#!/c:/perl/bin/perl.exe -w
sub doubleit {
    my ($x) = $_[0];
    $x *= 2;
    return ($x); # Return the result
}
}
```

## LIBRARIES

A library is nothing but a collection of subroutines, each responsible for performing tasks. The work that has to be done by these tasks is packaged into the subroutines code, which is then stored in a file. When required, those subroutines can be used in any PERL program by referring to the file where the subroutines are stored within the PERL program. This encourages (re-useable) modular programming.

### Creating PERL Libraries

To create a library in PERL there are simple guidelines to follow. Assume that a date is required formatted in a specific manner. A `get_date()` will return the date as a string in the required format. This subroutine can then be stored in a library to be called in any PERL program that may need the date and time in the format specified. For which, simply call the `require` function with the library filename as an argument. Following is the method to write the subroutine and then save it in a PERL Library named `date.pl`:

```
# File: date.pl
# Return current data and time as a string. Uses the localtime function
# Usage: require "date.pl";
#!/c:/perl/bin/perl.exe -w
sub get_date {
    local ($mday, $mon, $year, $yday) = (localtime)[3,4,5,6];
    local $day = (Sun, Mon, Tue, Wed, Thu, Fri, Sat)[$yday];
    local $month = (Jan, Feb, Mar, Apr, May, June, Jul, Aug, Sept, Oct, Nov, Dec)[$mon];
```

```

$year += 1900;      #, localtime subtracts 1900 from the year
$day, $month, $mday, $year;
}
1; # End the file with a 1 as the last expression

```

## Note



The only key point is that 1 must be included as the last expression in the subroutine file. The file can have more than one subroutine and may include other variables. Just place all the subroutine definitions one after another and **make sure** a 1 is at the very end of the file. The PERL interpreter uses the value of the last expression in the file as an indicator of success or failure. By making 1 the last expression, indicates that the subroutines in the file were successfully included.

## CGI PROGRAMMING WITH THE CGI-LIB.PL LIBRARY

Each CGI program is designed to handle information submitted by a user, by filling in a form. All form based data returned by a Web browser requires similar processing at the Web server to extract the fields and their values returned. To do this, a **common** set of subroutines can be used at the Web server. Since PERL is a popular programming environment, there is already a PERL library that does just this, called **cgi-lib.pl** created by Steven Brenner.

## Tip



Just as PERL is the accepted programming language for CGI programs, **cgi-lib.pl** is the accepted library for parsing the CGI query string that the Web browser sends to the Web server. By using the **cgi-lib.pl** library, PERL programs can be quickly created to handle HTML based form input.

## Obtaining cgi-lib.pl

The latest version of **cgi-lib.pl** can be downloaded from the official **cgi-lib.pl** homepage at the following URL:

<http://www.bio.cam.ac.uk/cgi-lib/>

Since **cgi-lib.pl** is a library of PERL subroutines in a text file, there is nothing much to install. All that needs to be done is to download the latest version of the file **cgi-lib.pl** and place it in the **/cgi-bin/** directory of the Web server.

## The cgi-lib.pl Includes The Following PERL Subroutines

Name	Description
ReadParse	Reads and parses the CGI query for both GET and POST methods. Places parsed result in associative array that you provide as an argument. Returns TRUE if there is a query, otherwise, returns FALSE.
PrintHeader	Prints the MIME header (Content-type:text/html) with an extra blank line indicating end-of-header
HtmlTop	Prints the top of the HTML document, including a title that you provide as argument
HtmlBot	Prints the bottom of the HTML document with the ending <code>&lt;/BODY&gt;</code> and <code>&lt;/HTML&gt;</code> tags.
MethGet	Returns true if the CGI query came as a GET request
MethPost	Returns true if the CGI query came as a POST request
MyURL	Returns a complete HTTP URL for your script (such as: <a href="http://www.someplace.com:8000/cgi-bin/dbquery.pl">http://www.someplace.com:8000/cgi-bin/dbquery.pl</a> )

Name	Description
CgiError	Prints an HTML document with an error message. If you provide an array of strings as an argument, the first string is used as a title and the rest are displayed in the body of the HTML document. If you do not provide any arguments, a generic error message is used as the body of the HTML document.
CgiDie	Calls CgiError and then the script quits with a call to the <i>die</i> function
PrintVariables	Prints the variables in the associative array that are passed as arguments (use this subroutine to print the fields and their values).

## Using cgi-lib.pl

It's straightforward to use subroutines from the cgi-lib.pl library. The general sequence of subroutine calls is as follows:

1. Call the ReadParse() subroutine to process the CGI query.
2. If ReadParse() returns TRUE, process the user information returned by a Web browser is now available in an associative array.
3. Output the desired HTML document by calling PrintHeader, HtmlTop, and HtmlBot.
4. Call CgiDie or CgiError to report any error conditions through an HTML page.

## IMPLEMENTING A FEEDBACK FORM

A common use of CGI programming is to solicit and accept feedback from users about the functionality of the website. The next few sections present a simple CGI program to handle user data, collected via a feedback form, served by the Web server to the user's Web browser. Once the form reaches the users Web browser the users fills in the form and submits it back to the Web server from where it came to be processed.

**feedback.cgi** is a PERL program that illustrates the use of the cgi-lib.pl library and cookies.

### Designing The Feedback Form

The first step is to create an HTML form through which a user can submit comments. The HTML source for the form could be as follows: (Refer diagram 19.8)

<HTML>

```
<HEAD><TITLE>User Feedback Form</TITLE></HEAD>
<BODY BgColor="#FFFFFF" Text="#000000">
  <H2>Thank You For Your Time </H2>
  <FORM Action="/cgi-bin/feedback.cgi" Method="POST">
    <P><STRONG>Comments:</STRONG><DL><DD>
      <TEXTAREA Name="Comments" Rows="5" Cols="42"></TEXTAREA>
    </DD></DL></P>
    <P><STRONG>Tell us how to get in touch with you:</STRONG><DL><DD><PRE>
      Name <INPUT Type="text" Size="35" Maxlength="256" Name="UserName">
      E-mail <INPUT Type="text" Size="35" Maxlength="256" Name="UserEmail">
      Tel <INPUT Type="text" Size="35" Maxlength="256" Name="UserTel">
      FAX <INPUT Type="text" Size="35" Maxlength="256" Name="UserFAX">
```

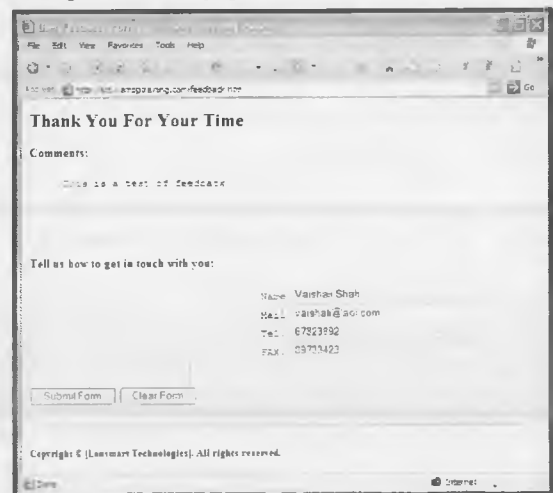


Diagram 19.8: The feedback.html output in I.E.



```

</PRE></DD></DL></P><P>
  <INPUT Type="submit" Value="Submit Form">
  <INPUT Type="reset" Value="Clear Form">
</P></FORM><HR>
  <H5>Copyright © [Lansmart Technologies]. All rights reserved.<BR></H5>
</BODY>
</HTML>

```

The **action** attribute of the `<form>` tag points to a script named **feedback.cgi** at the Lansmart Technologies web site.

The Web Server which services the Lansmart Technologies web site **requires** `/cgi-bin/` as its script directory, which is why the script's URL is `../cgi-bin/feedback.cgi`.

On most sites with NCSA and Apache Web servers, the PERL CGI scripts will reside in the `/cgi-bin/` directory.

**Note**



The `/cgi-bin/` directory name is an **alias** for a real directory on the Windows system, usually `c:\sct\wampp\cgi-bin\`.

The **method** attribute of the `<form>` tag specifies that the Web browser should send the form's data through a POST request.

**Processing User Feedback**

The **feedback.cgi** program that processes the data returned by the feedback form does the following:

- Reads and parses the form data returned to the Web server
- Writes (appends) user comments to a file called `comments.txt`
- Returns a **Thank-you** note to the user in the form of an HTML page

Here are some points to help design the CGI program in PERL:

- Include `cgi-lib.pl` in the program to parse the CGI input returned by the user's Web browser
- Call `ReadParse()` to parse the input
- Open the `comments.txt` file and append data appropriately into the file
- Spawn the HTML code that is the thank-you note returned to the user via PERL `codespec`

The following is the complete listing of **feedback.cgi** that handles input from a user's Web browser when it is returned to the Web server: (Refer diagram 19.9)

```

#!c:/perl/bin/perl.exe -w
require "cgi-lib.pl";
&ReadParse;
&PrintHeader;

```

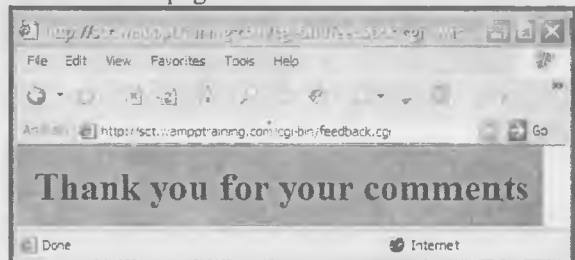


Diagram 19.9: The **feedback.cgi** output in I.E.

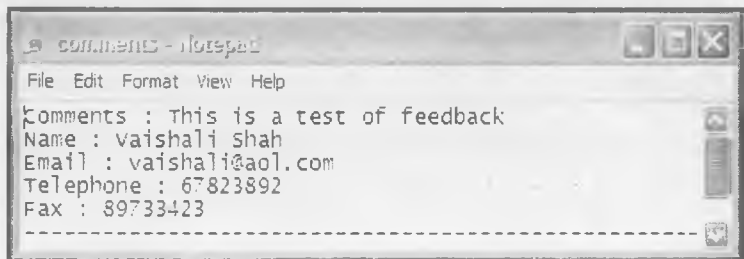


Diagram 19.10: The `comments.txt` file created by **feedback.cgi**.

```

$comments = ${Comments};
$username = ${UserName};
$useremail = ${UserEmail};
$userstel = ${UserTel};
$userfax = ${UserFax};
## Spawning the Thank-you note as HTML
print "<HTML>";
print "<BODY BGCOLOR=#5F9f9f>";
## Opening the text file comments.txt and loading it appropriately
open(COMMENTS,>>comments.txt)|| die print "<SCRIPT> alert('This file does not exist') </SCRIPT>";
print COMMENTS "Comments \: $comments\n";
print COMMENTS "Name \: $username\n";
print COMMENTS "Email \: $useremail\n";
print COMMENTS "Telephone \: $userstel\n";
print COMMENTS "Fax \: $userfax\n";
print COMMENTS "-----\n\n";
## Closing the text file comments.txt
close(COMMENTS);
## HTML continues
print "<CENTER>";
print "<H1> Thank you for your comments </H1>";
print "</CENTER>";
print "</BODY>";
print "</HTML>";

```

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. A \_\_\_\_\_ is a block of Perl statements with a name which are grouped together to perform a specific task.
2. The \_\_\_\_\_ keyword has to be used to define a subroutine.
3. When a subroutine is invoked a list of values can be placed in parentheses following the subroutine's name. These are the subroutine's \_\_\_\_\_.
4. \_\_\_\_\_ and \_\_\_\_\_ can be used to declare variables local in a subroutine.
5. \_\_\_\_\_ function can be explicitly used to return a value from a Perl subroutine.
6. \_\_\_\_\_ is a collection of subroutines that perform several tasks, which is stored in a file.
7. \_\_\_\_\_ is the accepted library for passing the CGI query string that the Web server sends to the CGI program.
8. \_\_\_\_\_ prints the variables in the associative array that are passed as arguments.
9. \_\_\_\_\_ reads and parses the CGI query for both GET and POST methods.

**TRUE OR FALSE**

10. To invoke a subroutine, the subroutine's name has to be prefixed with a percentage (%) sign.
11. When argument values are copied from the @\_array into a local variable, any changes made to the local variable will affect the values in the @\_array.
12. PrintHeader prints the MIME header with an extra blank line indicating end of header.
13. MethGet returns false if the CGI query came as a GET request.
14. MyURL returns a complete HTTP URL for the CGI script.

**HANDS ON EXERCISES**

1. Create a subroutine named 'message', which will print the greeting "Good Morning India". Then invoke it.
2. Modify the subroutine named 'message'. Pass a name as an argument to the subroutine. The subroutine should then print "Good Morning Name".
3. Create a subroutine to calculate the tax on income @ 30% of income. Pass the income as an argument to the subroutine. The subroutine should then return the tax liability on that income.

## 20. GOING THE OBJECT WAY WITH PERL

This material provides an introduction to new PERL features. In particular, the new reference variable enables the creation of useful data structures. PERL supports object-oriented programming through its modules.

### UNDERSTANDING REFERENCES

In PERL, there was no way of creating a variable that refers to another variable. Sometimes, however, it's convenient to have a variable that simply points to another variable. This helps to create useful data structures, such as a linked list. The diagram 20.1 given below illustrates how a linked list needs a variable that points to the next item in the link.

As shown in diagram 20.1, a linked list consists of a number of data structures, where each structure has a link to the next structure. For example, in a PERL program, the structure may be a simple list of scalar variables (think of an employee structure that has fields such as name, social security number, and so forth).

However, the variable that acts as the link to the next structure needs to refer to another structure. This is where a variable is referenced. A **reference** is nothing more than a scalar value that refers to another variable.

With the introduction of the reference type, PERL adds some new syntax to initialize a reference and to access the variable that's referred to by the reference

#### Defining A Reference

A **reference** is a **scalar** variable. Therefore, a reference looks just like any other scalar variable. What's new is the way a value is assigned to a reference.

The syntax for defining a reference variable is quite simple. Suppose there is a scalar variable named `$x`, a reference variable can then be initialized that points to the `$x` variable, as follows:

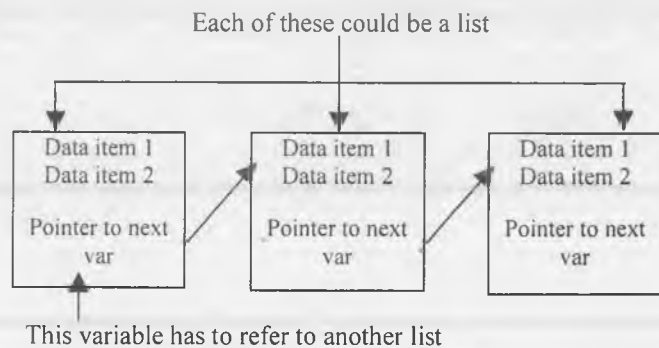
```
$x = 24.99;  
$rx = \ $x; # A backslash prefix indicates reference
```

In this case, `$rx` is a reference variable that points to the variable `$x`. Any variable can be referred to by adding a **backslash prefix** to the variable's **name**.

The value of `$x` can also be accessed through the reference variable `$rx`. This is known as **dereferencing**. The syntax is:

```
$y = $$rx;
```

This statement sets `$y` to the value of the variable being referred to by `$rx`. Thus, a reference variable can be de-referenced by **adding an extra \$ prefix** to the reference variable's **name**.



**Diagram 20.1:** Linked lists need a reference variable type

*Tip*



Referencing and dereferencing can be done not only to scalar variables but also to other variables such as indexed array and associative array using the following syntax:

```
$r_array = \@array;
@y = @{$r_array};
$r_hash = \%hash;
%z = %{r_hash};
```

In addition to this, a reference can be made to a subroutine and also to another reference using following syntax:

```
$r_proc = \$process_date;
&{$r_proc}("arg");
$r_ref = \$ref; # $ref is a reference
```

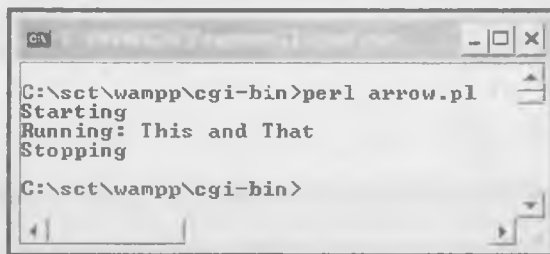
**Using The Arrow Operator To Dereference**

The arrow operator (a **minus (-)** sign followed by the **greater than (>)** sign) is actually a way to dereference a reference to an array or a hash. PERL implements objects as a reference to a hash, that's why an arrow operator has to be used to call a subroutine in an object.

**Example 1:** (Refer diagram 20.2)

**#!c:/perl/bin/perl.exe -w**

```
# Calling subroutines via hash reference, & defining some subroutines
sub start { print "Starting\n"; }
sub run { print "Running: @_ \n"; }
sub stop { print "Stopping\n"; }
# Define a hash with subroutine references
%SUBS = ('Start' => \&start, 'Run' => \&run, 'Stop' => \&stop);
# Define a reference to the %SUBS hash
$Object = \%SUBS;
# Now call subroutines via the hash reference
&{$Object -> {Start}};
&{$Object -> {Run}} ("This", "and", "That");
&{$Object -> {Stop}};
```



**Diagram 20.2:** The arrow.pl output in command window

Here are some points to understand the above program:

- The program begins with the definition of three subroutines, **start**, **run** and **stop** that simply print specific text messages.
- Then it defines a hash (associative array) named %SUBS that associates text labels with references to the subroutines, as follows: **%SUBS = ('Start' => \&start, 'Run' => \&run, 'Stop' => \&stop);** Thus a reference can be made to the start subroutine by writing \$SUBS{Start}, and the subroutine can be called with the syntax **&{\$SUBS{Start}}**.
- To show how to use the arrow operator to access a hash reference, a reference to %SUBS is defined as follows: **\$Object = \%SUBS;** (It is called as \$Object, because a PERL object is a reference to a hash)
- After the hash reference, \$Object, is defined, it is used to access the subroutines by name and call them. For example, the call to the start subroutine is as follows: **&{\$Object -> {Start}};**

To interpret this expression, first focus on `$Object ->{Start}`. Here the arrow (`->`) operator de-references `$Object` (remember, it's a reference to the hash `%SUBS`) and accesses the item associated with the `Start` key. The item accessed by `$Object->{Start}` happens to be the reference to the `start` subroutine, which is called by adding an ampersand prefix.

If a subroutine needs arguments, the argument list can be appended, as follows:

```
&($Object ->{Run})("This", "and", and "That");
```

For object-oriented programming, PERL provides a few additional features that result in a simpler syntax for using subroutines defined in an object. In addition to use of references, PERL objects rely on modules, which are basically collections of subroutines and variables.

## Understanding PERL Packages And Modules

A PERL package is a way to group together data and subroutines. Essentially, it's a way to use variable and subroutine names without conflicting with any names used in other parts of a program.

The package provides a way to control a `namespace`, the collection of variable and subroutine names. When you write a PERL program, it automatically belongs to a package named `main`. Besides `main`, there are other PERL packages in the PERL library (that's in the `lib` subdirectory of the PERL5 installation directory).

## PERL Packages

A PERL package is a convenient way to organize a set of related PERL subroutines. Another benefit is that variable and subroutine names defined in a package do not conflict with names used elsewhere in the program. Thus, a variable named `$count` in one package remains unique to that package and does not conflict with a `Count` variable name used elsewhere in a PERL program.

A PERL package is in a single file. The `package` statement is used at the beginning of the file to declare the file as a package and to give a package name. For example, the file `timelocal.pl` defines a number of subroutines and variables in a package named `timelocal.pl`. The `timelocal.pl` file has the following package statement in various places:

```
package timelocal;
```

This effect of this package declaration is that all subsequent variable and subroutine names are considered to be in the `timelocal` package. Such a `package` statement can be put at the very beginning of the file that implements the package.

PERL provides the following syntax for referring to a variable in another package:

```
$Package : : Variable
```

where, `Package` is the name of the package and `Variable` is the name of the variable in that package. If the package name is omitted, PERL assumes that a reference is being made to a variable in the `main` package.

To use a package in the program, simply call `require()` with the package filename as an argument. For instance, there is a package named `date` defined in the file `date.pl`. That package includes the `get_date()` subroutine, which converts a binary date into a string. Here is a simple program that uses the date package from the `date.pl` file:

```
#!/c:/perl/bin/perl.exe -w
# Use the date package defined in date.pl file
require "date.pl";
# Call the get_date subroutine
$mydate = date::get_date( );
# Print the date string
print "mydate";
```

## PERL Modules

PERL takes the concept of package one step further and introduces a **module**, which is a package that follows certain guidelines and is designed to be **reusable**. Each module is a package that is defined in a file with the same name as the package, but with a **.pm** extension. Each PERL object is implemented as a module. For example, the CGI object is implemented as the CGI module, stored in the file named CGI.pm.

### Using A Module

The subroutine **require()** or **use()** is called to include a PERL module in the program. For example, there is a PERL module named **Cwd** (defined, as expected, in the **Cwd.pm** file) that provides a **getcwd()** subroutine that returns the current directory. **require()** can be called to include the **Cwd** module. Then call **getcwd()** as follows:

```
require Cwd;           # No need for .pm extension
$curdir = Cwd::getcwd ();
print "Current directory = $curdir\n";
```

The first line brings the **Cwd.pm** file into this program. It is not required to specify the file extension because **require()** automatically appends **.pm** to the module's name to figure out which file to include. The second line shows how to call a subroutine from the **Cwd** module. When **require()** is used to include a module, each subroutine needs to be invoked with the **Module :: subroutine** format.

If this sample program is rewritten with **use()** in place of **require()**, it takes the following form:

```
use Cwd;
$curdir = getcwd (); # No need for Cwd::prefix
print "Current directory = $curdir\n";
```

The most significant difference is that it is no longer required to qualify a subroutine name with the module name prefix (such as **Cwd::**).

### Note



The differences between **require** and **use**

- When a module is included by calling **require()**, the module is included only when **require()** is invoked as the program runs. The **Module :: subroutine** syntax must be used to invoke any subroutines from a module included with the **require** function.
- When a module is included by calling **use()**, the module is included in the program as soon as the **use** statement is processed. Thus, subroutines and variables can be invoked from the module if they were part of the program. It is **not** necessary to qualify subroutine and variable names with a **Module::** prefix.

## USING OBJECTS IN PERL

An **object** is a data structure together with the functions that operate on that data. Each object is an instance of a **class** that defines the type of the object.

For example, a rectangle class may have four corners of the rectangle as data and functions, such as one that computes the rectangle's area and another that draws the rectangle. Then each rectangle object can be an instance of the rectangle class with different coordinates for the four corners. It's in this sense that an object is an instance of a class.

The functions (or subroutines) that implement various operations on an object's data are known as its **methods**.

Classes also have the notion of **inheritance**. A new class of objects can be created by extending the data or methods (or both) of an existing class. A common use of inheritance is to express the IS A relationship among various classes of objects. Consider, for example, the geometric shapes. Because a circle IS A shape and a rectangle IS A shape, the circle and rectangle classes inherit from the shape class. In this case, the shape class is called a **parent class**, or **base class**.

## Note



The basic idea behind object-oriented programming is that the data and the associated methods (subroutines) of an object can be packaged as a black box. Programmers access the object only through its advertised methods, without having to know the inner workings of the methods. Typically, a programmer can create an object, invoke its methods to get or set attributes and destroy the object.

## Understanding PERL Objects

PERL implements objects using modules, which package data and subroutines in a file. PERL presents the following simple model of objects:

- An **object** is denoted by a reference (objects are implemented as references to a hash)
- A **class** is a PERL module that provides the methods to work with the object
- A **method** is a PERL subroutine that expects the object reference as the first argument

Programmers who implement objects have to follow certain rules and provide certain methods in a module that represents a class.

## CREATING AND ACCESSING PERL OBJECTS

As the name implies, the CGI object is meant for writing Common Gateway Interface applications for World Wide Web Servers. A CGI program accepts queries submitted by a user and writes back an HTML document in response.

When a CGI object is created, it automatically parses a query string submitted by the user via an HTML form. The CGI object provides methods to access the parameters entered by the user on a form, creates headers needed for Web pages, and generates HTML code for the Web page that will be sent back by the CGI program.

To use the CGI object, the following steps need to be implemented:

1. Place the following line to include the CGI module in the program: use CGI;  
This line must be included before you create a CGI object.
2. To create a CGI object, use the following syntax: \$query = new CGI;  
Where \$query is the reference to the CGI object. In the case of the CGI object, creating the object automatically parses the query and sets up the internal variables of the object.
3. Invoke methods from the CGI objects, as illustrated by the following examples:

```
print $query->header;    // Send the HTTP header
print $query->start_html("Title of document");
print $query->end_html;  // End HTML document
```

Here, **\$query->header** calls the **header** method of the \$query CGI object. Similarly, **start\_html()** and **end\_html()** are methods in the CGI object. All of these methods return strings, which is why they are used as arguments to the **print** function.

The object's methods can be accessed by using the **arrow operator** and the object reference that is obtained after creating the object.



## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. A \_\_\_\_\_ is a variable that simply points to another variable.
2. A variable can be referred to by adding a \_\_\_\_\_ prefix to the variable's name.
3. A reference can be de-referenced by adding an extra \_\_\_ prefix to the reference variable's name.
4. The \_\_\_ operator is used to de-reference a reference.
5. A \_\_\_\_\_ in PERL is a way to group together data & subroutines.
6. Any program in PERL automatically belongs to the package named \_\_\_\_\_.
7. The \_\_\_\_\_ function is used to call a package in a program.
8. In addition to the require function the \_\_\_\_\_ function can be used to include a PERL module in a program.
9. A \_\_\_\_\_ is a PERL subroutine that expects the object reference as the first argument to implement it.

### TRUE OR FALSE

10. Variables defined in a package are global by default.
11. A module is a package with certain guidelines and which is designed to be re-usable.
12. When a module is invoked using the use function, the Module : : subroutine syntax must be used.
13. The CGI object provides methods to access the parameters entered by the user on a form.

## HANDS ON EXERCISES

1. Define a scalar variable named \$scalarVar with a value. Create a reference named \$scalarRef to \$scalarVar. Then print the contents of the reference.
2. Create the array @ weeks as follows:  
@weeks = qw (Sun Mon Tue Wed Thu Fri Sat);  
Create a reference to this @week and print out the contents of the array reference.
3. Create the array %who as follows:  
%who = ('Name' => 'John', 'Age' => 25, 'Height' => '182 cm', 'Weight' => '80 kg');  
Create a reference to an associative array %who and print out the contents of the array reference.
4. Create a subroutine. Create a reference to that subroutine and then de-reference the reference to call the subroutine.
5. Create a package with a subroutine. Call a subroutine of that package in a program.

## 21. DATABASE CONNECTIVITY

### DATABASE ACCESS USING PERL

The true power of the World Wide Web can be realized only through the development and deployment of Commercial Applications on the web. Currently, the most popular method of doing this is by using the Common Gateway Interface to communicate with some data storage system, use the information stored in that system and create HTML pages dynamically at run time.

The most popular language for CGI scripting purposes is PERL and the most popular data storage system currently in use is Oracle. An intermediate platform is required for PERL to connect and communicate with Oracle. In Windows environment this intermediate platform is found in the PERL WIN32::ODBC extension and Windows ODBC drivers. Using ODBC, PERL can communicate with any data storage system of choice.

### THE PERL WIN32::ODBC EXTENSION

The WIN32::ODBC module is an extension to Perl, which permits connectivity to a data storage system using ODBC drivers.

#### Creating an ODBC Object

The following steps need to be performed to use **WIN32::ODBC** in any Perl script:

```
use WIN32::ODBC;
# Create a new data connection to the DSN
$Data = new WIN32::ODBC("MyDSN");

# MyDSN can either be a DSN as defined in the ODBC Administrator or a
# Connect String such as "DSN=Database Name; UID=User Id; PWD=Password;"

# After processing is done, the data connection needs to be closed
$Data -> Close();
```

### ODBC OBJECT METHODS

#### The new Method.

##### Syntax:

```
new Win32::ODBC(<DSN>);
```

This method creates a new ODBC object for the given DSN. The parameter passed to this method can either be an already defined DSN or a properly formatted Connect String.

If this method is successfully executed, it will return a new ODBC object. In the event of an error, it will return **undef**.

#### Example 1:

```
#!c:/perl/bin/perl.exe -w
require "cgi-lib.pl";
use Win32::ODBC;
if (!$Db = new Win32::ODBC("dsn=PerlOracle;UID=scott;PWD=tiger")) {
    print "Content-type: text/html\n\n";
    print "<HTML><HEAD><TITLE>Database Test</TITLE></HEAD>";
    print "<BODY BGColor=\"#FFFFFF\" Text=\"#088000\"><H2>Error Connecting to Database</H2>";
    print "<P>Please Try Your Request at A Later Time</P></BODY></HTML>";
    exit;
}
else { print "Content-type: text/html\n\n Connection OK"; }
```

### The Connection Method

#### Syntax:

##### Connection

This method will return the object's ODBC connection number. The WIN32::ODBC extension for Perl, permits multiple ODBC connections to be open at an given point of time. Each of these connections is assigned a unique number for identification purposes. Specific active connections can be accessed in a Perl script by using the Connection method.

#### Example 2:

```
#!c:/perl/bin/perl.exe -w
require "cgi-lib.pl";
use Win32::ODBC;
if (($Db = new Win32::ODBC("dsn=PerlOracle;UID=scott;PWD=tiger"))) {
    $cnumber = $Db->Connection;
    print "ODBC Connection Number is: $cnumber";
}
else {
    print "Could not Create ODBC Object";
}
```

### The Sql Method

#### Syntax:

##### Sql (<SQL Statement>)

This method executes the SQL command. On successful execution, this method returns **undef**. If the execution fails, it returns an SQL error code.

#### Example:

```
$value = "SalesGroup";
if (!$Db->Sql("SELECT * FROM Passwd WHERE LOGIN = '$value'")) {
    print "Content-type: text/html\n\n Successfully Executed the SELECT statement \n";
}
else {
    print "Content-type: text/html\n\n Failed to execute the SELECT statement \n";
}
```

## The FetchRow Method

This method fetches the next row of data from the specified SQL sentence. This method returns TRUE if successfully executed and undef if there is an error. FetchRow should be used in conjunction with either Data or DataHash to actually retrieve the individual elements of data.

### Example:

```
if ($Db->FetchRow()) {
    undef %Data;
    %Data = $Db->DataHash();
    print "$Data{'LOGIN'} \n"; # All fieldnames are in Uppercase
}
else {
    print "Could not execute FetchRow. Please check to see if data exists in the table \n";
}
```

## The Data Method

### Syntax:

```
Data
Data <List>
```

This method retrieves data from a fetch (FetchRow method) for a list of fieldnames. If no field names are given, all fields are returned in an unspecified order. In a scalar context, the Data method returns all of the specified field values concatenated together. In an array context, it returns an array of the values, in the specified order.

### Example:

```
if ($Db->FetchRow()) {
    @data = $Db->Data();
    print "User Name is : $login \n";
    print "assword is : $passwd \n";
}
```

or

```
if ($Db->FetchRow()) {
    $login = $Db->Data('LOGIN_ID');
    print "@data \n";
}
```

## The DataHash Method

### Syntax:

```
DataHash
DataHash <List>
```

Similar to the Data method, this method retrieves data from a fetch (FetchRow method) for a list of fieldnames. If no field names are given, all fields are returned in an unspecified order. This method returns a hash or associative array, where the field name is the key.

**Example:**

```

if ($Db->FetchRow()) {
    undef %Data;
    %Data = $Db->DataHash();
    foreach $key (keys %Data) {
        print "$key : $Data{$key} \n";
    }
}

```

**The Close Method****Syntax:**

Close

This method closes the ODBC connection for the object. The return value of this method is always undef

**Example:**

```
$Db->Close();
```

*Tip*



In Perl, Cookies can be set using the following syntax:

```
Set-cookie: NAME=<Value>; EXPIRES=<Date>; PATH=<Path>;
          DOMAIN=<Domain>; SECURE
```

**Example:**

```
print "Set-cookie: Cname = $name";
```

Cookies can be accessed using an environment variable, \$ENV{'HTTP\_COOKIE'}

**Example:**

```
$var_cookie = $ENV{'HTTP_COOKIE'}
```

**CASE STUDY – TRAINING INFORMATION AT SCT****Requirements**

SCT is implementing an Intranet at its premises. A part of this Intranet will be required to maintain and display information about the courses conducted at SCT. The Intranet will have to maintain data about the following:

- Course Title
- Course Duration
- Course Content
- Course Fees

The Intranet will have to permit the user to:

- Create New Courses
- Edit existing Courses
- Delete existing Courses
- View existing Courses

For editing and deleting existing courses, the user requires an interface, which permits the selection of any one course, which is to be edited or deleted. The list of existing courses, which get displayed need to be generated at run time depending on the data available in the database.

The Intranet should also provide security by means of a login consisting of a Username and Password combination.

## Implementation

This Intranet will be implemented using Oracle as the data storage system. The interface between the browser and Oracle will be via the CGI specification. CGI scripting will be done using Perl 5.0 and the Perl WIN32::ODBC extension will be used for communicating with Oracle.

### Step I - Create The Table Structures For Oracle.

From the Requirements section of the case study it is clear that the system will need two database tables:

- Password file
- Course Details file'

Create The Password File:

```
CREATE TABLE passwd(Username VarChar2(25) NOT NULL, Login VarChar2(10) NOT NULL,
    Passwd VarChar2(10) Not Null);
```

Create the Course Details File:

```
CREATE TABLE course_details(Course_Title VarChar2(25) NOT NULL,
    Course_Subjects VarChar2(60) NOT NULL, Duration Number(4) NOT NULL,
    Cost Number(9,2) NOT NULL);
```

### Step II – Insert data into the PASSWD File and COURSE DETAILS File

Insert data into PASSWD File:

```
INSERT INTO passwd VALUES('John Ferguson', 'John', 'J020');
INSERT INTO passwd VALUES('Mary Poppins', 'Mary', 'M021');
```

Insert data into course\_details File:

```
INSERT INTO course_details
    VALUES('Oracle Programming', 'Oracle 8.0, D2k Rel 2.0, DBA', 120, 16000);
INSERT INTO COURSE_DETAILS
    VALUES('Internet Programming', 'Oracle 8.0, HTML, JavaScript, CGI-Perl, Java', 260, 25000);
```

### Step III – Create an HTML page for User Authentication

<HTML>

```
<HEAD><TITLE>SCT – User Authentication </TITLE></HEAD>
```

```
<Body Bgcolor="#FFFFFF" Text="#008000">
```

```
<TABLE Border=1 Width=100%><TR><TH BgColor="#000000">
```

```
<FONT Color="#FFFFFF">User Authentication</FONT>
```

```
</TH></TR></TABLE>
```

```
<DIV><Form Action="/cgi-bin/check.cgi" Method="POST">
```

```
<Table Align="Center" BgColor="#FFDAB9" Border=1 Width=50%><TR>
```

```
<TD>User Name</TD>
```

```
<TD><P><INPUT Name="Login" Size=10 Type="text" ><BR></TD>
```

```
</TR><TR>
```

```

        <TD>Password</TD>
        <TD><INPUT Name="Passwd" Size=10 Type="password"></TD>
    </TR></TABLE>
    <CENTER><INPUT value="Submit" name="B1" Type="submit"></CENTER>
</FORM></DIV>
</BODY>
</HTML>

```

As can be seen from the HTML code above, when the user submits this form, the Web Server will invoke the **check.cgi** CGI script. The purpose of the **check.cgi** script is to validate the login and password entered by the user and either allow or deny access to the Intranet.

#### Step IV – Create the check.cgi Script

```

#!c:/perl/bin/perl.exe -w
require "cgi-lib.pl";
# Set Up Database Connection
use Win32::ODBC;
if (!$Db = new Win32::ODBC("dsn=PerlOracle;UID=scott;PWD=tiger")) {
    print "Content-type: text/html\n\n";
    print "<HTML><HEAD><TITLE>Database Test</TITLE></HEAD>";
    print "<BODY BgColor=\"#FFFFFF\" Text=\"#088000\"><H2>Error Connecting to Database</H2>";
    print "<P>Please Try Your Request at A Later Time</P></BODY></HTML>";
    exit;
}
# Call ReadParse from cgi-lib.pl to decode form information
&ReadParse;
$value = $in{'Login'};
if (!$Db->Sql("SELECT * FROM Passwd where LOGIN = '$value'")) {
    if ($Db->FetchRow()) {
        undef %Data;
        %Data = $Db->DataHash();
    }
}
else {
    print "Content-type: text/html\n\n";
    print "<HTML><HEAD><TITLE>Database Test</TITLE></HEAD>";
    print "<BODY BgColor=\"#FFFFFF\" Text=\"#088000\">";
    print "<STRONG>There has been an Unknown Error!!</STRONG></BODY></HTML>";
    exit;
}
if (($Data{'LOGIN'} eq $in{'Login'}) && ($Data{'PASSWD'} eq $in{'Passwd'})) {
    print "Content-type: text/html\n\n";
    print "Set-cookie: Name=$Data{'LOGIN'}; expires=Wednesday, 01-Jan-99 12:00:00 GMT\n\n";
    &LoginPage;
}
else {
    print "Content-type: text/html\n\n";
    print "<HTML><HEAD><TITLE>Database Test</TITLE></HEAD>";
    print "<BODY BgColor=\"#FFFFFF\" Text=\"#088000\">";

```

```

    print "<P>Access Denied. Please Check your Login and Password!!</P></BODY></HTML>";
}
close($db);
exit;

sub LoginPage {
    print <<"RESPONSE";
<HTML>
    <HEAD><TITLE>Welcome to the SCT Intranet</TITLE><Base Target="rbottom"></HEAD>
    <BODY BgColor="#FFFFFF" Text="#088000">
        <TABLE BgColor="#000000" Border=1 Width=100%><TR>
            <TD BgColor="#000000" Colspan="3"><P Align="Center">
                <FONT Color="#FFFFFF">SCT Intranet</FONT></TD>
            </TR><TR>
                <TD BgColor="#FFDAB9"><FONT Color="#000000">
                    <A HRef="request.cgi">Leave Module</A></FONT></TD>
                <TD BgColor="#FFDAB9"><FONT Color="#000000">
                    <A HRef="user.cgi">Users Module</A></FONT></TD>
                <TD BgColor="#FFDAB9"><FONT Color="#000000">
                    <A HRef="enquiry.cgi">Enquiry Module</A></FONT></TD>
            </TR></TABLE><BR>
            <TABLE BgColor="#FFDAB9" Border=0 Width=100%><TR>
                <Td>The Admin Department at SCT Welcomes $Data{'USERNAME'}</TD>
            </TR></TABLE>
        </BODY>
</HTML>
    RESPONSE
}

```

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

- Using \_\_\_\_\_ PERL can communicate with any data storage system.
- The \_\_\_\_\_ module in PERL permits connectivity to a data storage system using ODBC drivers.
- A \_\_\_\_\_ must be created in order to use the Win32::ODBC module.
- The \_\_\_\_\_ method creates a new ODBC object for a given DSN.
- The \_\_\_\_\_ method fetches the next row of data specified by the SQL statement.
- The \_\_\_\_\_ method closes the ODBC connection for the object.
- The \_\_\_\_\_ method returns a hash array, where the field name is the key.

### TRUE OR FALSE

- If the "new" method is unsuccessfully executed it returns undef.
- The Win32::ODBC extension for PERL permits only one ODBC connection at a given point of time.
- If the "Sql" method is successfully executed it returns undef.
- The Close method always returns undef.
- The Data method returns a hash or associative array.



## 22. DEBUGGING IN PERL

It is natural that PERL programs may not run perfectly the first time they run. PERL being an interpreted language unless an attempt is made to run the PERL script **errors will never surface**.

Error handling code needs to be embedded in the PERL program using techniques such as the `||` operator **and** the `die` keyword. Such error handling code spec attempts to let the user know what went wrong when the PERL script executes within the PERL interpreter.

Errors could occur because of syntax mistakes in the code that prevents it from running. Tracking down and fixing bugs is a process known as **debugging**.

PERL contains a multifunction debugger that allows the use of several techniques to find and fix syntax or semantic errors in the PERL code. Quite apart from syntax or semantic errors, errors in program logic also ensure that the throughput of a PERL program could be erroneous.

The following material shows how to use the PERL debugger to do things such as:

- List parts of a program
- Set breakpoints
- Trace the program's execution
- Execute the program one statement at a time

And so on thus allowing a PERL programmer to pinpoint exactly what is going wrong in the PERL program and hopefully set it right.

### LOADING AND LEAVING THE PERL DEBUGGER

Load the PERL debugger by specifying the `-d` switch when running a PERL script. For example the program named **marker.pl** can be run through the debugger by typing the following at the command prompt.

```
<Command Prompt> perl -d marker.pl ↵
```

The following is the code for **marker.pl**, which is used as reference through this material.

```
#!c:/perl/bin/perl.exe -w
%costs = ("Mercedes" => 90000, "BMW" => 82000,
        "Toyota" => 37500);
foreach $car (keys (%costs)) {
    $marker = " * ";
    $amount = $costs{$car};
    $amount /= 10000;
    $marker x= $amount;
    print "$car: $marker\n";
}
$marker = "-" x 40;
print "$marker\nNote: * = \$10,000\n";
```

When the `-d` command is issued, the PERL debugger loads immediately and displays a multi line message. The message looks similar to:

```
Stack dump during die enabled outside of evals.
Loading DB routines from PERL5DB.pl patch level 0.95
Emacs support available
Enter h or 'h h' for help.
```

```
main::(marker.pl:2): %costs = ("Mercedes" => 90000, "BMW" => 82000,
main::(marker.pl:3):      "Toyota" => 37500);
```

```
DB<1>_
```

The first three lines displayed by the debugger are system messages. The fourth line says that typing the letter `h` provides help on using the debugger.

The next line displays the first executable line of the program loaded with the debugger. When the debugger finds an executable line, it displays the following information about the line:

- The Package the line is part of (in this case package `main`)
- The name of the program currently executing (in this case `marker.pl`)
- The line number of the program statement

After these lines are displayed, the debugger's command prompt is displayed indicating that the debugger is waiting to accept a command.

The number in the angular brackets is the command number of the debugging session. Each command is numbered so that if the command entered needs to be referenced later in the debugging session it can be done by using its command number.

Once the debugging session is over, the debugger can be closed using the command `q` at the debugger command prompt.

## LISTING THE PROGRAM CODE

A basic debugging technique is listing program code. The PERL debugger has several ways to list program code. The following sections cover the listing commands provided by the debugger.

### Using the `l` Command

The `l` command lists the next 10 lines of program code. When the `l` command is typed at the prompt, the next ten program statements from the current listing point will be printed on the screen.

The following is the example of issuing the `l` command on the `marker.pl` program:

```
DB<1> l
2:==> %costs = ("Mercedes" => 90000, "BMW" => 82000,
3:      "Toyota" => 37500);
4:      foreach $car (keys %costs) {
5:          $marker = "-";
6:          $amount = $costs{$car};
7:          $amount /= 10000;
8:          $marker x= $amount;
9:          print "$car: $marker\n";
10:     }
11:     $marker = "-" x 40;
```

Entering the `l` command again will result in the next 10 statements being listed and so on till the last line of the program code is listed. In the case of `marker.pl` giving the `l` command will list the remaining line of code as shown below:

```
12:          print "$marker\nNote: * = \$10,000\n";
```

Specifying the line number after the `l` command lists the specified line. The following example lists just line 11 of the `marker.pl`:

```
DB<1> l 11 ↵
```

```
10:          $marker = "-" x 40;
```

The prompt now is:

```
DB<2>
```

One can also specify the range of line numbers to list by including the range after the `l` command. The following example lists the lines 5-10 of `marker.pl`.

```
DB<2> l 5-10 ↵
```

```
5:          $marker = "-";
6:          $amount = $costs{$car};
7:          $amount /= 10000;
8:          $marker x= $amount;
9:          print "$car: $marker\n";
10:         }
```

To list a certain number of lines from a specified number of lines, specify the line number to be listed, a plus sign (+), and the number of lines to be listed following the specified line number. For example:

```
DB<3> l 5+5 ↵
```

```
5:          $marker = "-";
6:          $amount = $costs{$car};
7:          $amount /= 10000;
8:          $marker x= $amount;
9:          print "$car: $marker\n";
10:         }
```

Finally if the PERL script contains a subroutine, list that subroutine by specifying the subroutine name after the `l` command. For example if `marker.pl` had a subroutine called `starmark` then, the subroutine can be listed as follows:

```
DB<4> l starmark ↵
```

## Using The `w` Command

The `w` command displays the line specified, **three** lines preceding it and **several** lines after it, filling the rest of the debugger window.

```
DB<5> w 5 ↵
```

```
2:==> %costs = ("Mercedes" => 90000, "BMW" => 82000,
3:          "Toyota" => 37500);
4:          foreach $car (keys (%costs)) {
5:              $marker = "-";
6:              $amount = $costs{$car};
7:              $amount /= 10000;
8:              $marker x= $amount;
9:              print "$car: $marker\n";
10:         }
```

```

11:          $marker = "-" x 40;
12:          print "$marker\nNote: * = \$10,000\n";

```

## Using the // Command

The debugger can be used to search for a pattern in program code by surrounding the pattern between slashes (/). To search for the keyword **print** in **marker.pl** use the following command:

```

DB<6> /print/ ↵
9:          print "$scar: $marker\n";

```

Issuing this command will display the first instance of **print**, if the pattern is found. If the pattern is not found then the debugger will display the following:

```

DB<7> /max/ ↵

```

To search for the same pattern a multiple times, leave out the final slash. The command looks as follows:

```

DB<8> /print ↵

```

In this case, the debugger will look for the next instance of the pattern **print**. When the pattern is found, subsequent searches will be made from the line number of where the pattern was found right to the end of the file.

## Using The ?? Command

To search a program from the bottom upwards to find a pattern, surround the pattern with question marks. If, for example, to search for **print** through the **marker.pl** from the bottom up use the following:

```

DB<9> ?print? ↵
12:          print "$marker\nNote: * = \$10,000\n";

```

Like the // command, if searches for a pattern with ?? cannot be found by the debugger, the debugger will display a **not found** message. Leave out the trailing question mark when wanting to search for a pattern from the bottom of the file to the top.

## Using the S Command

The **S** command lists all the subroutine names found in the current file. This command displays not only the subroutines defined in a program but also the subroutines used by PERL. For example:

```

DB<9> S ↵
DB :: fake :: at_exit
Term :: ReadLine::BEGIN
Term :: ReadLine::Stub::Attribs
Term :: ReadLine::Stub::Features
Term :: ReadLine::Stub::IN
Term :: ReadLine::Stub::MinLine
Term :: ReadLine::Stub::OUT
Term :: ReadLine::Stub::ReadLine
Term :: ReadLine::Stub::addhistory
Term :: ReadLine::Stub::findConsole
Term :: ReadLine::Stub::new
Term :: ReadLine::Stub::newTTY
Term :: ReadLine::Stub::readline
Term :: ReadLine::TermCap::LoadTermCap
Term :: ReadLine::TermCap::ornaments
Term :: ReadLine::Tk::Tk_loop

```

```

Term :: ReadLine::Tk::get_c
Term :: ReadLine::Tk::get_line
Term :: ReadLine::Tk::handle
Term :: ReadLine::Tk::register_Tk
Term :: ReadLine::Tk::tkRunning
main :: BEGIN

```

All the subroutines listed are used by PERL internally. If for example `starmark` was a subroutine defined in the `marker.pl` file then that subroutine would also be listed after `main::BEGIN` as

```

main::BEGIN
main::starmark

```

## USING THE DEBUGGER TO STEP THROUGH A PROGRAM

A commonly used debugging technique is to step through a program line by line in order to understand what is happening in the program as each line is executed. The PERL debugger provides several commands that do exactly this.

### Using the `s` Command

The `s` command is used to execute the current statement of the program. When the command is issued the current line is executed and the next line to be executed is displayed. Any input or output operations are performed before the next line is displayed, so if the program is looking for an input from the user, the debugger will pause until the input is entered before displaying the next line. If the execution of the statement causes the output, the debugger will display the output before displaying the next line of code. The following example executes the first several lines of `marker.pl`:

```

DB<1> s ↵
main::(marker.pl:4):      foreach $car (keys (%costs)) {
DB<1> s ↵
main::(marker.pl:5):      $marker = "";
DB<1> s ↵
main::(marker.pl:6):      $amount = $costs{$car};
DB<1> s ↵
main::(marker.pl:7):      $amount /= 10000;
DB<1> s ↵
main::(marker.pl:8):      $marker x= $amount;
DB<1> s ↵
main::(marker.pl:9):      print "$car: $marker\n";
DB<1> s ↵
Toyota: * * * * *
main::(marker.pl:5):      $marker = "";
DB<1> s ↵
main::(marker.pl:6):      $amount = $costs{$car};
DB<1> s ↵
main::(marker.pl:7):      $amount /= 10000;
DB<1> s ↵
main::(marker.pl:8):      $marker x= $amount;
DB<1> s ↵
main::(marker.pl:9):      print "$car: $marker\n";

```

```

DB<1> s ↵
Mercedes: * * * * * * * * * *
main::(marker.pl:5):          $marker = "*";
DB<1> s ↵
main::(marker.pl:6):          $amount = $costs{$car};
DB<1> s ↵
main::(marker.pl:7):          $amount /= 10000;
DB<1> s ↵
main::(marker.pl:8):          $marker x= $amount;
DB<1> s ↵
main::(marker.pl:9):          print "$car:  $marker\n";
DB<1> s ↵
BMW: * * * * * * * * * *
main::(marker.pl:11):         $marker = "-" x 40;
DB<1> s ↵
main::(marker.pl:12):         print "$marker\nNote: * = \$10,000\n";
DB<1> s ↵
-----
Note: * = $10,000

```

When a subroutine is encountered when using the **s** command, the debugger treats it just like the other parts of the program and assumes that it also needs to be debugged. However the debugger **will not** execute the first statement in the subroutine, instead it waits for a **s** command and then proceeds to execute the statement.

### Using the **n** Command

The **n** command works similarly to the **s** command, which executes a statement and displays the next statement to be executed. It differs from the **s** command, as it **will not** step through a subroutine. Instead it will execute the whole subroutine and then display the first statement **after** the call to the subroutine.

For example, if the current line as displayed by the debugger is:

```
main::(marker.pl:18):         &starmark(\%star_ref \ %star_fref1);
```

When the **n** command is executed, the subroutine will be executed and the next line of the program is displayed. None of the subroutines lines are displayed, and the subroutine cannot be debugged using the **n** command.

### Using the **r** Command

The **r** command can be used when stepping through a subroutine and not executing the subroutine line by line anymore. Issuing the **r** command executes the remainder of the subroutine and displays the first line of code following the subroutine.

### Pressing ENTER with the **s** and **n** Commands

When using the debugger with the **s** or **n** commands, use the **ENTER** key to execute either the **s** or **n** command, entered last. The above works just like typing either an **s** or an **n** again. using the debugger to view variable values

When debugging a program, often values of different variables must be checked while the program is being executed. The PERL debugger offers two commands to use to view variable values. These commands are discussed in the following section.

## Using The X Command

The **X** command displays the value of any variable that is part of the current package. If no other package is specified, the **X** command will grab the values of variables from the package **main**.

The **X** command can either be issued by itself, in which case every variable of the current package and its value will be displayed, or can be issued with a variable name, which will display the value of the variable specified. This is shown in the following example:

```
DB<10> s ↵
main::(marker.pl:4):      foreach $car (keys (%costs)) {
DB<10> s ↵
main::(marker.pl:5):      $marker = "*";
DB<10> s ↵
main::(marker.pl:6):      $amount = $costs{$car};
DB<11> X marker ↵
$marker = "*"
DB<11> s ↵
main::(marker.pl:7):      $amount /= 10000;
DB<11> X amount ↵
$amount = 37500
DB<11> s ↵
main::(marker.pl:8):      $marker x= $amount;
DB<11> s ↵
main::(marker.pl:9):      print "$car: $marker\n";
DB<11> X car ↵
$car = 'Toyota'
```

In the example, the first few statements in the **marker.pl** are executed so that the variables of the program can obtain values. Once these variables have obtained values, the **X** command was issued with the variable name. Notice that the variables were entered **without** the **\$** prefix. Had the variables been entered as **\$marker**, **\$amount** no values would have been displayed because the debugger **assumes the \$ prefix**.

Issuing the **X** command without an argument displays every variable value pertinent to the current program environment, which includes values for all the environment variables, special variables if any and PERL's internal variables.

## Using The v Command

To display the value of a variable in a package other than **Main**, use the **v** command. For example if there is a package titled **Domarks** in the program **marker.pl**, display the values of the variables in the package as follows:

```
DB<12> v Domark var1 var2 ↵
$var1 = "<Some Value>"
$var2 = "<Some Value>"
```

## SETTING AND WORKING WITH BREAKPOINTS

Another common and powerful debugging technique is the setting of **breakpoints**. A breakpoint is a place in a program where its execution stops. Normally, breakpoints are set at certain key spots in a program, to examine the value of variables or check to see if the logic of the program is working as it was supposed to work. The following section will deal with setting breakpoints.

## Using The b Command

The **b** command is used to set a breakpoint in a PERL program. To execute a program only till line 10 use the following command:

```
DB<12> b 10 ↵
```

By issuing the above command the debugger will execute all the statements in the program **up to but not including** line 10.

Use the **b** command with a conditional expression, so that its program execution will halt only if the expression evaluates to true. Set breakpoints at a subroutine. When this is done the debugger will break the program **before** the first statement of the subroutine is executed. Once a breakpoint is set, it remains until removed as described below.

## Using The c Command

Once breakpoints have been set in a program, use the **c** command to execute the program until it reaches a breakpoint or the end of the program. The following example shows how this works:

```
DB<13> b 5 ↵
```

```
DB<13> c ↵
```

```
main::(marker.pl:5):                               $marker = "*";
```

In this example a breakpoint was set at line 5. Then the program was executed using the **c** command. Execution of the program stops at line 5, the breakpoint set.

Use the **c** command to set temporary breakpoints. Temporary breakpoints are convenient because once the program has executed and the breakpoint is reached, the debugger **deletes it automatically**.

```
DB<14> c 13 ↵
```

```
Mercedes: * * * * * * * * *
```

```
main::(marker.pl:5):                               $marker = "*";
```

If **c** command is issued next, the program **will not** break at the line 15 because the temporary breakpoint is deleted as soon as it is reached.

## Using the L Command

Occasionally, in the course of a debugging session, breakpoints may be set and their position in the program forgotten. The **L** command lists the line number and the statements where the breakpoints are applied along with the message that the program will break at this line. This is shown in the following example:

```
DB<15> L ↵
```

```
marker.pl
```

```
9:          $marker = "*";
```

```
break if(1)
```

```
10:         $ amount = $costs{$car};
```

```
15:         $ marker = "-" x 40;
```

```
break if(1)
```

The statement **break if(1)** simply means that a break will always occur at this line since (1) always evaluates to true.



## Using the d and D Commands

The **d** command passed along with a line number having a breakpoint deletes single breakpoints. In the next example, the breakpoint at line 5 of **marker.pl** is deleted.

```
DB<16> d 5 ↵
DB<17>
```

There is no **breakpoint delete confirmation** demanded, the breakpoint is simply deleted. If all the breakpoints previously set must be deleted, use the **D** command. Again no breakpoint delete confirmation is demanded, but all previously set breakpoints are deleted.

## DEBUGGING BY PROGRAM TRACING

Program tracing is a debugging technique in which each line of a program is displayed as it is executed. This technique is particularly useful when trying to debug loops and conditional expressions. This section discusses how to use program tracing to help debug PERL programs.

Issuing the **t** command turns on program tracing. When this command is issued the debugger is said to be in the **trace mode**.

```
DB<18> t ↵
Trace = on
```

Use **trace** with other debugging commands to help debug PERL programs effectively. For example the following command uses trace mode with a breakpoint to display the execution of a program up to a temporary breakpoint:

```
DB<18> t ↵
Trace = on
DB<18> c 8 ↵
main::(marker.pl:4):      foreach $car (keys (%costs)) {
main::(marker.pl:5):      $marker = "*";
main::(marker.pl:6):      $amount = $costs{$car};
main::(marker.pl:7):      $amount /= 10000;
main::(marker.pl:8):      $marker x= $amount;
DB<19>
```

Each statement in the program is displayed and executed, up to line 10, which is displayed but not executed. To turn off trace mode, simply reissue the trace (**t**) command, which turns off program tracing.

## DEBUGGING WITH LINE ACTIONS

During a debugging session, as the lines of the program are displayed maybe the value of a variable has to be displayed or changed. The PERL debugger provides several commands for performing such **line actions**. The following section covers these commands.

### Using The a And A Commands

If a particular action must be performed **before** a line of code executes, use the **a** command. The **a** command takes a **line number** and an **action** as its arguments. This is shown in the following example:

```
DB<20> a 5 print "this is where marker is made *" ↵
DB<21>
```

In this example, the **a** command is used to tell the debugger to print the statement in quotes before line 5 is executed. When the program is executed the following statement appears after the current line is displayed, but before the next debugger command.

```
this is where marker is made *
```

If more than one action needs to be performed, simply separate the statements as done with regular PERL statements. This is as follows:

```
DB<1> a 5 print "this is where marker is made *"; \ ↵  
cont: $marker = "*"; ↵  
DB<2>
```

The example shows how to have more than one action per command and how to continue a line when the first line is too long. When a line needs to be broken, enter a backslash (\) and press ENTER. The debugger will then display **cont:** continue typing on this line. Notice that the first statement **ends** with a **semicolon**.

When finished with line actions for a debugging session, issue the **A** command at the debugger prompt which deletes all the line actions currently defined.

### Using The < And > Commands

These commands are used to perform line actions at certain times in the debugging session. The **>** command is used to perform an action **before** any other statements are executed, and the **<** command is used to perform actions **after** all other statements are executed.

These commands are useful when there is a variable, which has a bad value but it cannot be determined which statement in the code is assigning the bad value to the variable. For example, to print the value of a variable before a line is executed, issue the **>** command as follows:

```
DB<24> > print "the value of marker is $marker" ↵
```

To display the value after a line of code executes issue the **<** command as given below:

```
DB<24> < print "the value of marker is $marker" ↵
```

If during the debugging session the lines have, **<** or **>** commands associated with them need to be seen, use the **L** command to display this information

## MISCELLANEOUS DEBUGGING COMMANDS

The following are some of the other debugging commands, which can be used through a debugging session.

### Using the R Command

The **R** command is used to restart the debugger without actually quitting debugger mode. If the **R** command is issued new breakpoints, line actions, and so on may have to be set. This is because the old ones are **often lost** during debugger restart.

## Using The H Command

The H command is used to display the **previous** debugging command entered during the current debugging session, as shown in the following example:

```
DB<25> H ↵
3: b 11
2: b 10
1: b 9
DB<25>
```

Previous debugger commands are listed from the **most** recent to the **least** recent (i.e. in reverse order in which they were set).

## Using The ! Command

The ! command is used to repeat a previously issued debugger command. The command's only argument is the line number of the debugger command, which can be found by issuing the H command.

This command is used if all breakpoints have been accidentally deleted and need to be reset again. Use the H command together with the ! command to reset them again. As shown in the following example:

```
DB<25> H ↵
3: b 11
2: b 10
1: b 9
DB<25> D ↵
Deleting all breakpoints...
DB<25> !1 ↵
b 9
DB<25> !2 ↵
b 10
```

As seen in the example the ! command simply reissues the command. This command is very handy if there are long debugging commands that are inconvenient to retype.

## Using The p Command

Many a times in a debugging session a quick calculation may need to be done, perhaps to determine what the value of a variable should be. PERL's debugger provides the p command for evaluating the value of an expression while in the debugger. For example:

```
DB<25> p (52+56)/12 ↵
```

## Using The T Command

The T command is used to do a stack trace, which is useful for determining which subroutines are in progress and where they were called. The following is a sample stack trace that shows how several subroutines are called in a program:

```
DB<26> T ↵
$. = &main :: double($var1) from file DoubleNum line 3
$. = &main :: sqrt($var1) from file DoubleNum line 6
```

This stack trace is working with a program called **DoubleNum.pl**. The first line that is displayed after the T command is issued shows that the subroutine **double**, which is within the package **Main**, has been called with a scalar argument (**\$var1**) and is supposed to return a scalar variable.

The second line of the stack trace shows that the subroutine **double** was called from another subroutine, **sqrt**. This subroutine was also called with a scalar argument and is supposed to return a scalar variable.

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. The process of tracing and fixing bugs in an program is known as \_\_\_\_\_.
2. The PERL debugger is loaded by specifying the \_\_\_\_\_ switch.
3. One can exit the debugger by simply entering the \_\_\_\_\_ command.
4. The \_\_\_\_\_ command is used to display a window of lines that surround a specific line in the program.
5. The \_\_\_\_\_ command is used to search for a pattern in the program.
6. The S command lists all the \_\_\_\_\_ names found in the current program file.
7. The \_\_\_\_\_ command displays the value of any variable that is part of the current package.
8. Breakpoints in a Program can be set using the \_\_\_\_\_ command.
9. When the t command is issued the debugger is said to be in the \_\_\_\_\_ mode

### TRUE OR FALSE

10. When the PERL debugger is loaded using the -d switch the first three lines display the first three lines of the program code that is to be debugged.
11. The l command lists all the lines of the program code.
12. The ?? command is used to do a backward pattern search in the program code.
13. The S command is used to execute the current statement of the program.
14. The R command is used to restart the debugger without actually quitting it.

## 23. INSTALLING AND SETTING UP APACHE WEB SERVER

### THE BIRTH OF APACHE

When Rob McCool, who had developed NCSA HTTPd, left NCSA in 1994, the project fizzled out. Since the source code was publicly available, many people using it had developed their own bug fixes and additional features that they needed for their own sites. The patches were shared via Usenet, without any centralized place for collecting and distributing these patches.

Brian Behlendorf and Cliff Skolnick had put up a mailing list and Brian set up a CVS (Concurrent Versioning System) tree, now anyone who wanted to could contribute new features and bug fixes. This led to place where a group of developers could collect and distribute the patches and bug fixes. Thus came into existence - **Apache**.

### AN INTRODUCTION TO APACHE

Since it was a patchy Web server, the name APACHE got from A PAtCHy sErver. The Apache version 0.6.2 was released in April 1995. Currently there are two versions of Apache available the first is 1.3.X.XX-X the most popular and tried and test and completely stable. Apache.org, the Apache website indicates that there will be no enhancements made to 1.3.X.XX-X on bug fixes done. The second is 2.0.X.XX, which is the latest version of Apache, which is being constantly being **bug fixed** and **enhanced** (when creating this material). Either version can be use as a full-fledged production Web Server when required.

Apache is the hottest Web server for the Internet. Apache is the world's largest used Web server. Apache is a freely available Web server. It can be downloaded from <http://www.apache.org>.

### GETTING STARTED

#### Download Apache2

Download Apache2 from <http://httpd.apache.org/download.cgi>. In the **Downloads** section choose the **latest stable version** of Apache available for download. At the time of creating this material the latest stable version of Apache was **apache\_2.0.50-win32-x86-no\_ssl.msi**.

#### Apache 2.0.50 Is The Best Available Version

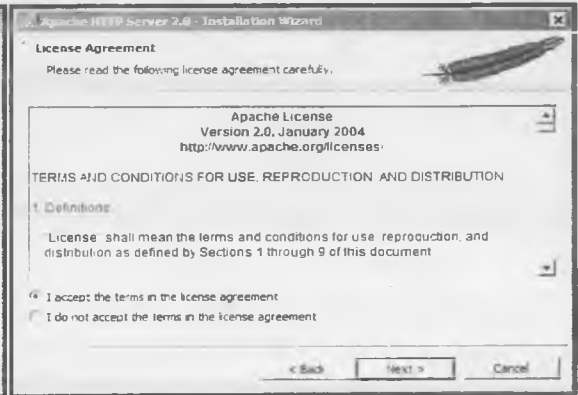
This release fixes security problems described in [CAN-2004-0493](#) and [CAN-2004-0488](#). It also contains bug fixes and some new features. For details see the [Official Announcement](#) and the [CHANGES 2.0](#) list

Apache 2.0 add-in modules are **not compatible** with Apache 1.3 modules. If third party add-in modules are currently being run using Apache 1.3, obtain new modules written for Apache 2.0 from that third party before attempting to upgrade from Apache 1.3 to Apache 2.0.

- Unix Source: [httpd-2.0.50.tar.gz](#) [PGP] [MD5]
- Unix Source: [httpd-2.0.50.tar.Z](#) [PGP] [MD5]
- Win32 Source: [httpd-2.0.50-win32-src.zip](#) [PGP] [MD5]
- Win32 Binary (MSI Installer): [apache\\_2.0.50-win32-x86-no\\_ssl.msi](#) [PGP] [MD5]
- Other files

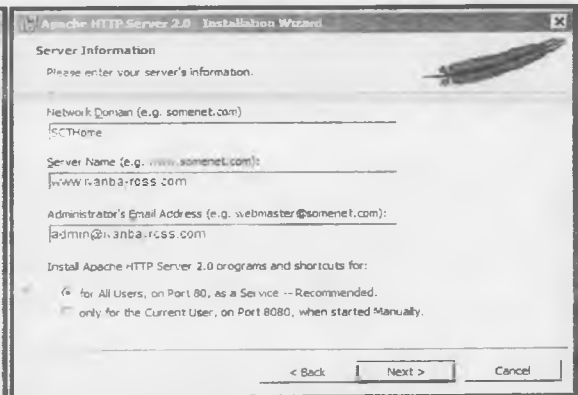
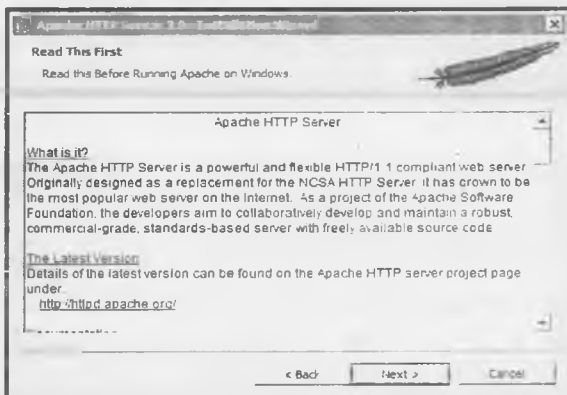
## The Apache 2 Installation Process

1. Double click the apache\_2.0.50-win32-x86-no\_ssl.msi file
2. The Dialog box as seen in diagram 23.1 appears. Click **Next >** to proceed
3. A License Agreement of Apache appears as seen in diagram 23.2. Select the option *I accept the terms in the license agreement* and the click **Next >** to proceed further



**Diagram 23.1:** Welcome Screen of Installing Apache      **Diagram 23.2:** License Agreement Screen of Apache

4. A brief information about Apache HTTP Server is displayed as shown in the diagram 23.3. Click **Next >**
5. The installation wizard prompts for server details such as the domain name, server name and the email address of administrator as seen in diagram 23.4. Fill in the appropriate details and click **Next >**



**Diagram 23.3:** The Screen showing the details of Apache HTTP Server

**Diagram 23.4:** Server Information

If either of this information is left blank the wizard will popup errors as seen in diagram 23.5.1 – diagram 23.5.3.

6. If no errors are encountered the wizard proceeds by prompting the setup type as seen in diagram 23.6. Select the setup type as Typical and click **Next >** to proceed

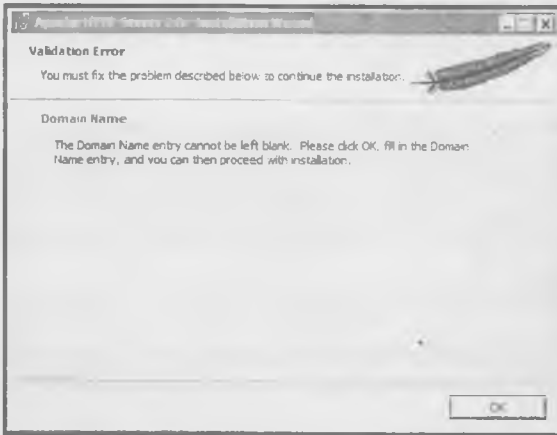


Diagram 23.5.1: Validation error for Domain name

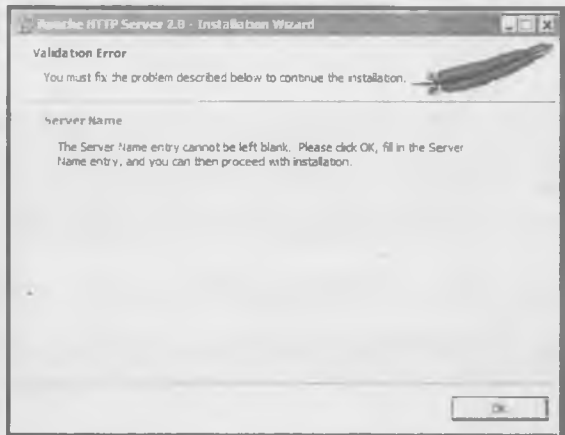


Diagram 23.5.2: Validation error for Server name

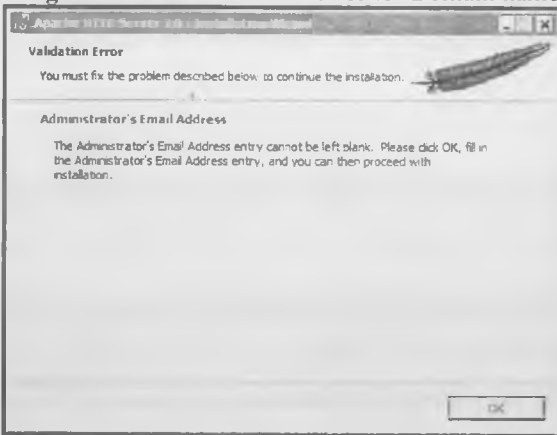


Diagram 23.5.3: Validation error for Email Address



Diagram 23.6: Setup Type

7. The wizard will now prompt for the destination folder as seen in the diagram 23.7. Change the path if required and click  to proceed
8. Finally the wizard displays the screen as shown in the diagram 23.8. Click  to start the actual installation process

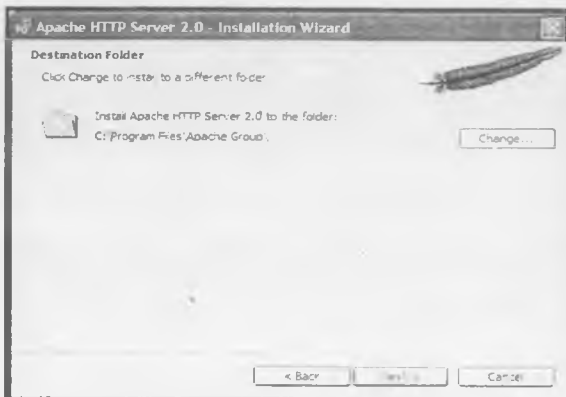


Diagram 23.7: Destination folder

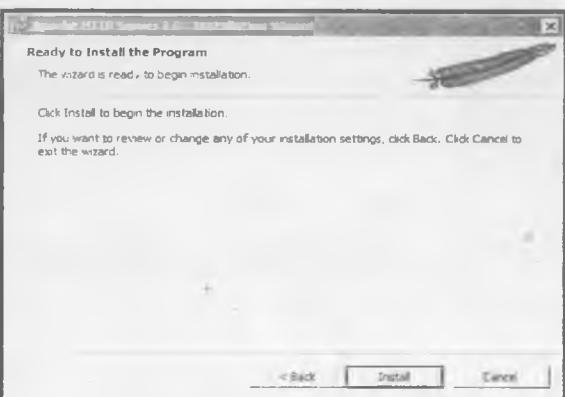


Diagram 23.8: Ready to Install

9. The installation progresses as seen in diagram 23.9

10. Click  to complete the installation as seen in diagram 23.10



Diagram 23.9: Installation in progress



Diagram 23.10: Installation complete

This concludes the actual installation process on the computer's HDD. Apache now needs to be configured so that it knows each and very virtual domain created under it.



Diagram 23.11.1: Apache2 Icon

### Testing Apache2

Once the installation is complete an icon as seen in diagram 23.11.1 comes up showing the apache2 status. This icon allows starting, stopping and restarting of apache server as and when required. The web server can be tested by pointing the web browser to the url <http://127.0.0.1> If a screen as seen in diagram 23.11.2 is displayed then the installation was successful.

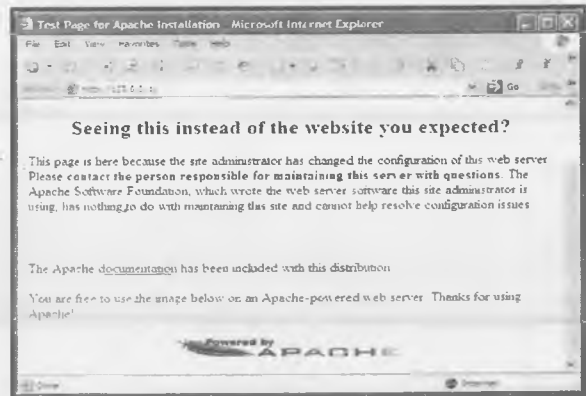


Diagram 23.11.2: Apache Testing Via I.E. 6

### DIRECTORY TREE STRUCTURE OF APACHE.

Once Apache is installed, its installation process creates several sub-directories under its root directory.

If Apache is installed in C:\, the installation process would by default place the Apache executables in C:\Program Files\Apache Group\Apache2.

The sub-directories are as seen in diagram 23.12:

Once the Apache executables are installed and its directory tree structure has been created and populated through its install process, it is necessary to configure Apache to work on the computer on which it is installed. There has to be specific entries made into several Apache files to make Apache run successfully. Appropriate entries should be made in its `httpd.conf`, `access.conf`, `srml.conf` (`conf-iguration`) files.

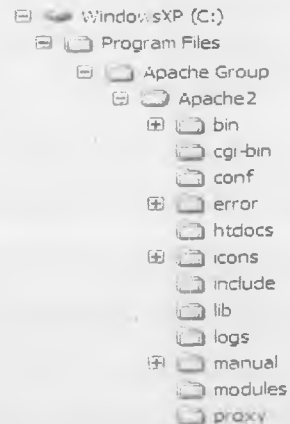


Diagram 23.12: Directory Structure



## Note



In previous versions of Apache for example Apache 1.3.3 entries have to be made in all the three configuration files i.e. `httpd.conf`, `access.conf` and `srm.conf`. But since the later versions for example Apache 1.3.6 onwards, entries have to be made in only one configuration file, i.e. `http.conf`. This file includes the data of all the other files i.e. `access.conf` and `srm.conf`.

All these files will be available in the `conf` sub-directory located under:

`C:\Program Files\Apache Group\Apache2`

## CONFIGURATION OF APACHE SERVER

When learning web page creation very often Apache server is loaded on the very same Windows XP computer on which the web pages are designed, for testing them. This enables the web page creators to physically check to see what the web pages would look like and behave before being mounted on an Internet based server.

With this in mind, the following must be done on the computer.

Locate the `Hosts` file available under `C:\WINDOWS\system32\drivers\etc`. This file is a pure ASCII file. This file will be used to hold the **IP address** and **name** of the local machine.

Open the `Hosts` file in NotePad and make the following entries: (Refer diagram 23.13)

ip. Address	Hostname
127.0.0.1	www.ivanbayross.com

(This can be any name of choice)

```

hosts - Notepad
File Edit Format View Help
Copyright (c) 1995-1999 Microsoft Corp.
This is a sample HOSTS file used by Microsoft TCP/IP for Windows.
This file contains the mappings of IP addresses to host names. Each
entry should be kept on an individual line. The IP address should
be placed in the first column followed by the corresponding host
name.
The IP address and the host name should be separated by at least
one
space.
Additionally, comments (such as these) may be inserted on
individual
lines or following the machine name denoted by a # symbol.
For example:
102.51.94.37 rhino.acre.com # source server
38.25.63.10 x.acre.com # x client host
127.0.0.1 localhost
127.0.0.1 www.ivanbayross.com
  
```

Diagram 23.13: hosts file

From now on, if Apache is passed the ip address 127.0.0.1, (the name **localhost**) or **www.ivanbayross.com** Apache will understand that a reference is being made to the same computer on which it is installed, which has a virtual domain named **ivanbayross.com** hosted on it.

## SETTINGS TO BE MADE IN THE HTTPD.CONF FILE

The main configuration file of Apache is `httpd.conf`, which contains directives written in plain text. The location of this file is set at compile-time.

**Any** directive may be placed in **any** of these configuration files. Apache is pretty flexible like that. Changes made to the main configuration files are only recognized by Apache when it is **started** or **restarted**. If any configuration file is actually a directory, Apache will enter that directory and parse any files (and sub-directories) found there as configuration files.

Apache **works best** if there is **only one** configuration file used and all its directives are placed in that file (i.e. `httpd.conf`). Apache also reads a file containing document mime types. This filename is set by the `TypesConfig` directive, and is `mime.types` by default.

Apache configuration files contain **one directive** per line. If a directive must continue onto the next line use back-slash `\` as the **last character** on the previous line.

Directives in configuration files are case-insensitive, but **arguments** to directives are often **case-sensitive**. Any line beginning with a hash (#) character is ignored. Blank lines and white spaces before a directive are ignored. Configuration files can be checked for syntax errors **without** starting the server by using **apachectl configtest** or the **-t** command line option.

## Understanding Some Important Entries In HTTPD.CONF File

### Global Settings

**ServerRoot** C:/Program Files/Apache Group/Apache2

This sets the absolute path to the server directory. Generally, the argument of ServerRoot should be the path to where Apache is installed.

**ErrorLog** logs/error.log

**CustomLog** logs/access.log common

The various directives that end in Log control indicate whether log files exist at all. These directives also indicate exactly where the log files exist on the file system.

**PidFile** logs/httpd.pid

It is a file in which the server should record its process identification number when it starts.

**ScoreBoardFile** logs/apache runtime status

It is a file used to store internal server process information. Not all architectures require this. But if local architecture does (This will be known because this file will be created when Apache is run) then **ensure** that no two invocations of Apache share the same Scoreboard file.

**Timeout** 300

Indicates the number of seconds before Apache receives and sends a time out.

**KeepAlive** On

Indicates whether or not to allow persistent connections (more than one request per connection). Set it to **Off** to deactivate.

**MaxKeepAliveRequests** 100

Indicates the maximum number of requests to allow during a persistent connection. Set it to **0** to allow an unlimited amount. It is recommended that this number is set high, for maximum performance.

**KeepAliveTimeout** 15

Indicates the number of seconds to wait for the next request from the same client on the same connection.

**ServerAdmin** admin@ivanbayross.com (As specified during the install process)

Accepts the Email address, where problems with the server should be e-mailed. This address appears on some server-generated pages, such as error documents.

**ServerName** www.ivanbayross.com:80 (As specified during the install process)

This sets the hostname the server will return. Set the name of the server using the ServerName directive. This is especially useful when the computer has multiple names or IP addresses.

**DocumentRoot** C:/Program Files/Apache Group/Apache2/htdocs

This indicates the absolute path of the document tree, which is the top directory from which Apache will serve files. The **DocumentRoot** is the root of the Web tree and it defaults to **C:/Program Files/Apache Group/Apache2/htdocs**. Assuming that Apache is installed in **C:/Program Files/Apache Group/Apache2/**, this can be changed if required.

**DirectoryIndex** index.html index.htm index.shtml index.php index.php4 index.php3  
index.phtml index.cgi

Specifies the name of the file or files to use as a pre-written HTML. Separate multiple entries with **spaces**.

**Alias** /icons/ "C:/Program Files/Apache Group/Apache2/icons/"

**ScriptAlias** /cgi-bin/ "C:/Program Files/Apache Group/Apache2/cgi-bin/"

These both map a URL path to a directory on the server.

Additional options can be added to directories using **<Directory>** and **Options** directives. **<Directory>** is a **container directive**. Container directives enclose multiple lines in the **config** file. They require a closing directive of the form **</Directive>**.

Aspects of Apache's behavior can be controlled on a per-directory or per-filename basis using **<Directory>** and related **<Files>** and **<FilesMatch>** directives. Directives placed between **<Directory></Directory>** apply to **sub-directories** as well. What options are allowed in a directory can be controlled using the **Options** directive. Possible values to an **Options directive** are given below with brief descriptions:

Option	Brief Explanation
ExecCGI	CGI scripts can be run from this directory tree
FollowSymLinks	The server will follow symbolic links in this directory
Includes	Server-side includes are permitted
IncludesNoEXEC	Server-side includes are permitted, but the #exec command and #include of CGI scripts are disabled
Indexes	If a URL, which maps to a directory is requested, and there is no DirectoryIndex (e.g. index.html) in that directory, then the server will return a formatted listing of the directory
MultiViews	Content negotiated MultiViews are allowed
SymLinksIfOwnerMatch	The server will only follow symbolic links for which the target file or directory is owned by the same user id as the link
All	Everything except for MultiViews. This is the default value

Apache allows for decentralized management of configuration via special files placed inside the web tree. The special files are usually called **.htaccess**, but **any name** can be specified in the **AccessFileName** directive.

Directives placed in **.htaccess** files apply to the directory where the file is placed, and all sub-directories. **.htaccess** files follow the same syntax as main configuration files. Since **.htaccess** files are read on every request, **changes** made in these files **take immediate effect**. What directives may be placed in **.htaccess** files can be controlled by configuring the **AllowOverride** directive in the main configuration file **httpd.conf**.

These were the some of the **default settings** provided by Apache2. No changes are made in this file yet.

## VIRTUAL HOSTS

The **VirtualHost** directive in the **httpd.conf** configuration file is used to set the values of **ServerName**, **DocumentRoot**, **ErrorLog** and **TransferLog** or **CustomLog** configuration directives to different values for each virtual host.

Multiple websites can be served from one computer, when they have **different** hostnames. Each host name that is served from the single computer (that hosts them **all**), is referred to as a **virtual host**. There are **two** ways provided by Apache for setting up virtual hosts on a single computer, **IP based** and **Name based**.

IP based virtual hosts use the IP address of the connection to determine the correct virtual host. Hence, a **unique** IP address is required for each host.

With **name based** virtual hosting, Apache Web server relies on the client to deliver the hostname as part of the HTTP headers sent to Apache. Using this technique, many different virtual hosts can share the **same IP address**. Apache will do its own name resolution from the HTTP headers sent by the client's Browser.

## Caution



Older browsers do not support delivering a hostname with their HTTP headers. This is not part of their **HTTP 1.1** header encoding. Hence these browsers will only work with IP based virtual hosts.

Apache can be configured to support multiple virtual hosts either by running a **separate httpd** daemon for each hostname, or by running a **single httpd** daemon, which supports all the virtual hosts.

If separate httpd daemons must be run for each host, separate installations of Apache for each virtual host have to be created. For each installation, use the **Listen** directive in the **httpd.conf** configuration file to select which IP address or virtual host that daemon services

For example, Listen 192.168.0.1:80.

A single http daemon can also be used to service to the main server and **all** its virtual hosts.

## Name Based Virtual Hosts

Using the new name based virtual hosts is quite easy, and superficially looks like the old method. The notable difference between IP-based and Name-based virtual host configuration is the **NameVirtualHost** directive, which specifies a single IP address that should be used as a target for name-based virtual hosts.

For example, suppose that both `sct.perlproj.com` and `sct.phpproj.com` point to the IP address 172.16.9.66. Then simply add the following to the Apache's `httpd.conf`:

```
NameVirtualHost 172.16.9.66
<VirtualHost 172.16.9.66>
    DocumentRoot c:\sct\perlproj
    ServerName sct.perlproj.com
</VirtualHost>
<VirtualHost 172.16.9.66>
    DocumentRoot c:\sct\phpproj
    ServerName sct.phpproj.com
</VirtualHost>
```

The Following entries should be appended to the **hosts** file available under `C:\WINDOWS\system32\drivers\etc` directory:

```
172.16.9.66      sct.perlproj.com
172.16.9.66      sct.phpproj.com
```

```
httpd - Notepad
File Edit Format View Help
# The first VirtualHost section is used for requests without a known
# ServerName
#<VirtualHost *:80>
#   ServerAdmin webmaster@dummy-host.example.com
#   DocumentRoot "c:/www/docs/dummy-host.example.com"
#   ServerName dummy-host.example.com
#   ErrorLog "logs/dummy-host.example.com-error_log"
#   CustomLog "logs/dummy-host.example.com-access_log" common
#</VirtualHost>
# Added by Vaishali Shah For Name Based virtual Host
NameVirtualHost 172.16.9.66
<VirtualHost 172.16.9.66>
    DocumentRoot c:\sct\perlproj
    ServerName sct.perlproj.com
</VirtualHost>
<VirtualHost 172.16.9.66>
    DocumentRoot c:\sct\phpproj
    ServerName sct.phpproj.com
</VirtualHost>
```

Diagram 23.14: hosts file with name based virtual hosting

To test the name based virtual host settings create an index file in the root of each host as:

### Index.html at c:\sct\perlproj\

```
<HTML>
  <HEAD><TITLE>SCT PERL Project</TITLE></HEAD>
  <BODY>This is PERL Project</BODY>
</HTML>
```

Index.html at c:\sct\phpproj\

```
<HTML>
  <HEAD><TITLE>SCT PHP Project</TITLE></HEAD>
  <BODY>This is PHP Project</BODY>
</HTML>
```

Point the browser to both the URLs, i.e. one at a time as seen in diagram 23.15.1 and diagram 23.15.2



Diagram 23.15.1: Testing PERL project

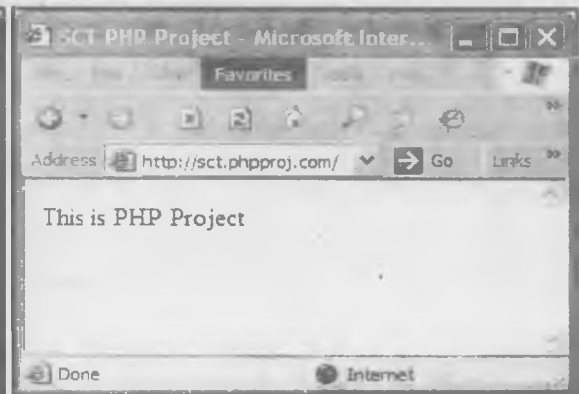


Diagram 23.15.2: Testing PHP project

## APACHE MODULES

Apache modules are code segments that are written to work with Apache because they are written using Apache's API specifications. They can be loaded statically into the Apache Web server executable by being added using appropriate instructions when compiling the Apache source files on Linux. The tool called **apxs** (APache eXtension) tool, allows the building and installation of an Apache extension module during source compile time.

Alternatively Apache modules can be dynamically loaded via Apache's httpd.conf configuration file, also known as **DSO** (Dynamic Shared Object) loading, which is a fine feature of Apache.

**DSO** allows the extension of the features and capabilities of Apache by adding specific modules **on demand, without** recompiling Apache's binary executable. Use the **LoadModule directive** placed within Apache's httpd.conf file to load a Dynamic Shared Object into the Apache Web Server namespace at startup time.

For all this to work properly the O/s platform on which Apache is installed has to support DSO features and the Apache **httpd** binary has to be built (i.e. compiled from source) along with the **mod\_so** module.

### Changes in the httpd.conf File For The Framework

Create a directory called **wampp** under C:\sct.

The following entries should exist in the **httpd.conf** file, found in, **conf** directory:

```
ServerName 172.16.9.66 (Here enter the actual IP of the host computer this IP is the one we used)
NameVirtualHost 172.16.9.66
<VirtualHost 172.16.9.66>
  ServerAdmin webmaster@sct.com
  DocumentRoot c:\sct\wampp
  ServerName sct.wampptraning.com
</VirtualHost>
```

## Sample Of The hosts File For The Framework

The Following entry should be appended to the **hosts** file, found in, /etc directory:  
172.16.9.66                    sct.wampptesting.com

## Registering The Changes Made In The httpd.conf With Apache2

After making any changes to the **httpd.conf** ensure that Apache is restarted to apply the new changes. This can be done by using the icon on the task bar: (Refer Diagram 23.16)



Diagram 23.16: Restarting Apache2

## SELF REVIEW QUESTIONS

### FILL IN THE BLANKS

1. Configuration files can be checked for syntax errors without starting the server by using \_\_\_\_\_ or the \_\_\_\_\_ command line option.
2. The \_\_\_\_\_ and \_\_\_\_\_ directives map a URL path to a directory on the server.
3. <Directory> is a \_\_\_\_\_ directive.
4. Apache modules are code segments are written using \_\_\_\_\_ specifications.
5. The \_\_\_\_\_ and \_\_\_\_\_ directives indicate the Server-pool size regulation.
6. The \_\_\_\_\_ indicates the absolute path of the document tree.
7. Apache 2.0.50 fixes security problems described in \_\_\_\_\_ and \_\_\_\_\_.
8. The \_\_\_\_\_ directive indicates the number of seconds to wait for the next request from the same client on the same connection.
9. The \_\_\_\_\_ tool allows the building and installation of an Apache extension module during source compile time.

### TRUE OR FALSE

10. Apache and either MySQL are from the open source domain.
11. The latest version of Apache is 1.3.X.XX-X.
12. Apache 2.0 add-in modules are compatible with Apache 1.3 modules.
13. The various directives that end in Log control indicate whether log files exist at all.
14. Apache disallows decentralized management of configuration via special files placed inside the web tree.
15. Multiple websites can be served from one computer, even if they have different hostnames.
16. A unique IP address is required for each host.
17. The ServerAdmin directive sets the hostname the server will return.
18. The ServerName accepts the Email address, where problems with the server should be e-mailed.
19. A Web server (*Apache2*) cannot communicate directly with a database management system (*MySQL*).

## C. PROJECTS IN PERL

### BUILDING A WEB SITE REGISTRATION SYSTEM

Since the learning of PERL is now completed, it is time to consolidate this learning by building a few utilities that would be excellent additions for any web site. Such as:

- A Web Site Registration system
  - A Login Page, mapped to challenge and reply sub-system
- A Pen Pals system
- A Guest Book

#### Web Site Registration

Many web sites require a site visitor to register with the site before being given access to all the sites resources. Being able to create a web site registration system to handle this will strengthen the PERL skills acquired earlier.

A registration form is used to capture, validate and store required visitor data at the Web site. The same storage system will be used to validate a user, when the user tries to login again to access the Web site's resources. To register, the HTML form show in diagram C.1.1 is used.

This form captures two blocks of information.

- A visitor's Login ID and Password used to access web site services in future
- The services that the visitor wishes to sign up for

Silicon Chip Technologies  
**Register For Our Goodies**

Login Name: \_\_\_\_\_  
E-Mail Address: \_\_\_\_\_  
Password: \_\_\_\_\_  
Confirm Password: \_\_\_\_\_

PASSWORD RULES

**Select From The Spread Below**

Pen Pals  News Letter

Submit | Reset | Abort

Diagram C.1.1: The Registration User Interface

There is some data validation done at the client's browser. This is handled by Java Script embedded within the HTML page run on the client. Valid data received from the client browser is processed at the Web Server using PERL codebase contained in a file.

After this information is processed at the Web server it is stored on the Web server's hard disk for future reference in a Text file in PERL.

#### The User Interface

Has three buttons, they are:

<b>Submit</b>	Submits form data to the Web server for further processing
<b>Reset</b>	Clears the data keyed in form
<b>Abort</b>	Moves the user back to the web sites Home page from the current form

### Message Of Thanks

A HTML page, which is a message of **THANKS**, is returned to the client browser when the Goodies Registration information is successfully received, processed and stored at the Web Server as shown in diagram C.1.2.

This form has a **Home** button, which takes the user back to the web sites Home page.

### In Case Of An Erroneous Submission

If a client submits registration data, and an error occurs, such as a duplicate data being submitted, this error is trapped and a suitable **Error message** HTML page is dispatched back to the client's browser as shown in diagram C.1.3.

This form has a **Back** button, which takes the user back to the registration form, from where the data was submitted.

### In Case Of Other Errors

In case of any other type of error, the HTML page, as shown in diagram C.1.4, is displayed.

### The Data Stored

The data captured by the User Interface is stored in a simple text file on the Server's HDD by PERL codebase.

The text file stores details such as:

name	Name of the visitor who has registered
emailid	The Email Id of the visitor
password	The password of the visitor
newsletter	A service for which the user has registered
penpals	A service for which the user has registered

The data captured by the user interface must be stored in a Text file table on the Web server's hard disk.



Diagram C.1.2: Message Of Thanks HTML Page



Diagram C.1.3: The Duplicate Data Entered HTML page



Diagram C.1.4: Non duplicate Entry Error Message



## WEB SITE LOGIN

Whenever a user attempts to log into the web site and use any of its services for which prior registration is required, a login, challenge / response, system goes into action. This starts by a login page being sent to the user's browser by the Web server.

The Login form, another very commonly found HTML form on an interactive Web site, is used to allow registered users (i.e. users registered via the Register Form dealt with earlier) to login and access the services provided by the Web Site.

Once the Login name and Password of a registered user is validated, instant access to the site's services is available.

### Note



The Login form also allows an unregistered visitor to immediately register for the services that the web site provides.

This form captures the visitors Login ID and Password and returns this to the Web server for further processing at the Web server end.

Server side PERL code is required to process the data returned by the Login form.

### Server Side Processing In Brief

Essentially, the Server side, PERL code, for the login page will compare the form data submitted against data already stored in a Text file thus validating the data submitted. The Text file is where the Login IDs and Passwords of all registered users are stored.

### Client Side Processing In Brief

Client side data validation is done via Java Script embedded in the HTML page. The Java Script ensures that the user leaves none of the mandatory text boxes empty. That the Email ID entered contains at least one @ and one period (.).

After this client side data validation the Login form data is dispatched from the user's browser to the Web server for further processing.

A Login form created captures:

- The Login ID of a user
- The Password of a user

The Login form is as shown in diagram C.2.1.

Diagram C.2.1: The Login User Interface

Diagram C.2.2: The Error Message HTML page

## The User Interface

Has three buttons, they are:

<b>Submit</b>	Submits Login form data to the Web server for further processing
<b>Reset</b>	Clears the data keyed in the Login form
<b>Sign Up!</b>	Opens the registration form to register with the Web site as shown in diagram C.2.1

The data captured by the Login form is verified against data held in the registration Text file. If the Login name is entered correctly then the Password for the corresponding Login name is checked. Access to web site resources is allowed only if the **combination** is correct.

### In Case Of An Erroneous Submission:

If the Login ID and password provided do not match with the ones present at the Web server (i.e. the Login ID / Password has not been registered earlier) then an **Error message** HTML page is displayed as shown in diagram C.2.2.

This form has a **Back** button, which takes the user back to the web site's Login Page. As mentioned earlier from the Login page New Registrations are accepted.

### In Case Of Other Errors

In case of any other type of error a form is displayed as shown in the diagram C.2.3. This is to indicate some kind of error, other than the Login ID and Password not being registered with the Web site, has occurred.

This form has a **Home** button, which takes the user back to the web site's Home Page.

### The Welcome User HTML Page

After submitting the Login ID and Password **and** if both are registered with the Web site then the user is presented a Welcome page.

This page has links to **Enter Pen Pals** or **Sign Guestbook** of the website. Via this page a user can **logout** by clicking the **Logout** button, which logs out a user and takes the user back to the web site, Home page.



Diagram C.2.3: The Error Message HTML page



Diagram C.2.4: Welcome Message HTML page

## PEN PALS

Building an interactive Pen Pal service for a Web site is another exercise in strengthening PERL skills learned earlier. Several PERL programs, placed on a Web server, are responsible for making the pen pal service work properly. Using this service, visitors can post their Name, Email id, Sex, Date of birth, Interests and Hobbies on the Web Site.

Other site visitors can then refer to these pages and send emails to individuals whose Age, Sex, Interests and other criteria either match up or are interesting.

There is some data validation done at the client browser. Java Script embedded within the HTML page run on the client handles this. Valid data dispatched from the client is then processed at the Web server through PERL code spec. After processing the data returned by the visitor's browser, it is stored either in a Text file on the Web server's hard disk for further reference.

Visitors to the site view pen pal information via PERL code spec. The PERL code spec first dispatches an HTML page to a client browser that captures an Age range, Sex, Interests and other such information, which will then be matched in the Text file.

If several matches are found they are properly formatted and returned back to the browser that requested for such information as an HTML page. Site visitors then read through these pages and can send Emails anyone whose Age, Sex, Interests and other criteria are of interest.

To post Pen Pal Information on the Web Site a simple HTML form needs to be filled in with details as shown in diagram C.3.1.

### The User Interface

Has three buttons, they are:

- Submit** submits form data to the web Server for further processing
- Reset** clears the data keyed in the form
- Abort** moves the user back to the Home page from the current form

### Message Of Thanks

A HTML page with message of **THANKS** is displayed when pen pal information is successfully submitted as shown in diagram C.3.2.

This form has a **Home** button, which takes the user back to the **Home** page.

### Note



This form has a **View** button via which Pen Pal listing can be viewed.

Diagram C.3.1: The User Interface

Diagram C.3.2: The HTML Thanks Page

## In Case Of An Erroneous Submission

If a duplicate entry into the Penpals list is attempted, this is trapped and a suitable **Error page** is displayed as shown in the diagram C.3.3. This form has a **Back** button, which takes the user back to the Penpals registration form, from where the data was submitted.

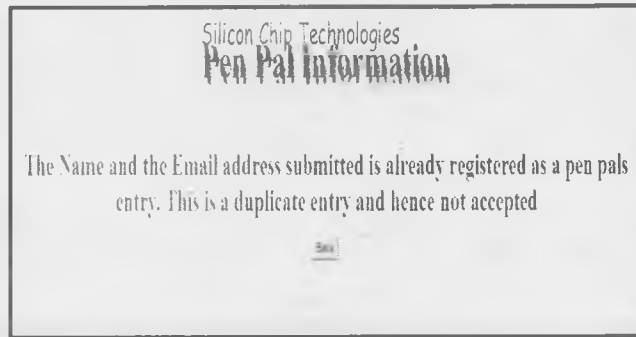


Diagram C.3.3: Duplicate Data Message

## In Case Of Other Errors

In case of any other type of error, a suitable **Error page** is displayed as shown in the diagram C.3.4. This is to indicate some kind of error other than the Duplicate Record error.

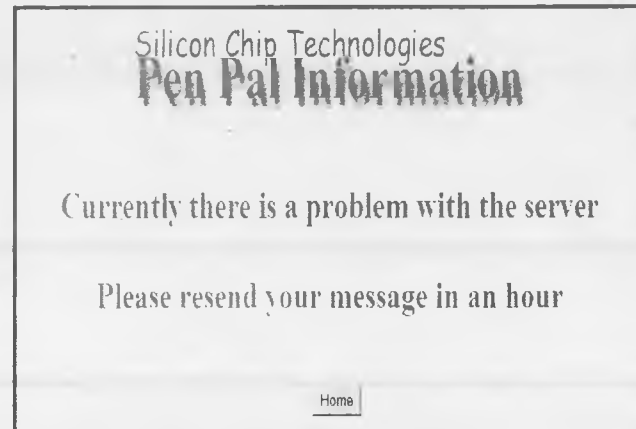


Diagram C.3.4: HTML Error Page

## Data Storage System

The data captured by the User Interface is stored in a simple Text file on the Web server's HDD by PERL code base.

The Text file stores details such as:

Memory Variable	What it holds
postdate	The Date on which the Pen Pal Information was posted (Pen Pal Information is valid for <b>one year</b> from the first date of placement)
name	The Name of the visitor who has placed the Pen Pal Information
emailid	The Email id to which replies must be sent
sex	The Sex of the visitor Sex [Male / Female] is captured by radio buttons hence no validation code required
dob	The Date of Birth of the visitor
age	The Age of the visitor
dtls	The Pen Pal Information entered by the visitor
moreinfo	A flag storing information about visitor's choice to subscribe

## Note



PERL code base on the Web server actually **calculates the Age** of the visitor on the basis of data of birth dynamically when required.

### View Pen Pal Information

After pen pal information has been successfully added, View the data submitted by clicking on the **View** button as shown in diagram C.3.5. Prior viewing the contents of the Pen pals, there is a filter page that allows a viewer to filter Pen pal information retrieved based on:

Sex	Age
-----	-----

Using this filter a viewer can view the pen pal information of choice.

### Pen Pal Information Display Format

The Email ID of the pen pal appears as a hot spot on **top** of the **general comments** retrieved and displayed from the text file as shown in diagram C.3.6. This makes it convenient to write to a pen pal through the pen pal Information page.

Pen pal Information remains on display for a **maximum period of one year**. Hence, the date on which the information was first received **must** be stored.

### The User Interface

Has two buttons, which are:

<b>Re-Search</b>	Takes back to the Search Form as seen in diagram C.3,5
<b>Logout</b>	Logs out and takes to the index page of the web site

### GUEST BOOK

A Guest Book is used to store comments that a visitor may have regarding the web site. The comments can help improve the functionality of a web site.

Since the HTML form captures user information, there is a small amount of data validation to be done on the client. Java Script embedded within the HTML page, run on the client's browser, handles the data validation. The data is then returned to the Web server. This data is appropriately processed at the Web server through PERL code spec.

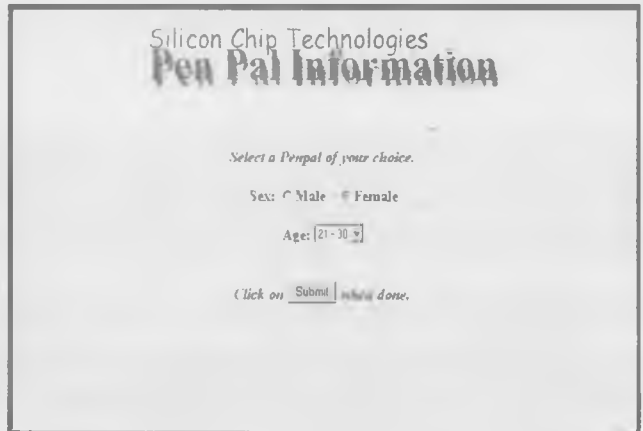


Diagram C.3.5: View Pen Pal Information HTML Page

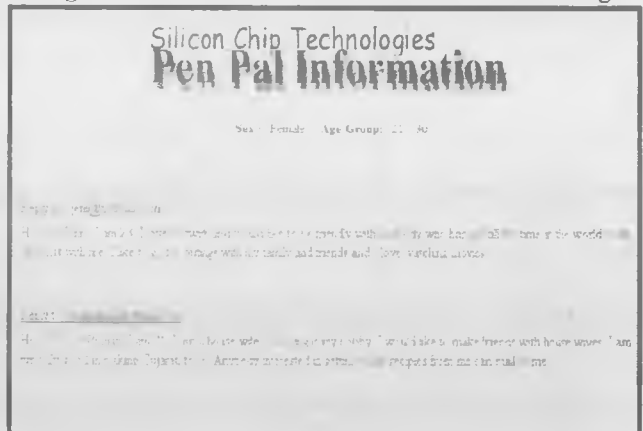


Diagram C.3.6: Display Of Pen Pals Information



Diagram C.4.1: The User Interface

Server side PERL code accepts input from the client browser and writes (**appends**) this input to a Text file on the Web server's hard disk for future reference.

An HTML form is created which captures the following:

- Visitors Name
- Email address
- Visitors comments

### Message Of Thanks

A HTML page with a message of '**Thanks! Your comments have been registered.**' is displayed when Guest Book information is successfully submitted as shown in diagram C.4.2.

This form displays a **Home** button, which takes the user back to the web site's **Home** page.

### In Case Of An Erroneous Submission

If while submitting the form an error occurs, this error is trapped and a suitable **Error** page is displayed. Refer to diagram C.4.3.

This also has a **Home** button, which takes the user back to the Home page.

### Data Storage System

The data captured by the U.I (User Interface) is appended and stored in a simple Text file on the Server's HDD by PERL code base. When required by the website administrator, the text file is downloaded from the Web server using FTP, and its contents are read in an ASCII editor and viewed. If there is sound criticism the Web site is updated as indicated. If major site errors were pointed out, the visitor should get a message of thanks.

The text file stores information such as:

Form Variable Name	Purpose
postdate	The date on which the visitor keyed in the comments
name	The name of the visitor who keyed in the comments
emailid	The Email Id of the visitor
comments	The critique keyed in by the visitor
moreinfo	A flag storing information about visitor's choice to subscribe

### The User Interface

Has three buttons, which are:

<b>Submit</b>	Transfers Form data to the Web Server for further processing
<b>Reset</b>	Clears the data keyed in the form
<b>Abort</b>	Moves the user back to the <b>Home page</b> from the current form and does not send anything back to the Web server for processing



Diagram C.4.2: Message of Thanks

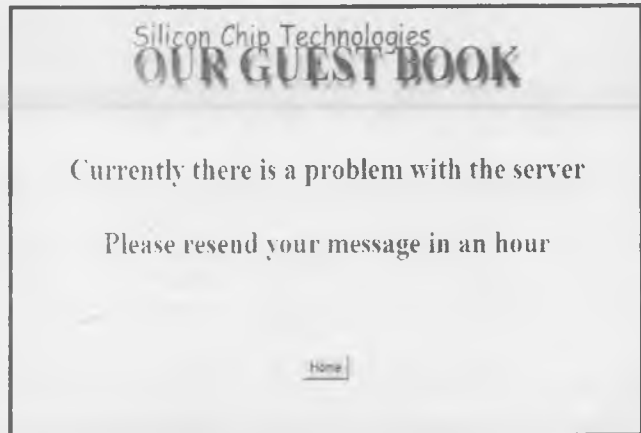


Diagram C.4.3: Error Message

## UNDERSTANDING THE WEB SITE REGISTRATION SYSTEM

### The Framework On Which A Web Site Registration System Is Built

The **Web Site Registration** system is developed using **HTML** forms as its **front-end**. **JAVASCRIPT** as the **client side** scripting language. **PERL** as a **server side** scripting language. **Text files** are demonstrated as **back-end** storage systems of choice. The system runs on an Intranet, composed of Windows based **clients** and a Windows XP **server**. The Web site registration forms can be called from any Windows based client on the Intranet. An **Apache2** Web Server, running on the Server, services all the clients on the Intranet. PERL scripts do all data processing at the Web server.

### Modules Under The Web Site Registration System

The system consists of the following modules:

- Web site registration module
- Login challenge and response module

The following services are accessible via the above modules:

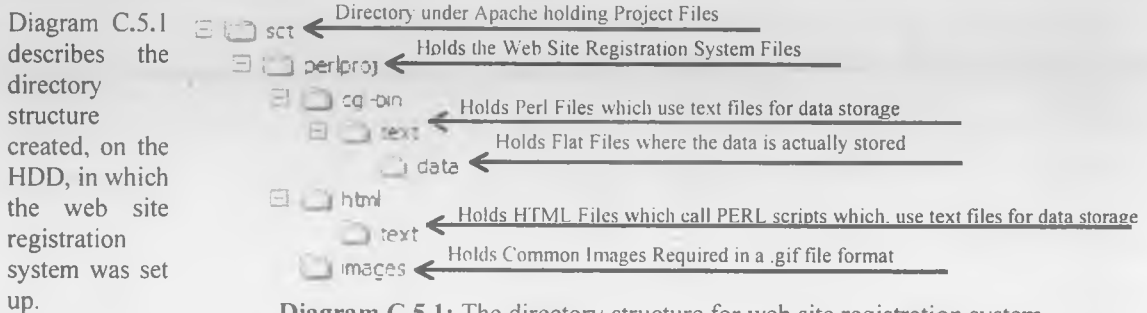
- Pen Pals
- Guest book

Described below is the directory tree structure, created under `c:\sct\`, which will hold the files used by each module in the web site registration system.

### Approach

The web site registration system has been build using the PERL server side scripting languages and FLAT/Text files as the type of storage systems.

### Directory Structure

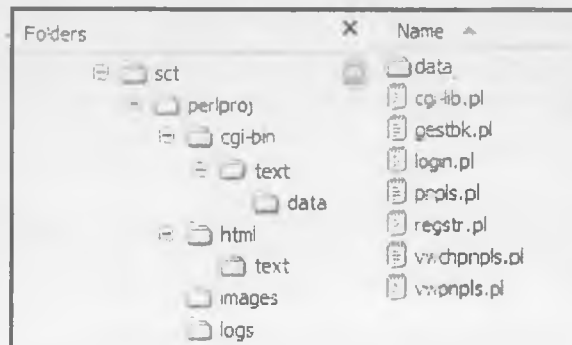


### Files Created For The Web Site Registration System

#### PERL Scripts

- `c:\sct\perlproj\cgi-bin\text`

These are the files, which do server side processing for the text based storage system. They are stored in `c:\sct\perlproj\cgi-bin\text`. They store the processed data, in **Text Files**. These data files are stored in the `c:\sct\perlproj\cgi-bin\text\data\`. (Refer diagram C.5.2)



## HTML Files

### □ c:\sct\perlproj\html\text\

The HTML forms, that act as the front-end of the Web site registration system are stored in **c:\sct\perlproj\html\text** for the text based storage system. They use JavaScript to do client side processing before sending form data to the Web server for final processing and storage. (Refer diagram C.5.3)

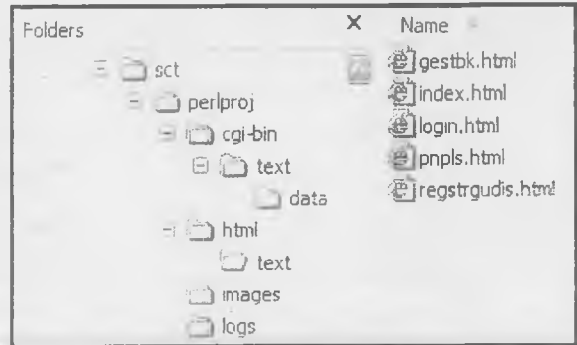


Diagram C.5.3: The location of HTML Text files

## Image Files (.gif)

### □ c:\sct\perlproj\images\

These are **.gif** files, which are embedded in HTML pages, which are used in the front end files for the Text based system. These files are used to make the HTML pages look attractive. (Refer diagram C.5.4)

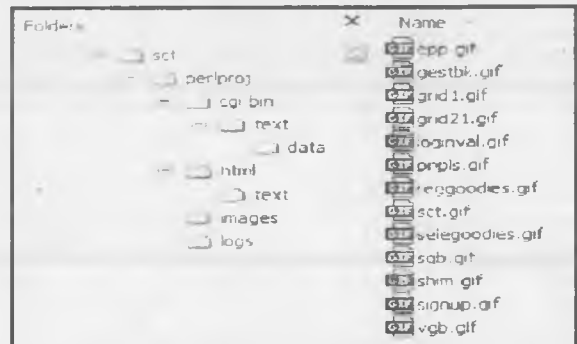


Diagram C.5.4: The location of the Image files

## Flat Files (Storage)

### □ c:\sct\perlproj\cgi-bin\text\data\

These are **<filename>.dat** files, which actually store the data processed by the PERL file on the Web server. The files contained in this directory should have **read and write** permissions set correctly in order to allow the PERL script files to store data in them. (Refer diagram C.5.5)

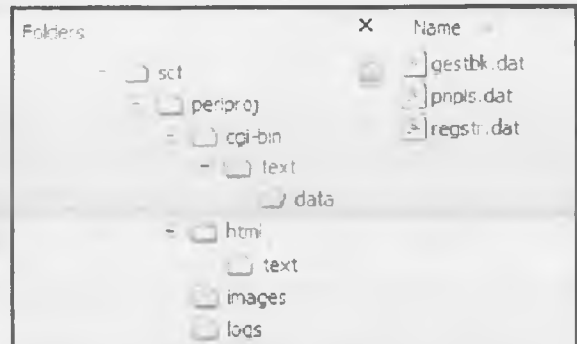


Diagram C.5.5: The location of the Flat files

## DATA STORAGE SYSTEM

### For Web Site Registration

The data captured by the **Registration Form** (regstrgudis.html) and used by **Login Form** (login.html) for reference, is stored in a simple text file on the Web Server's HDD by a PERL script.

The **regstr.dat** file stores details captured by the HTML form separated by a colon (:).

### For Pen Pals Registration

The data captured by the **Pen Pals Form** (pnpls.html) is stored in a simple text file on the Web Server's HDD by a PERL script.

The **pnpls.dat** stores details captured by the HTML form separated by a colon (:).



## For The Web Site Guest Book

The data captured by the **Guest Book Form** (gestbk.html) is stored in a simple text file on the Web server's HDD by a PERL script.

The **gestbk.dat** file stores details captured by the HTML form separated by a colon (:).

## STARTING THE WEB SITE REGISTRATION SYSTEM

To access the PERL based web site registration system a name based, virtual host entry is created under Apache2, pointing to a directory. This is followed by an entry in the **hosts** (c:\windows\system32\drivers\etc\hosts) file as follows:

```
172.16.9.66          sct.perlproj.com          (IP Address may differ on individual machine bases)
```

The following entries are made in the httpd.conf file as follows:

```
NameVirtualHost 172.16.9.66
<VirtualHost 172.16.9.66>
  ServerAdmin webmaster@sct.com
  DocumentRoot c:\sct\perlproj
  ServerName sct.perlproj.com
  ScriptAlias /cgi-bin/ "c:\sct\perlproj\cgi-bin\"
</VirtualHost>
```

### Note



The default ScriptAlias tag of Apache2 should be disabled by commenting the line in the httpd.conf. For example:  
#ScriptAlias /cgi-bin/ "C:/Program Files/Adobe Group/Adobe2/cgi-bin/"

To start a session of the Web site registration system, follow these steps:

- Open a browser
- Type in the url:

**http://sct.perlproj.com/html/text**

This opens the **index.html** page as seen in diagram C.6.1 allowing a login to the website or registration with the website.

## PROCESSING OF WEB SITE REGISTRATION

This provides an overview of how the code base of the Web Site Registration System is structured.

When **Login To SCT** option on the index page is activated then a login screen as shown in diagram C.6.2 appears prompting for a valid user name and password, required to access system resources.

### Form Layout

(Refer to diagram C.6.2)

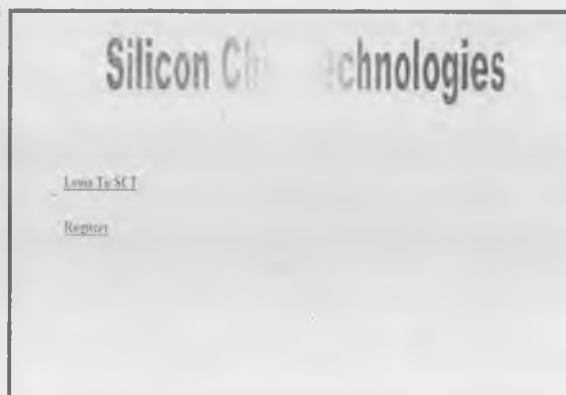


Diagram C.6.1: The index.html page

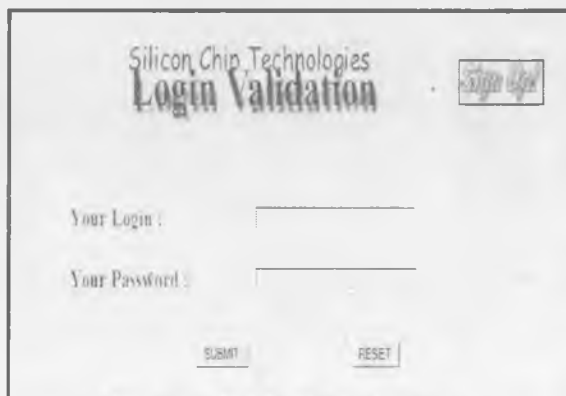


Diagram C.6.2: The login screen

**Purpose**

- Authenticates the user attempting the login
- Determines access to system resources like **Pen Pals** and **Guest Book**

**Processing**

The visitor can choose to do any of the following:

Key in the **Login ID** and **Password** and:

- Click **Submit**. This leads to validating the user name and the password provided. On submit a call is made to **login.pl**. Processing is now handed over to this file.
- Click **Reset**. This clears the field contents on the HTML Form

OR

Click **Signup!**. This opens up the Register For Our Goodies (regstrgudis.html) Form. This form will allow a visitor to register with the website and then login with the Login ID and Password just registered.

When the **login.pl** file is called:

- The CGI Library (**cgi-lib.pl**) is loaded in memory
- The text file (rgstr.dat) is loaded in memory for storing the data captured
  - A connection to <rgstr.dat> is established via a File Handle
- If connection is not successful then an error page is generated on the fly stating, *Currently there is a problem with the server. Please resend your message in an hour*
- The Login ID and Password captured via the login.html form is stored in memory variables
- Records are fetched records from the Text File (regstr.dat)
- The value held in the memory variables is compared with the data retrieved
- If a match is found then a welcome message is displayed by generating an html page on the fly. This page will have a link to enter the Pen Pals and Guest Book Section .
- If the match is not found then an error page stating *Sorry your login and password does not exist.* is dispatched to the client browser

If a visitor clicks **SignUp!** on the Login Form (login.html) then a Registration Form is invoked as shown in diagram C.6.3. This prompts for the details required for a valid registration.

**Form Layout**

(Refer to diagram C.6.3)

**Purpose**

- Prompt and validates for details required for a valid registration
- Stores the details captured in Text Files
- Provides access to system resources like **Pen Pals** and / or **Newsletter** if opted for

**Processing**

The visitor can choose to do any of the following:

Key in the desired **Login ID**, **Email Address** and **Password** and:

- Click **Submit**. This leads to registration of the details provided  
On submit a call is made to **regstr.pl** file. Server side processing is now handed over to the code spec in either of these files
- Click **Reset**. This clears the contents of all fields on the form
- Click **Abort**. This takes the visitor back to the Login Form

Silicon Chip Technologies  
**Register For Our Goodies**

Login Name: \_\_\_\_\_  
E-Mail Address: \_\_\_\_\_  
Password: \_\_\_\_\_  
Confirm Password: \_\_\_\_\_

**Select From The Spread Below**

Pen Pals  News Letter

[Submit](#) | [Reset](#) | [Abort](#)

**Diagram C.6.3: The registration screen**

When the **regstr.pl** file is called:

- The CGI Library (**cgi-lib.pl**) is loaded in memory
- The text file (rgstr.dat) is loaded in memory for storing the data captured
  - A connection to <rgstr.dat> is established via a File Handle
- If any connection is not successful then an error page is generated on the fly stating, Currently there is a problem with the server. Please resend your message in an hour
- The data captured in the form fields in regstrgudis.html is stored in memory variables
- Data is fetched from <rgstr.dat>
- Values held in the memory variables are compared with the data retrieved from the storage system (i.e. either a text file or a database table)
- If the values held in the variables for **Login ID** and **Email address** match the data retrieved from the storage system then an error page stating The Login ID and Password submitted is already registered with this Web Site. This is a duplicate entry and hence not accepted is dispatched to the user and the process of registration is aborted. At this point of time the visitor can **re-enter** data and **re-submit**. This will again **re-process** the data as described above
- If no duplicate data is encountered then the data captured, held in variables is transferred to the storage system. This is followed by a html page being sent back to the user with a message stating Thank You! Your information has been registered!
- The page rendered **may or may not** have an option to **Enter Pen Pals** section. This depends on the options selected in the Registration Form. Similarly if Newsletter option was selected then a message stating Newsletters will be sent shortly at <email address specified on the registration form>
- The **Enter Guest Book** option will be available irrespective of what was selected

A visitor after logging in or signing up with the website can use website resources like **Pen Pals** or **Guest Book** or subscribe to its **newsletters**. If the Pen Pals section was opted then the Pen Pals Form is invoked as seen in diagram C.6.4 prompting for details required for a valid Pen Pals Entry.

### Form Layout

(Refer to diagram C.6.4)

Diagram C.6.4: The Pen Pal Information screen

### Purpose

- Prompt and validates for details required for a valid entry in Pen Pals section
- Stores the details captured in a Flat Text File
- Calculates and Stores a visitor's age on the basis of the Date of birth provided

### Processing

The visitor can choose to do any of the following:

Key in the appropriate **Name**, **Email Address**, **Sex**, **Date of birth** and **Hobbies** and **other interests** and:

- Click **Submit**. This leads to registration of the data captured
  - On submit a call is made to **pnpls.pl** file. Server side processing is now handed over to the code spec of either of these files
- Click **Reset**. This clears the field contents on the Form
- Click **Abort**. This takes the visitor back to the page from where the pen pals option was activated

When either the **pnpls.pl** file is called:

- The CGI Library (**cgi-lib.pl**) is loaded in memory
- The text file (pnpls.dat) is loaded in memory for storing the data captured
  - A connection to <pnpls.dat> is established via a File Handle
- If any connection is not successful then an error page is generated on the fly stating, *Currently there is a problem with the server. Please resend your message in an hour*
- The data captured in the form fields via pnpls.html is stored in memory variables
- The age of the visitor is calculated on the basis of the date of birth provided and stored in a variable
- Records are fetched from the Text File (pnpls.dat)
- The value held in the memory variables is compared with the data retrieved storage
- If the values held in the memory variables match the data retrieved from the storage system then an error page stating *The Name and Email Address submitted is already registered as a Pen Pals Entry with this Web Site. This is a duplicate entry and hence not accepted* is sent back to the client's browser. The process of registration is aborted. At this point of time a visitor can **re-enter** data and try **re-submitting** it. This will **re-process** the data as described above
- If no duplicate data is encountered then the data captured, and held in memory variables is transferred to the storage system. This is followed by an html page being dispatched to the visitor's browsers stating *Thank You! Your information has been registered!*

This page has two buttons i.e. **View** and **Home**.

- View** passes the control to the PERL script **vwchpnpls.pl**. This script is responsible for generating an html page which allows viewing pen pals registered on basis of some search criteria
- Home** takes the visitor back to the web site's Home page

When either **vwchpnpls.pl** is called:

- The CGI Library (**cgi-lib.pl**) is loaded in memory
- An html page dispatched to the visitors browser, which will allow the visitor to select search criteria (i.e. sex and a specific age range) to scan the pen pals database for matches. This is done by radio buttons to select sex and a list box showing age range to choose from
- On submitting this criteria control is passed to either **vwpnpls.pl**. These scripts are responsible for searching the storage system using the search criteria specified and displaying the data retrieved via a neatly formatted HTML page

When **vwpnpls.pl** is called:

- The CGI Library (**cgi-lib.pl**) is loaded in memory
- The text file (pnpls.dat) for storing the data captured
  - A connection to <pnpls.dat> is established via a File Handle
- If any connection is not successful then an error page is generated on the fly stating, *Currently there is a problem with the server. Please resend your message in an hour* and sent back to the visitors browser
- The form fields data captured is stored in memory variables

The **Email ID** displayed in the HTML page is a **HOTSPOT**. This will invoke the visitors default Email client with the email address already entered in the address bar. This permits sending an email to a pen pal while viewing the data displayed.

After viewing the data the visitor can move back to the previous page via the **Back** button to search for pen pals based on some other criteria.

If the Guest Book section was opted then the Guest Book Form is invoked as seen in diagram C.6.5 prompting details required for a valid Guest Book Entry.

## Form Layout

(Refer to diagram C.6.5)

### Purpose

- Prompt and validates the details required for a valid entry in Guest Book section
- Stores the details captured in a Text File

### Processing

The visitor can choose to do any of the following:

Key in the appropriate **Name**, **Email Address**, **some comments** and:

- Click **Submit**. The data entered in saved to the web site's Guest book  
On submit a call is made to either **gestbk.pl** file. Server side processing is now handed over to the code spec of either of these files
- Click **Reset**. This clears all the field contents of the form
- Click **Abort**. This takes the visitor back to the page from where the guest book option was activated

When the **gestbk.pl** file is called:

- The CGI Library (**cgi-lib.pl**) is loaded in memory
- The text file (gestbk.dat) for storing the data captured
  - A connection to <gestbk.dat> is established via a File Handle
- If any connection is not successful then an error page is generated on the fly stating, *Currently there is a problem with the server. Please resend your message in an hour* and sent back to the visitors browser
- The data captured in the form fields via gestbk.html is stored in memory variables
- The visitor's age is calculated (on the basis of Date of birth provided) and stored in a memory variable
- The data captured and held in memory variables is transferred to the storage system. This is followed by an html page being dispatched to the visitors browsers stating *Thank You! Your information has been registered!*

Silicon Chip Technologies  
**OUR GUEST BOOK**

Please take a few moments to let us know you were here today

Please Give Us Your Name

Please Give Us Your Email Address

Bouquets Or Brickbats Are Welcome

Can we contact you with information about our products or services?  
Yes No (Yes/Thank)

Submit Reset Abort

Thank You For Stopping By Our Web Site

Diagram C.6.5: The Guest book screen

## PROJECT SOURCE CODE FOR THE INDEX PAGE

### Source Code For index.html

```
<!-- This HTML page will accept a Login Name and Password. Clicking on
its Submit button causes login page data to be passed to Login.class
for further processing. -->
```

```
<HTML>
```

```
<HEAD><TITLE>Welcome to SCT's Web Site</TITLE></HEAD>
```

```
<BODY Background = "/images/grid1.gif"><CENTER>
```

```
<TABLE Border = "0"><TR>
```

```
<TD><IMG Src="/images/shim.gif" width="100"></TD>
```

```
<TD><IMG Src="/images/sct.gif"></TD>
```

```
<TD><IMG Src="/images/shim.gif" width="100"></TD>
```

```
</TR></TABLE><BR><BR><BR>
```

```
<TABLE Align = "Center" Border = "0" Width = "60%"><TR>
```

```
<TD Colspan = "1" Width = "40%"><FONT Id = spl Color = "Green" Size = "5">
```

```
<A Href="/html/text/login.html"><B>Login To SCT </B></A>
```

```
</FONT></TD>
```

```

</TR><TR><TD Colspan = "2">&nbsp;  </TD></TR>
<!-- Row to enter the Password -->
<TR>
  <TD Colspan = "1" Width = "40%"><FONT Id = spl Color = "Green" Size = "5">
    <A HRef='html/text/reqstrgudis.html'><B>Register</B></A>
  </FONT></TD>
</TR></TABLE>
</CENTER></BODY>
</HTML>

```

## PROJECT SOURCE CODE FOR THE WEB SITE REGISTRATION SYSTEM

### Source Code For reqstrgudis.html

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
  <HEAD><TITLE>SCT'S GOODIES SIGN UP FORM</TITLE>
  <META content="text/html; charset=windows-1252" http-equiv=Content-Type>
  <SCRIPT language=JavaScript>
    // Declaration of Variables
    var value_of_pass1="";
    var value_of_pass2="";
    var whitespace = "\t\n\r";
    // Function to check whether the value in a Text Field is Null
    function isEmpty(s) {
      return ((s == null) || (s.length == 0))    }
    // Function to check whether the value in a Text Field
    // is a WhiteSpace
    function isWhitespace (s) {
      var i;
      // Is s empty?
      if (isEmpty(s)) return true;
      // Search through string's characters one by one until we find
      // a non-whitespace character.  When we do, return false;
      // if we don't, return true.
      for (i = 0; i < s.length; i++) {
        // Check that current character isn't whitespace.
        var c = s.charAt(i);
        if (whitespace.indexOf(c) == -1) return false;
      }
      // All characters are whitespace.
      return true;
    }
    // Function to ensure that the email address is in proper format
    function isEmail (s) {
      if (isEmpty(s))
        // Is s whitespace?
        if (isWhitespace(s)) return false;
      // There must be >= 1 character before @, so we start looking
      // at character position 1, (i.e. second character)
      var i = 1;

```

```

        var sLength = s.length;
        // Look for @
        while ((i < sLength) && (s.charAt(i) != "@")) {
            i++
        }
        if ((i >= sLength) || (s.charAt(i) != "@"))      return false;
        else      i += 2;
    // Look for .
        while ((i < sLength) && (s.charAt(i) != ".")) {
            i++
        }
    // There must be at least one character after the
        if ((i >= sLength - 1) || (s.charAt(i) != "."))      return false;
        else      return true;
    }
// Function to check whether the value in Password
// contains Alphabets and Charcters
function isCharsInBag (s, bag) {
    var i;
    // Search through string's characters one by one.
    // If character is in bag, append to returnString.
    for (i = 0; i < s.length; i++) {
        var c = s.charAt(i);
        if (bag.indexOf(c) == -1) return false;
    }
    return true;
}
// Function to check whether the Password
// contains atleast one number
function isNumberInPass (s, bag) {
    var i,flag;
    flag=0;
    // Search through string's characters one by one.
    // If character is in bag, append to returnString.
    for (i = 0; i < s.length; i++) {
        var c = s.charAt(i);
        if (bag.indexOf(c) == -1) {      continue;      }
        else {
            flag=1;
            break;
        }
    }
    if(flag == 1) {
        return true; }
    else {
        return false; }
    return false;
}
// Function to check the values entered in all the elements
// of the form called on the clicked event of the Submit button
function verify() {
    var flag =0;

```

```
// Check to see if any of the Text fields is left blank
for (i=0; i<=3; i++) {
    if (document.forms[0].elements[i].value == "") {
        alert("Please fill in the " + document.forms[0].elements[i].name + " field");
        document.forms[0].elements[i].focus();
        flag =1;
        break;
    }
}
if (flag == 1) {
    return( false); }
// Beginning the check for Email address, Password
// and Confirm Password
if (flag == 0) {
    var email = document.forms[0].elements[1].value;
    // Validate the email address
    if (!isEmail(email)) {
        alert("Please enter the Email address in the proper Format");
        document.forms[0].elements[1].focus();
        return false;
    }
    var passwd = document.forms[0].elements[2].value;
    // Validate the Password
    if (!isCharsInBag( passwd,
        "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789")) {
        alert( "Password must only contain alphabets and number");
        return false;
    }
    // Check to ensure that the Password is not less
    // than 5 characters
    else if ( passwd.length < 5 ) {
        alert( "Password must be 5 or more characters.");
        return false;
    }
    // Check to ensure that the Password is not greater
    // than 8 characters
    else if ( passwd.length > 8) {
        alert("Password cannot be more than 8 characters.");
        return false;
    }
    //Check to ensure that the Password
    // contains atleast one number
    else if (!isNumberInPass( passwd, "0123456789")) {
        alert( "Password must have atleast one number.");
        return false;
    }
    //Check to ensure that the Password fields have the same value
    else if (document.forms[0].elements[2].value != document.forms[0].elements[3].value ) {
        alert("Your passwords do not match. Please retype and try again.");
    }
}
```



```

        return false;
    }
}
//Check to ensure that atleast one of the Check Boxes is checked
for (j=4; j<=5; j++) {
    if(document.forms[0].elements[j].checked) {
        break; }
    else if (j>=5) {
        alert("Atleast Check on One of Our Services");
        document.forms[0].elements[j].focus();
        return (false);
    }
}
return(true);
}
// Function called on the Clicked event of the Abort button
function abort() {
    history.back(); }
</SCRIPT><META content="MSHTML 5.00.2014.210" name=GENERATOR></HEAD>
<BODY background="/images/grid1.gif">
    <CENTER><IMG src="/images/reggoodies.gif"> </CENTER>
    <FORM action="/cgi-bin/text/regstr.pl" method="post" onsubmit="return verify()">
        <TABLE align=center border=0 cellPadding=0 cellSpacing=0 Size="800">
            <TBODY><TR>
                <TD align=right><B>Login Name:</B></TD>
                <TD><INPUT name=login></TD>
            </TR><TR>
                <TD align=right><B>E-Mail Address:</B></TD>
                <TD><INPUT name=email></TD>
            </TR><TR>
                <TD align=right><B>Password:</B></TD>
                <TD><INPUT name=passwd1 type=password></TD>
            </TR><TR>
                <TD align=right><B>Confirm Password:</B></TD>
                <TD><INPUT name=passwd2 type=password></TD>
            </TR></TBODY></TABLE>
            <P><TABLE align=center border=0 cellPadding=0 cellSpacing=0 Size="800">
                <TBODY><TR>
                    <TD><FONT color=red face=Verdana size=1>
                        <B>PASSWORD RULES:</B></FONT></TD>
                </TR><TR>
                    <TD><FONT color=red face=Verdana size=1>A Password's minimum length is five
                        characters</FONT></TD>
                </TR><TR>
                    <TD><FONT color=red face=Verdana size=1>A Password's maximum length is eight
                        characters</FONT></TD>
                </TR><TR>
                    <TD><FONT color=red face=Verdana size=1>A Password should have at least one digit
                        included</FONT></TD>
                </TR><TR>

```

```

<TD><FONT color=red face=Verdana size=1>Other than Alphabets and Digits no other
values are allowed</FONT></TD>
</TR></TBODY></TABLE>
<CENTER><IMG height=25 src="/images/shim.gif" width=1> </CENTER>
<CENTER><IMG src="/images/selegoodies.gif"> </CENTER>
<CENTER><IMG height=30 src="/images/shim.gif" width=1> </CENTER>
<TABLE align=center border=0 cellPadding=0 cellSpacing=0 Size="800">
<TBODY><TR>
<TD><IMG height=5 src="/images/shim.gif" width=10></TD>
<TD><B>Pen Pals</B><IMG height=1 src="/images/shim.gif" width=3>
<INPUT type=checkbox name=penpals value="y"></TD>
<TD><IMG height=5 src="/images/shim.gif" width=10></TD>
<TD><B>News Letter</B><IMG height=1 src="/images/shim.gif" width=3>
<INPUT type=checkbox name=newsltr value="y"></TD>
<TD><IMG height=5 src="/images/shim.gif" width=10></TD>
</TR><TR>
<TD><IMG height=30 src="/images/shim.gif" width=1></TD>
</TR></TBODY></TABLE>
<P><P><CENTER>
<INPUT type=submit value=Submit>
<INPUT type=reset value=Reset>
<INPUT onclick=abort() type=button value=Abort><BR>
</CENTER>
</FORM>
</BODY>
</HTML>

```

### Source Code For regstr.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
##### Print the HTTP content-type header #####
&PrintHeader;
##### A variable declared to store the path to the text file #####
$basedir = "c:/sct/perlproj/cgi-bin/text/data/";
##### A variable declared to store the text file #####
$datafile = "regstr.dat";
##### The current date is captured from the system #####
##### using function 'localtime' and necessary processing #####
##### is carried out to extract the date from the function ####
$slsh="/";
$slctim=localtime(time);
($sday,$smonth,$sdate,$stime,$syear)=split(/\s+/, $slctim);
%mon=("Jan"=>"01", "Feb"=>"02", "Mar"=>"03", "Apr"=>"04", "May"=>"05", "Jun"=>"06",
"Jul"=>"07", "Aug"=>"08", "Sep"=>"09", "Oct"=>"10", "Nov"=>"11", "Dec"=>"12");
for $key(keys%mon) {
    if ($smonth eq $key) {
        $smon=$mon{$key};    }
}
$cdt=$syear.$slsh.$smon.$slsh.$sdate;

```

```

##### The ReadParse subroutine is called from Cgi-lib.pl #####
##### library to decode form information #####
&ReadParse;
##### Assigns the input from the HTML form #####
#####to the specified variables #####
$name=$in{'login'};
$semid=$in{'email'};
$password=$in{'passwd'};
$pnpls=$in{'penpals'};
$nwltr=$in{'newltr'};
if($pnpls ne "y") {
    $pnpls="n";    }
if($nwltr ne "y") {
    $nwltr="n";    }
##### A check for the duplicate values in the text file is done. #####
##### In case of duplicate values or an internal server error #####
##### an error page is rendered and a flag is set to 1 #####
open(FH1,"<$basedir/$datafile");
@lines=<FH1>;
close(FH1);
foreach(@lines) {
    chop();
    @arr=(split(/:/));
    if ($arr[1] eq $name) {
        if($arr[2] eq $semid) {
            print "<HTML><HEAD><SCRIPT>";
            print "function back(form) { history.back(); }";
            print "</SCRIPT></HEAD>";
            print "<BODY Background=/images/grid1.gif text=green>";
            print "<CENTER><IMG Src=/images/reggoodies.gif></CENTER><BR><BR><BR>";
            print "<CENTER><H1>The Login ID and Password submitted is already registered with this
                Web Site. This is a duplicate entry and hence not accepted.</H1>";
            print "<FORM><INPUT Type=button Value=Back onClick=back(this.form)></FORM>";
            print "</CENTER></BODY></HTML>";
            $flag=1;
        }
    }
}
##### Appending the 'pnpls.txt' with the through the filehandle
if ($flag!=1) {
    if (open(FH1,">>$basedir/$datafile")) {
        print FH1 "$cdt:$name:$semid:$password:$pnpls:$nwltr\n";
        print "<HTML><HEAD><SCRIPT>";
        print "function home(form) { history.back(); }";
        print "</SCRIPT></HEAD>";
        print "<BODY Background=/images/grid1.gif text=green>";
        print "<CENTER><IMG Src=/images/reggoodies.gif></CENTER><BR><BR><BR>";
        print "<CENTER><H1>Thank You !</H1><BR>";
        print "<H1>Your information has been registered !</H1>";
        print "<FORM>";
    }
}

```

```

if($pnpls ne "n") {
    print "<BR><A HREF=/html/text/pnpls.html><IMG
        Src=/images/epp.gif></A><BR><BR>";
}
if($nwltr ne "n") {
    print "<BR><BR>Newsletters will be sent shortly at ".$Semid;
}
print "<BR><A HREF=/html/text/gestbk.html><IMG Src=/images/sgb.gif></A><BR><BR>";
print "<INPUT Type=button Value=Home onClick=home(this.form)>";
print "</FORM></CENTER></BODY></HTML>";
}
else {
    print "<HTML><HEAD><SCRIPT>";
    print "function home(form) { history.back(); }";
    print "</SCRIPT></HEAD>";
    print "<BODY Background=/images/grid1.gif text=green>";
    print "<CENTER><IMG Src=/images/reggoodies.gif></CENTER><BR><BR><BR>";
    print "<CENTER><H1>Currently there is a problem with the server</H1><BR>";
    print "<H1>Please resend your message in an hour</H1><BR><BR>";
    print "<IMG Src=/images/shim.gif Width=20 Height=20>";
    print "<INPUT Type=button Value=Home onClick=home(this.form)></CENTER>";
    print "</FORM></BODY></HTML>";
}
}
}

```

## PROJECT SOURCE CODE FOR THE WEB SITE LOGIN

### Source Code For login.html

```

<!-- This HTML page will accept a Login Name and Password. Clicking on
its Submit button causes login page data to be passed to Login.class
for further processing. -->
<HTML>
    <HEAD><TITLE>Welcome to SCT's Login Authentication</TITLE>
    <SCRIPT Language = "JavaScript">
<!-- The function verify() checks whether data is filled in all the
elements. If any element is left empty, an alert() is displayed
informing the user to fill in the empty element. -->
    function verify(form) {
        for(i = 0; i <= 2; i++) {
            <!-- Checking for an empty field in the form. -->
            if(document.forms[0].elements[i].value == "") {
                <!-- Message indicating an empty field -->
                alert("Please Fill In Your " + document.forms[0].elements[i].name);
                <!-- Placing the form cursor on the appropriate blank field
                -->
                document.forms[0].elements[i].focus();
                return(false);
            } // End of if construct.
        } // End of for iteration.
        <!-- Sending the data captured by the form to the server. -->
        document.forms[0].submit();
    } // End of function verify().

```

```

</SCRIPT></HEAD>
<BODY Background = "/images/grid1.gif"><CENTER>
  <TABLE Border = "0"><TR>
    <TD><IMG Src = "/images/shim.gif" width="100"></TD>
    <TD><IMG Src = "/images/loginval.gif"></TD>
    <TD><IMG Src = "/images/shim.gif" width="100"></TD>
    <TD><A HREF='regstrgudis.html'><IMG Src = "/images/signup.gif"></A></TD>
  </TR></TABLE><BR><BR><BR>
  <FORM Action = "/cgi-bin/text/login.pl" Method="POST">
    <TABLE Align = "Center" Border = "0" Width = "60%">
      <!-- Row to enter the Login ID -->
      <TR>
        <TD Colspan = "1" Width = "40%">
          <FONT Id = spl Color = "Green" Size= "5"><B>Your Login :</B></FONT>
        </TD><TD><INPUT Name = "login" Size = "35" Type = "Text"></TD>
      </TR><TR><TD Colspan = "2">&nbsp;</TD></TR>
      <!-- Row to enter the Password -->
      <TR>
        <TD Colspan = "1" Width = "40%">
          <FONT Id = spl Color = "Green" Size= "5"><B>Your Password :</B></FONT>
        </TD><TD><INPUT Name = "passwd" Size = "35" Type = "Password"></TD>
      </TR></TABLE><BR><BR>
      <TABLE Align = "Center" Border = "0" Width = "40%">
        <!-- Row for the 'Submit' and 'Reset' buttons -->
        <TR>
          <TD Align = "Center" Width = "50%">
            <INPUT Type = "Button" Value = "SUBMIT" onClick = "verify(this.form)">
          </TD><TD Align = "Center" Width = "50%">
            <INPUT Type = "Reset" Value = "RESET"></TD>
          </TR></TABLE>
    </FORM>
  </CENTER></BODY>
</HTML>

```

### Source Code For login.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
##### Print the HTTP content-type header #####
&PrintHeader;
##### A variable declared to store the path to the text file #####
$basedir = "c:/sct/perlproj/cgi-bin/text/data/";
$datafile = "regstr.dat";
##### The ReadParse subroutine is called from Cgi-lib.pl #####
##### library to decode form information #####
&ReadParse;
##### Assigns the input from the HTML form #####
##### to the specified variables #####
$name=$in{'login'};
$password=$in{'passwd'};

```

```

##### The storing of records into the text file is done incase #####
##### there is no internal server error and incase of failure, #####
##### an HTML page informing about the failure is displayed #####
open(FH1,"$basedir/$datafile");
@lines=<FH1>;
close(FH1);
foreach(@lines) {
    chop();
    @arr=(split(/:/));
    if ($arr[1] eq $name) {
        if($arr[3] eq $passwd) {
            print "<HTML><HEAD><SCRIPT>";
            print "function back(form) { history.back(); }";
            print "</SCRIPT></HEAD>";
            print "<BODY Background='/images/grid1.gif text=green>";
            print "<CENTER><IMG Src='/images/reggoodies.gif></CENTER><BR><BR><BR>";
            print "<CENTER><H1>Welcome ".$name."</H1><FORM><BR>";
            print "<A HREF='/html/text/pnpls.html><IMG Src='/images/epp.gif></A><BR><BR>";
            print "<A HREF='/html/text/gestbk.html'><IMG Src='/images/sgb.gif></A><BR><BR>";
            print "<INPUT Type=button Value=Logout onClick=window.location='/html/text'>";
            print "</FORM></CENTER></BODY></HTML>";
            $flag=1;
            exit;
        }
    }
}
if ($flag != 1) {
    print "<HTML><HEAD><SCRIPT>";
    print "function home(form) { history.back(); }";
    print "</SCRIPT></HEAD>";
    print "<BODY Background='/images/grid1.gif text=green>";
    print "<CENTER><IMG Src='/images/reggoodies.gif></CENTER><BR><BR><BR>";
    print "<CENTER><H1>Sorry your login and password does not exist.</H1><BR><BR><BR>";
    print "<IMG Src='/images/shim.gif Width=20 Height=20>";
    print "<INPUT Type=button Value=Back onClick='home(this.form)'></CENTER>";
    print "</FORM></BODY></HTML>";
    exit;
}
print "<HTML><HEAD><SCRIPT>";
print "function home(form) { history.back(); }";
print "</SCRIPT></HEAD>";
print "<BODY Background='/images/grid1.gif text=green>";
print "<CENTER><IMG Src='/images/reggoodies.gif></CENTER><BR><BR><BR>";
print "<CENTER><H1>Currently there is a problem with the server</H1><BR>";
print "<H1>Please resend your message in an hour</H1><BR><BR>";
print "<IMG Src='/images/shim.gif Width=20 Height=20>";
print "<FORM><INPUT Type=button Value=Home onClick='home(this.form)'></FORM>";
print "</CENTER></BODY></HTML>";

```

## PROJECT SOURCE CODE FOR PEN PALS

Source Code For pnpls.html

&lt;HTML&gt;

```

<HEAD><TITLE>SCT's PEN PAL </TITLE>
<SCRIPT Language = "Javascript">
<!-- The function verify() checks whether appropriate -->
<!-- information is filled in all the elements. -->
    function verify() {
        for (i=0; i<=7; i++) {
            if (document.forms[0].elements[i].value == "") {
                alert("Please fill in the " + document.forms[0].elements[i].name + " field");
                document.forms[0].elements[i].focus();
                return (false);
            }
            if(document.forms[0].elements[1].value!="") {
                pass = document.forms[0].elements[1].value.indexOf('@',0);
                pass1 = document.forms[0].elements[1].value.indexOf('.',0);
                if((pass===-1) || (pass1===-1)) {
                    alert("not a valid email address");
                    document.forms[0].elements[1].focus();
                    return (false);
                }
            }
        }
        return(true);
    }
<!-- The function checks whether the date is between (1-31), -->
<!-- whether the month entered as a number is between (1-12) and -->
<!-- whether the year entered is within 1995. -->
    function checkdate() {
        year=document.forms[0].elements[6].value;
        if (year>2004) {
            alert("Enter proper year (till 2004)");
            document.forms[0].elements[6].focus();
            return(false);
        }
        monval=document.forms[0].elements[5].value;
        dtval=document.forms[0].elements[4].value;
        if ((dtval > 31) || (dtval < 1)) {
            alert("Enter proper date");
            document.forms[0].elements[4].focus();
            return(false);
        }
        if (isNaN(monval) != true) {
            if ((monval > 12) || (monval < 1)) {
                alert("Enter proper month");
                document.forms[0].elements[5].focus();
            }
        }
    }

```

```

        return(false);
    }
}
else {
    alert("Enter the month number");
    document.forms[0].elements[5].focus();
    return(false);
}
return (true);
}
}
<!-- The function checklen() checks whether the length-->
<!-- of name and email address does not exceed 30 characters -->
function checklen() {
    for (i=0;i<=1;i++) {
        val=document.forms[0].elements[i].value;
        len=val.length;
        if (len > 30) {
            alert ("Value exceeds 30 characters");
            document.forms[0].elements[i].value="";
            document.forms[0].elements[i].focus();
        }
    }
}
}
<!-- The function takes the user to the previous page -->
function abort(form) {
    history.back(); }
<!-- Sets the focus on the first field when the form is loaded -->
function set(form) {
    document.forms[0].elements[0].focus();}
</SCRIPT></HEAD>
<BODY Background="/images/grid1.gif" onLoad="set(this.form)">
<CENTER><IMG Src="/images/pnpls.gif"></CENTER>
<P><P><CENTER><P>
<FORM Action="/cgi-bin/text/pnpls.pl" Method="POST" onSubmit="return verify()">
    <P>Please enter your name<BR>
    <INPUT Type="text" Name="name" Size="40" onBlur="checklen()" ><BR>
    <P>Please enter your E-Mail address<BR>
    <INPUT Type="Text" Name="emailid" Size="40" onBlur="checklen()"><BR>
    <P>Please indicate your Sex<BR>
    <INPUT Type="Radio" Name="sex" Value="m" checked>Male
    <IMG Src="/images/shim.gif" Height="1" Width="5">
    <INPUT Type="Radio" Name="sex" Value="f" >Female<BR>
    <P>Please enter your DOB as DD;MM;YYYY<BR>
    <INPUT Type="Text" Name="day" Size="2" maxLength="2">
    <IMG Src="/images/shim.gif" Height="1" Width="2">
    <INPUT Type="Text" Name="month" Size="2" maxLength="2">
    <IMG Src="/images/shim.gif" Height="1" Width="2">
    <INPUT Type="Text" Name="year" Size="4" maxLength="4">
    <P><P>Tell us about your hobbies and interests in the 'Text-Box' below<BR>
    We'll keep your Penpal info posted for <B>One Year</B> from today

```



```

<P><TEXTAREA Name="request" Rows="8" Cols="65"
                onFocus="checkdate()"></TEXTAREA>
<P><P>Can we contact you with information about our products or services.<BR>
<INPUT Type="Radio" Name="moreinfo" Value="y"><FONT Color = "Blue">Yes</Font>
<IMG Src="/images/shim.gif" Width=20 Height=10 >
<INPUT Type="Radio" Name="moreinfo" Value="n" Checked="True">
<Font Color = "Red">No. Thanks!</Font>
<P><CENTER>
    <INPUT Type="Submit" Value="Submit" Name="Submit">
    <INPUT Type="Reset" Value="Reset" Name="Submit">
    <INPUT Type="Button" Value="Abort" Name="At" onClick = "abort(this.form)">
</CENTER>
</FORM><P><CENTER><B>Thank You For Stopping By Our Web Site</B></CENTER>
</BODY>
</HTML>

```

### Source Code For pnpls.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
##### Print the HTTP content-type header #####
&PrintHeader;
##### A variable declared to store the path to the text file #####
$basedir = "c:/sct/perlproj/cgi-bin/text/data/";
##### A variable declared to store the text file #####
$datafile = "pnpls.dat";
##### The current date is captured from the system #####
##### using function 'localtime' and necessary processing #####
##### is carried out to extract the date from the function #####
$slsh="/";
$slctim=localtime(time);
($cday,$cmonth,$cdate,$ctime,$cyear)=split(/s+/, $slctim);
%mon=("Jan"=>"01", "Feb"=>"02", "Mar"=>"03", "Apr"=>"04", "May"=>"05", "Jun"=>"06",
      "Jul"=>"07", "Aug"=>"08", "Sep"=>"09", "Oct"=>"10", "Nov"=>"11", "Dec"=>"12");
for $key(keys%mon) {
    if ($cmonth eq $key) {
        $cmon=$mon{$key};    }
}
$cdt=$cdate.$slsh.$cmon.$slsh.$cyear;
##### The ReadParse subroutine is called from Cgi-lib.pl #####
##### library to decode form information #####
&ReadParse;
##### Assigns the input from the HTML form #####
##### to the specified variables #####
$name=${in{'name'}};
$emid=${in{'emailid'}};
$sex=${in{'sex'}};
$day=${in{'day'}};
$month=${in{'month'}};
$year=${in{'year'}};
$dob=$year.$slsh.$month.$slsh.$day;

```

```

$dtls=$in{'request'};
$sinflg=$in{'moreinfo'};
$age=$year - $year;
##### A check for the duplicate values in the text file is done. #####
#### In case of duplicate value or an internal server error #####
##### an error page is rendered and a flag is set to 1 #####
open(FH1,"<$basedir/$datafile");
@lines=<FH1>;
close(FH1);
foreach(@lines) {
    chop();
    @arr=(split(/:/));
    if ($arr[1] eq $name) {
        if($arr[2] eq $emid) {
            print "<HTML><HEAD><SCRIPT>";
            print "function back(form) { history.back(); }";
            print "</SCRIPT></HEAD>";
            print "<BODY Background=/images/grid1.gif text=green>";
            print "<CENTER><IMG Src=/images/pnpls.gif></CENTER><BR><BR><BR>";
            print "<CENTER><H1>The Name and the Email address submitted is already registered as a
                pen pals entry. This is a duplicate entry and hence not accepted</H1>";
            print "<FORM><INPUT Type=button Value=Back onClick='back(this.form)'></FORM>";
            print "</CENTER></BODY></HTML>";
            $flag=1;
        }
    }
}
}
#### Appending the 'pnpls.dat' with the values #####
##### being entered by the user #####
if ($flag!=1) {
    if (open(FH1,">>$basedir/$datafile")) {
        print FH1 "$cdt:$name:$emid:$sex:$dob:$age:$dtls:$sinflg\n";
        print "<HTML><HEAD><SCRIPT>";
        print "function home(form) { history.back(); history.back(); }";
        print "</SCRIPT></HEAD>";
        print "<BODY Background=/images/grid1.gif text=green>";
        print "<CENTER><IMG Src=/images/pnpls.gif></CENTER><BR><BR><BR>";
        print "<FORM Action=/cgi-bin/text/vwchpnpls.pl' Method=Post>";
        print "<CENTER><H1>Thank You !</H1><BR>";
        print "<H1>Your information has been registered !</H1><BR><BR>";
        print "<INPUT Type=Submit Value=View>";
        print "<IMG Src=/images/shim.gif Width=20 Height=20>";
        print "<INPUT Type=button Value=Back onClick='home(this.form)'>";
        print "</CENTER></FORM></BODY></HTML>";
    }
}
else {
    print "<HTML><HEAD><SCRIPT>";
    print "function home(form) { history.back(); history.back(); }";
    print "</SCRIPT></HEAD>";
    print "<BODY Background=/images/grid1.gif text=green>";
    print "<CENTER><IMG Src=/images/pnpls.gif></CENTER><BR><BR><BR>";
}

```

```

print "<CENTER><H1>Currently there is a problem with the server</H1><BR>";
print "<H1>Please resend your message in an hour</H1><BR><BR>";
print "<IMG Src='/images/shim.gif' Width=20 Height=20>";
print "<FORM><INPUT Type=button Value=Home onClick='home(this.form)'></FORM>";
print "</CENTER></BODY></HTML>";
}
}

```

### Source Code For vwchpnpls.pl

```

#!c:/perl/bin/perl.exe
##### The program uses the cgi-lib.pl library #####
require "cgi-lib.pl";
##### Print the HTTP content-type header #####
&PrintHeader;
##### A subroutine defined to display an HTML page #####
sub srtn {
    print << "DISP";
        <HTML>
        <HEAD><TITLE>VIEWED CHOICE</TITLE></HEAD>
        <BODY Background='/images/grid1.gif text=green><CENTER><IMG Src='/images/pnpls.gif">
            <FORM Action="/cgi-bin/text/vwvwnpls.pl" Method="POST" onSubmit="return checked()"
                Name="One">
                <P><BR><BR><H3><I>Select a Penpal of your choice.</I><BR><BR>
                Sex:<INPUT Type=radio Name="sex" Value="m">Male
                <IMG Src='/images/shim.gif' Height=20 Width=20>
                <INPUT Type=radio Name="sex" Value="f" Checked="True">Female<BR><BR>
                Age: <SELECT Name="age">
                    <OPTION Value="0" Selected> 5 - 10
                    <OPTION Value="1" > 11 - 20
                    <OPTION Value="2"> 21 - 30
                    <OPTION Value="3"> 31 - 40
                    <OPTION Value="4">41 - 50
                    <OPTION Value="5">51 - 60
                    <OPTION Value="6"> 61 - 70
                    <OPTION Value="7">71 - 80
                </SELECT><BR><BR><BR>
                <I> Click on <INPUT Type=Submit Name='Submit' Value="Submit"> when done. </I></H3>
            </FORM>
        </CENTER></BODY>
        </HTML>
    DISP
}
##### The sub-routine is called #####
&srtn;

```

### Source Code For vwvwnpls.pl

```

#!c:/perl/bin/perl.exe
##### The program uses the cgi-lib.pl library #####
require "cgi-lib.pl";

```

```
##### Print the HTTP content-type header #####
```

**&PrintHeader;**

```
##### A variable declared to store the path to text file #####
```

```
$basedir = "c:/sct/perlproj/cgi-bin/text/data/";
```

```
##### A variable declared to store the text file #####
```

```
$datafile = "pnpls.dat";
```

```
##### The ReadParse subroutine is called from Cgi-lib.pl #####
```

```
##### library to decode form information #####
```

**&ReadParse;**

```
##### A sub-routine defined to display an HTML page #####
```

```
sub srout {
```

```
    $sex=$in{'sex'};
```

```
    $age=$in{'age'};
```

```
    if ($age == "0") {
```

```
        $range1=5;
```

```
        $range2=10;
```

```
    }
```

```
    elsif ($age == "1") {
```

```
        $range1=11;
```

```
        $range2=20;
```

```
    }
```

```
    elsif ($age == "2") {
```

```
        $range1=21;
```

```
        $range2=30;
```

```
    }
```

```
    elsif ($age == "3") {
```

```
        $range1=31;
```

```
        $range2=40;
```

```
    }
```

```
    elsif ($age == "4") {
```

```
        $range1=41;
```

```
        $range2=50;
```

```
    }
```

```
    elsif ($age == "5") {
```

```
        $range1=51;
```

```
        $range2=60;
```

```
    }
```

```
    elsif ($age == "6") {
```

```
        $range1=61;
```

```
        $range2=70;
```

```
    }
```

```
    elsif ($age == "7") {
```

```
        $range1=71;
```

```
        $range2=80;
```

```
    }
```

```
    print "<HTML><BODY Background=/images/grid1.gif text=green>";
```

```
    print "<CENTER><IMG Src=/images/pnpls.gif><BR><BR>";
```

```
    print "<B>Sex :</B><IMG Src=/images/shim.gif width=10>";
```

```
    if ($sex ne "f") {
```

```
        $dsex = "Male"; }
```

```

else {
    $dsex = "Female"; }
print "$dsex";
print "<IMG Src=/images/shim.gif width=20>";
print "<B>Age Group:</B>";
print "<IMG Src=/images/shim.gif width=10>";
print "$range1 -\ $range2";
print "<BR><BR></CENTER></BODY></HTML>";
##### A page displaying the emailid and penpal information #####
##### of the person is displayed based on the selection #####
##### which is done on the basis of age. #####
##### In case of any errors, an error page is displayed. #####
if (!open(FH1,"$basedir/$datafile")) {
    print "<HTML><HEAD><SCRIPT>";
    print "function home(form) { history.go(-2); }";
    print "</SCRIPT></HEAD>";
    print "<BODY Background=/images/grid1.gif text=green>";
    print "<CENTER><IMG Src=/images/ppinfmaint.gif></CENTER><BR><BR><BR>";
    print "<CENTER><H1>Currently there is a problem with the server</H1><BR>";
    print "<H1>Please resend your message in an hour</H1><BR><BR>";
    print "<IMG Src=/images/shim.gif Width=20 Height=20>";
    print "<FORM><INPUT Type=button Value=Home onClick='home(this.form)'></FORM>";
    print "</CENTER></BODY></HTML>";
}
else {
    @lines=<FH1>;
    close(FH1);
    foreach(@lines) {
        chop();
        @arr=(split(/:/));
        if ($arr[3] eq $sex) {
            if($arr[5] >= $range1 && $arr[5] <= $range2) {
                $ref=$arr[2];
                $semid=$arr[2];
                print "<HTML><HEAD><SCRIPT>";
                print "function back(form) { history.back(); }";
                print "</SCRIPT></HEAD>";
                print "<BODY Background=/images/grid1.gif text=green>";
                print "<CENTER><IMG Src=/images/pnpls.gif></CENTER><BR><BR>";
                print "<TABLE><TR>";
                print "<TD><A Href=mailto:$semid>Reply to : $ref</A></TD>";
                print "</TR></TABLE><TABLE><TR><TD>";
                print "$arr[6]";
                print "</TD></TR></TABLE></BODY></HTML>";
            }
        }
    }
}
}

```

```

print "<HTML><BODY><FORM><CENTER>";
print "<INPUT Type=button Value=Re-Search onClick=window.location='/cgi-
      bin/text/vwchpnpls.pl'>";
print "<INPUT Type=button Value=Logout onClick=window.location='/html/text/index.html'>";
print "</CENTER></FORM></BODY></HTML>";
}
}
##### The sub-routine is called #####
&srout;

```

## PROJECT SOURCE CODE FOR GUEST BOOK

### Source Code For gestbk.html

```

<HTML>
<HEAD><TITLE>SCT'S GUEST BOOK</TITLE>
<SCRIPT Language = "Javascript">
<!-- The function checks whether appropriate information is -->
<!-- filled in all the elements. If any element is left empty, -->
<!-- an alert() box is displayed informing the user to fill -->
<!-- in the empty element. The code also scans the Emailed for -->
<!-- the presence of an '@' and a '.' symbol. -->
  function verify(form) {
    for (i=1; i<=2; i++) {
      if (document.forms[0].elements[i].value == "") {
        alert("Please fill in the " + document.forms[0].elements[i].name + " field");
        document.forms[0].elements[i].focus();
        return (false);
      }
      if (document.forms[0].elements[1].value != "") {
        pass = document.forms[0].elements[1].value.indexOf('@',0);
        pass1 = document.forms[0].elements[1].value.indexOf('.',0);
        if ((pass== -1) || (pass1== -1)) {
          alert("Not a valid Email address");
          document.forms[0].elements[1].focus();
          return (false);
        }
      }
    }
    return(true);
  }
<!-- This function takes the user back to the 'Home Page' -->
<!-- from the current page -->
  function abort(form) {
    history.back(); }
<!-- Sets the focus on the first element when the form is loaded -->
  function set(form) {
    document.forms[0].elements[0].focus(); }

```

```

<!-- The function checklen() checks that the length of name -->
<!-- and email address does not exceed 30 characters. -->
function checklen(form) {
    for (i=0;i<=1;i++) {
        val=document.forms[0].elements[i].value;
        len=val.length;
        if (len > 30) {
            alert ("Value exceeds 30 characters");
            document.forms[0].elements[i].value="";
            document.forms[0].elements[i].focus();
        }
    }
}
</SCRIPT></HEAD>
<BODY Background="/images/grid1.gif" TEXT="green" onLoad="set(this.form)">
  <CENTER><IMG Src="/images/gestbk.gif" Alt="gestbk.gif"></CENTER>
  <P><P><CENTER>Please take a few moments to let us know you were here today.
  <P><P><FORM Action = "/cgi-bin/text/gestbk.pl" Method="POST" onSubmit=" return
    verify(this.form)">
    <P><P>Please Give Us Your Name<BR>
    <INPUT Type="text" Name="name" Size="40" onBlur="checklen(this.form)"><BR>
    <P><P>Please Give Us Your Email Address<BR>
    <INPUT Type="text" Name="emailid" Size="40" onBlur="checklen(this.form)"><BR>
    <P><P>Bouquets Or Brickbats Are Welcome<BR>
    <TEXTAREA Name="request" Rows="8" Cols="65"></TEXTAREA>
    <P><P>Can we contact you with information about our products or services.<BR><BR>
    <INPUT TYPE="radio" NAME="moreinfo" VALUE="y">
    <FONT Color="Blue">Yes</FONT><IMG Src="/images/shim.gif" Width=20 Height=10>
    <INPUT Type="radio" Name="moreinfo" Value="n" Checked="True">
    <FONT Color="Red">No, Thanks!</FONT>
    <P><INPUT Type="Submit" Value="Submit"><INPUT Type="Reset" Value="Reset">
    <INPUT Type="Button" Value="Abort" onClick="abort(this.form)">
  </FORM><P><B>Thank You For Stopping By Our Web Site</B></CENTER>
</BODY>
</HTML>

```

### Source Code For gestbk.pl

```

#!c:/perl/bin/perl.exe
##### The program uses the cgi-lib.pl library #####
require "cgi-lib.pl";
##### Print the HTTP content-type header #####
&PrintHeader;
##### The ReadParse subroutine is called from Cgi-lib.pl #####
##### library to decode form information #####
&ReadParse;
##### A variable declared to store the path to the text file #####
$basedir = "c:/sct/perlproj/cgi-bin/text/data/";
##### A variable declared to store the text file #####
$datafile = "gestbk.dat";

```

```

##### The current date is captured from the system #####
##### using function 'localtime' and necessary processing #####
##### is carried out to extract the date from the function #####
$slsh="/";
$ltim=localtime(time);
($cdy,$cmnth,$cdte,$ctim,$cyr)=split(/\s+/, $ltim);
%mon=("Jan"=>"01", "Feb"=>"02", "Mar"=>"03", "Apr"=>"04", "May"=>"05", "Jun"=>"06",
      "Jul"=>"07", "Aug"=>"08", "Sep"=>"09", "Oct"=>"10", "Nov"=>"11", "Dec"=>"12");
for $key(keys%mon) {
    if ($cmnth eq $key) {
        $cmn=$mon{$key};
    }
}
$cdt=$cyr.$slsh.$cmn.$slsh.$cdte;
##### Assigns the input from the HTML form to the variables #####
$name=$in{'name'};
$emid=$in{'emailid'};
$infg=$in{'moreinfo'};
$dttl=$in{'request'};
#### Appending the 'gestBk.txt' through the filehandle #####
#### In case of any errors, an error page is displayed #####
if (open(FHD,">>$basedir/$datafile")) {
    print FHD "$cdt:$name:$emid:$dttl:$infg\n";
    print "<HTML><HEAD><SCRIPT>";
    print "function home(form) { history.go(-2); }";
    print "</SCRIPT></HEAD>";
    print "<BODY Background=/images/grid1.gif text=green>";
    print "<CENTER><IMG Src=/images/gestbk.gif></CENTER><BR><BR><BR>";
    print "<CENTER><H1>Thank You !</H1><BR>";
    print "<H1>Your information has been registered !</H1><BR><BR>";
    print "<IMG Src=/images/shim.gif Width=20 Height=20>";
    print "<FORM><INPUT Type=button Value=Home onClick='home(this.form)'></FORM>";
    print "</CENTER></BODY></HTML>";
}
else {
    print "<HTML><HEAD><SCRIPT>";
    print "function home(form) { history.go(-2); }";
    print "</SCRIPT></HEAD>";
    print "<BODY Background=/images/grid1.gif text=green>";
    print "<CENTER><IMG Src=/images/gestbk.gif></CENTER><BR><BR><BR>";
    print "<CENTER><H1>Currently there is a problem with the server</H1><BR>";
    print "<H1>Please resend your message in an hour</H1><BR><BR>";
    print "<IMG Src=/images/shim.gif Width=20 Height=20></FORM>";
    print "<FORM><INPUT Type=button Value=Home onClick='home(this.form)'>";
    print "</CENTER></BODY></HTML>";
}
}

```



# D. PROJECTS IN PERL USING A DATABASE

## OBJECTIVE

To develop a system, which keeps track of all Problem Areas, Errors and Techniques identified and implemented during the course of software development.

Information stored in the system should be freely available to every programmer when required.

Additionally new techniques encountered by programmers, which may be useful to other programmers, need to be specified and accessed when required.

The optimum method of achieving the above is by implementing a web solution using Oracle as a data storage system.

## Modules

The core modules of this system will be as follows:

1. Managing Users / Employees (Add / Edit / Delete / View)
2. Data Storage of identified Problem Areas and Errors
3. Data Storage of probable Solutions to Problem Areas and Errors
4. Data Storage of Tips & Tricks
5. Data Retrieval



Diagram D.1

Using these, employees can post their queries/problems on the Intranet. All other employees can access these, and any employee can post appropriate solutions on the Intranet, if they know the Solution. Besides, if an employee comes across some information that might prove helpful to other employees, they can post it on the Intranet as a Tip, where all other employees can access it.

## System Study

The user first needs to log in to the system. For this, the user's identity (in terms of Login-Id and Password) needs to be accepted and verified against the Information Base. To implement this, an interface is created which includes a login screen with:

- Two text fields to accept the Login-Id and Password
- One SUBMIT Button

After the user enters the Login-Id and Password, and clicks on SUBMIT button, the user is identified against the Information Base. If recognized to be a Valid User, the user is led to a Welcome Screen (home page), with five options:

- Problems and Solutions
- Tips and Tricks
- Employee Information
- Home
- Logout

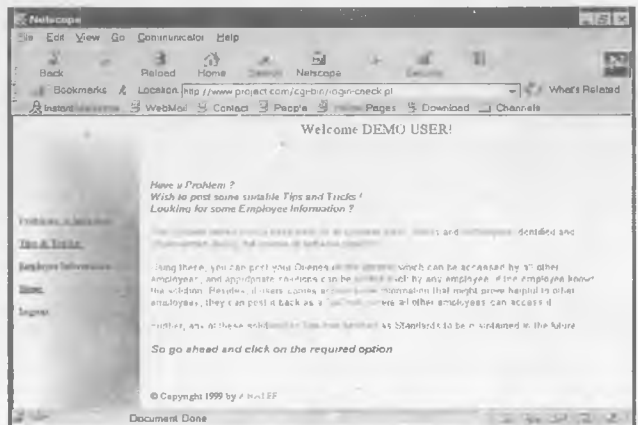


Diagram D.2

The five options on the left are Hyperlinks, and will lead the User to respective areas of information. The options are explained in detail below.

## Problems And Solutions

This is the main area of usage and deals with the following:

- Posting a Query
- Replying to a Posted Query i.e. Posting a Solution
- Viewing Problems and its related Solutions

Choosing different options under Problems & Solutions displays suitable interfaces.

### Add Problem

The interface includes:

- The name (first name and last name) of the user posting the problem, which is automatically displayed.
- The date and time when the problem is being posted also displayed automatically.
- A text box provided for the title of the problem.
- A text area for the description of the problem.
- Two buttons, SUBMIT and RESET.

Upon entering the Problem Title and its description in the text area and clicking on the SUBMIT button, the data gets stored in the database.

### Add Solutions

The Interface will display:

- A list of all the Posted Queries (*Problem Title and Problem Description*).

Clicking on a Problem Title shows an interface, which includes,

- Problem Title automatically displayed.
- Employee Name who has posted the problem automatically displayed.
- Date and time of when the problem was posted automatically displayed.
- Problem description automatically displayed.
- A text area is provided for the solution to be entered by the user.

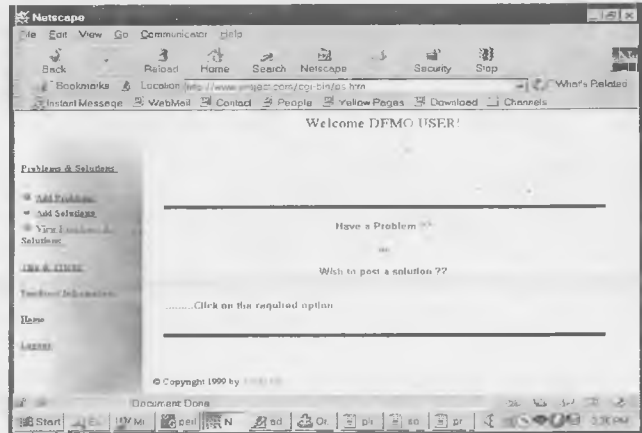


Diagram D.3

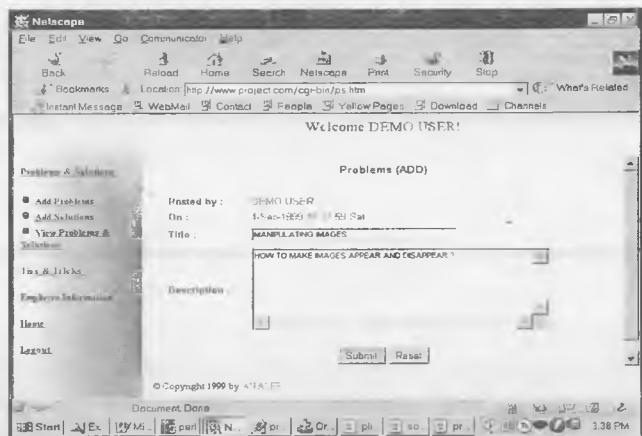


Diagram D.4

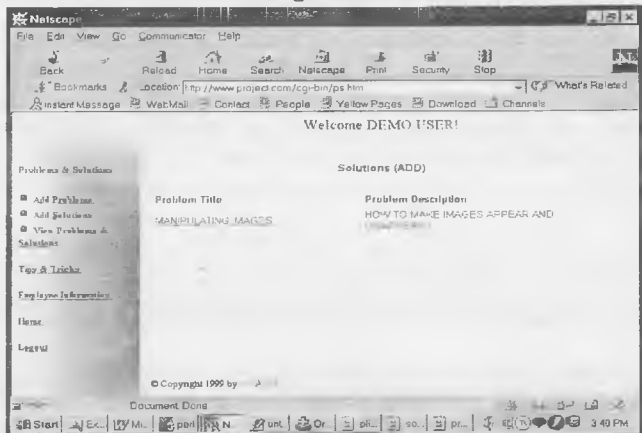


Diagram D.5

- ❑ The name of the user posting the solution automatically displayed.
- ❑ Date and time of solution posted is also automatically displayed.
- ❑ Two buttons, SUBMIT and RESET.

Upon entering the solution in the textarea and clicking on the SUBMIT button, the solution gets stored in the database.

**View Problems And Solutions**

Clicking on View Problems & Solution provides an interface, which includes

- ❑ A list of Problem Titles and their description.

Clicking on the Problem Title takes the user to a more detailed interface.

In this interface:

- ❑ Problem Title automatically displayed.
- ❑ Employee Name who has posted the problem automatically displayed.
- ❑ Date and time of when the problem was posted automatically displayed.
- ❑ Problem description automatically displayed.
- ❑ Solution/s for the problem is displayed.

Clicking on any of the available solutions displays:

- ❑ Name of the user who has posted the solution.
- ❑ Date and time of when the solution was posted
- ❑ Solution.

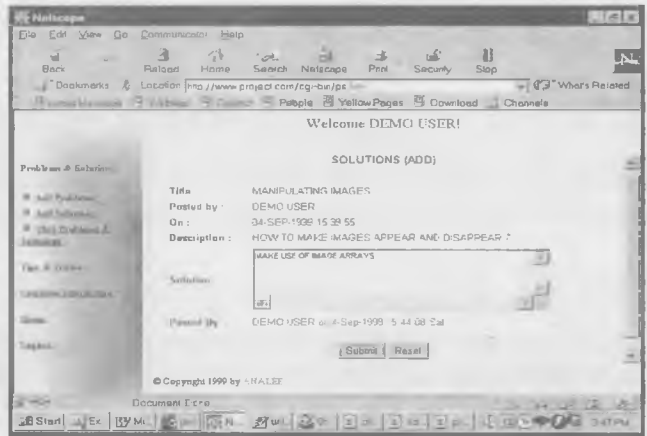


Diagram D.6.

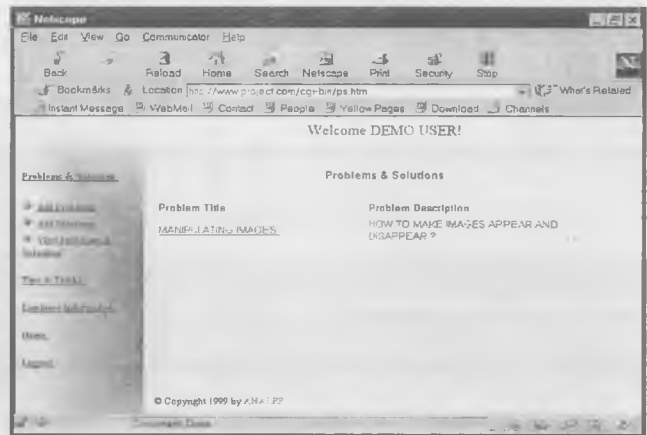


Diagram D.7



Diagram D.8



Diagram D.9

### Tips And Tricks

This module deals with the following:

- Posting a Tip
- Viewing Tips and Tricks posted.

The options provided are explained in detail below.

### Add Tips

Clicking on this link shows an interface, which displays:

- Name of the user who is posting the Tip.
- Date and time when the Tip is being posted.
- A text box for the Title of the Tip
- A TextArea for the description of the Tip.
- A SUBMIT and RESET button.

Upon entering the Tip Title and its description and clicking on SUBMIT button, the data gets stored in the database.

### View Tips

Clicking on this link shows a List of all the Tips and Tricks posted.

Clicking on the title of the tip takes the user to a more detailed interface, which includes:

- Title of the Tip
- Employee who posted the Tip
- Date and Time when the Tip was posted
- Tip Description

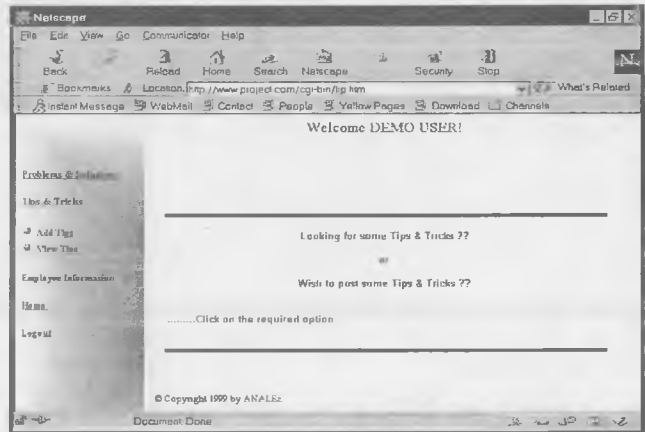


Diagram D.10

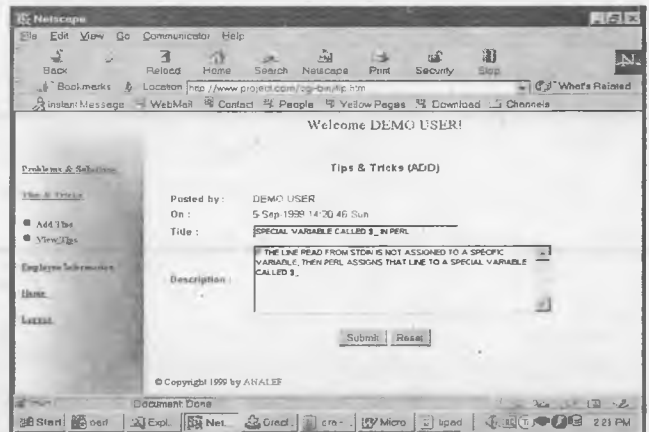


Diagram D.11

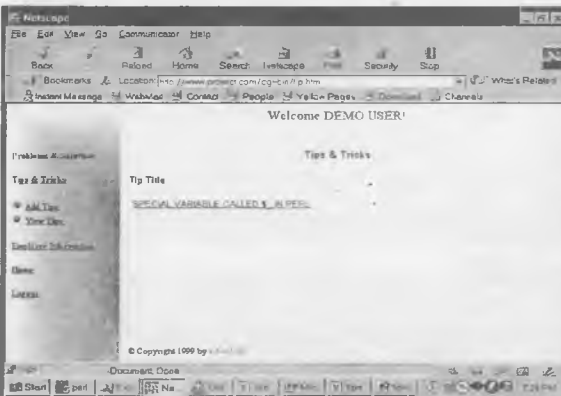


Diagram D.12

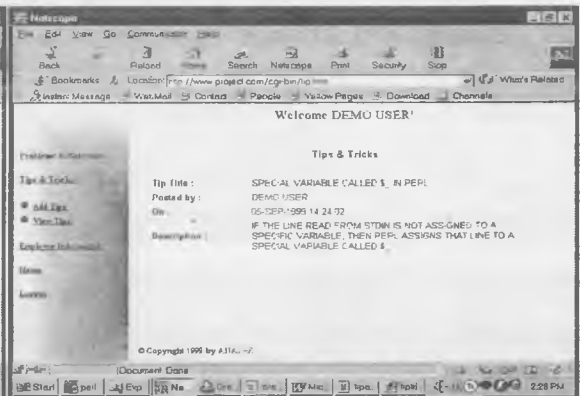


Diagram D.13

## Employee Information

This module deals with the following:

- Adding Employee Details
- Editing Employee Details
- Deleting Employee Details
- Viewing Employee Details

The options provided are explained in detail below.

### Add

Clicking on this link displays an interface, which includes:

- Employee Number that is generated and displayed automatically.
- Employee Name (First and Last Name)
- Designation
- Joining Date
- Confirmation Date
- Login Id
- Password
- Two buttons, SUBMIT and RESET.

Upon filling the form and clicking on the SUBMIT button, employee information gets stored in the database.

### Edit

Clicking on this link provides an interface with:

- A list of all employees displayed in a drop down list box
- Two buttons, DETAILS and RESET.

Selecting a user from the Drop down list and clicking on DETAILS button provides the user with an interface, which gives all details about the employee. This interface includes

- Employee No
- Employee Name (First and Last Name)
- Designation
- Joining Date
- Confirmation Date
- Login Id
- Password

This interface also provides a facility to change the password with additional fields:

- Old password
- New password

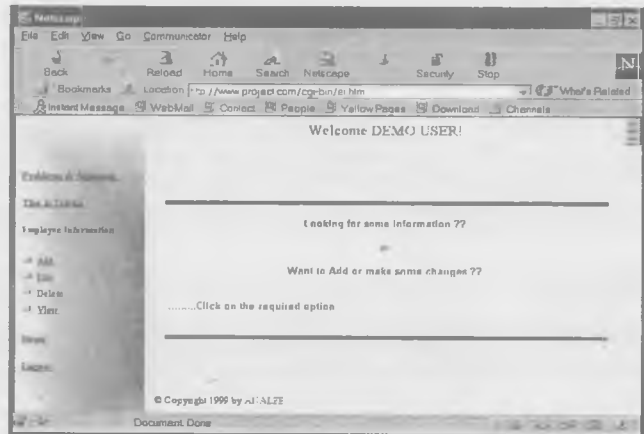


Diagram D.14

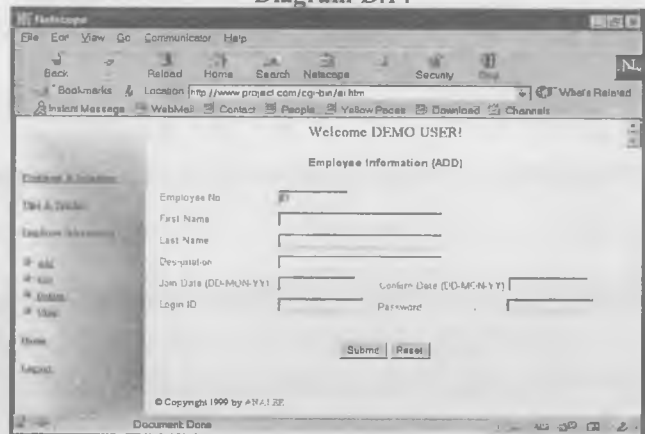


Diagram D.15



Diagram D.16

Two buttons SUBMIT and RESET.

Making required changes and clicking on SUBMIT button updates the employee information.

### Delete And View

Clicking on these links provides the user with a similar interface as with the Edit option, a list of all employees in a drop down list with two buttons Details and Reset for View and a Delete button for Delete

- ❑ The Password Field, when entered, should be displayed as *Asterisks*. There should also be a Confirm Password Field (also displayed as *Asterisks*), and the values of these two fields should match when the user tries to save the record.

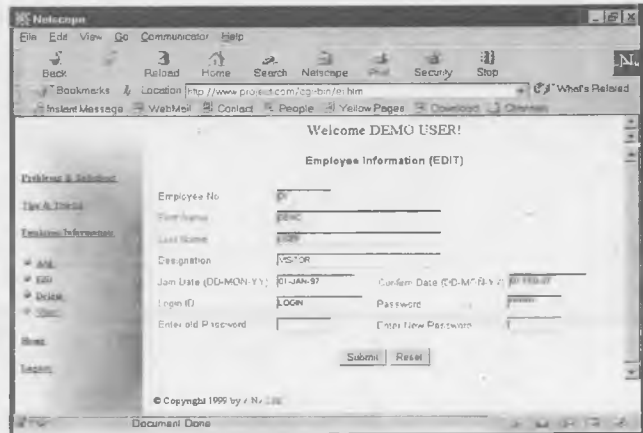


Diagram D.17

### Validations To Be Considered

1. When the user logs in for the first time and clicks on the Submit Button, his validity as a User (Login Id and Password) needs to be verified and only then he should be able to proceed further.
2. On every page that the user traverses, the first name and the last name of the user needs to be displayed on top of the page as a header. **Hint:** Use Cookies.
3. In the Add Problem Interface,
  - ❑ Name (First name and last name) of the user posting a query must be displayed automatically.
  - ❑ Date and Time of posting the Query must also be displayed automatically.
  - ❑ The Problem Title and Problem Description should not be left blank.
  - ❑ As soon as a User Posts (Submits) a Query, the Screen displaying a list of all the queries must be updated to include the most recent Query.
4. In the Add Solution Interface,
  - ❑ Problem Title must be displayed automatically.
  - ❑ Employee Name who has posted the problem must be displayed automatically.
  - ❑ Date and time of when the problem was posted must be displayed automatically.
  - ❑ Problem description must be displayed automatically.
  - ❑ The name of the user posting the solution must be displayed automatically.
  - ❑ Date and time of solution posted must be displayed automatically.
  - ❑ Solution Description should not be left blank.
  - ❑ As soon as a User Posts (Submits) a Solution, the Screen displaying a list of all the Solutions must be updated to include the most recent Solution.
5. In the View Problems & Solution Interface,
  - ❑ Problem Title must be displayed automatically.
  - ❑ Employee Name who has posted the problem must be displayed automatically.
  - ❑ Date and time of when the problem was posted must be displayed automatically.
  - ❑ Problem description must be displayed automatically.
  - ❑ List of all solutions related to that problem must be displayed.
  - ❑ Once the user clicks on any one solution listed, the next interface should display automatically Name of the employee who posted the solution along with Date and Time of Solution posted and its description.
6. In the Add Tip Interface,
  - ❑ Employee Name who is posting the Tip must be displayed automatically.
  - ❑ Date and time of when the Tip is posted must be displayed automatically.
  - ❑ Tip Title and Tip Description should not be left blank.

- As soon as a User Posts (Submits) a Tip, the Screen displaying a list of all the Tips must be updated to include the most recent Tip.
- 7. In the View Tip Interface,
  - List of all Tips and Tricks must be displayed.
  - Once the user clicks on any Tip Title listed, the next interface should display automatically Name of the employee who posted the Tip along with Date and Time of Tip posted and its description.
- 8. In the Add Employee Interface,
  - Employee Number must be generated automatically and the user should not be allowed to modify the same.
  - None of the fields should be left blank.
  - Date of Joining and Date of Confirmation should be in specified format only i.e. DD-MON-YY
  - The Password Field, when entered, should be displayed as Asterisks.
  - After the user Submits, an alert has to be displayed with a message that the information has been added successfully.
- 9. In the Edit Employee Interface,
  - User should not be allowed to modify the Employee Number.
  - None of the fields should be left blank.
  - Date of Joining and Date of Confirmation should be in specified format only i.e. DD-MON-YY
  - The Password Field, when entered, should be displayed as Asterisks.
  - The Old Password and New Password Field, when entered should also be displayed as Asterisks.
  - After the user Submits, an alert has to be displayed with a message that the information has been updated successfully.
- 10. In the Delete Employee Interface,
  - User should not be allowed to modify any of the fields.
  - The Password Field should be displayed as Asterisks.
  - After the user Submits, an alert has to be displayed with a message that the information has been deleted successfully.
- 11. In the View Employee Interface,
  - User should not be allowed to modify any of the fields.
  - The Password Field should be displayed as Asterisks.

## TABLE STRUCTURES

**Table Name** : Employee\_Info

**Column Definition:**

Name	Datatype/Size	Remarks
Employee No	varchar2(8)	Primary Key
First Name	varchar2(60)	
Last Name	varchar2(60)	
Designation	varchar2(60)	
Date of Joining	date	
Date of Confirmation	date	
Login Id	varchar2(20)	Combination has to be Unique
Password	varchar2(10)	

Insert a record as shown below, to login successfully from the login screen:

```
INSERT INTO employee_info
```

```
VALUES ('D1', 'DEMO', 'USER', 'VISITOR', '01-JAN-97', '01-FEB-97', 'LOGIN', 'PASSWORD');
```

**Table Name** : LU\_ENO

**Column Definition:**

Name	Datatype/Size	Remarks
Employee No	varchar2(8)	Holds a unique number of a new employee

Insert a record as shown below:

INSERT INTO LU\_ENO VALUES ('E2');

**Table Name** : Problem\_Definition

This table stores not only the solutions to the Queries, but also Tips and Tricks

**Column Definition:**

Name	Datatype/Size	Remarks
Problem_Date_Time	date	Primary Key (Automatically Populated, taking the Date and Time from the Server, at the time when the user saves the Query.)
Employee_No	varchar2(8)	Foreign Key (Automatically Populated, depending on the Current User's Login Id and Password)
Problem Title	varchar2(60)	
Problem Description	varchar2(2000)	
Problem Status	varchar2(6)	

**Table Name** : Probable\_Solutions

**Column Definition:**

Name	Datatype/Size	Remarks
Solution_Date_Time	date	Primary Key (Automatically Populated, taking the Date and Time from the Server, at the time when the user saves the Solution)
Problem_Date_Time	date	Foreign Key to Problem_Definition (Automatically Populated, depending on the Query that the user is trying to provide the Solution to)
Employee_No	varchar2(8)	Foreign Key to Employee_Info (Automatically Populated, depending on the Current User's Login Id and Password)
Solution_Title	varchar2(60)	Populated automatically with the Problem Title, if the current record is a solution to an existing Query. Null, if the current record is a Tip or a Trick.
Solution Description	varchar2(2000)	

The SQL queries for the above tables are available within the book CD-ROM and are located at:  
ChapterWiseCode&Projects\Part 4 CGI\Project\Tips Tricks\SQL\TipsNTricks.txt

## STARTING THE TIPS AND TRICKS SYSTEM

To access the PERL based web site on Tips and Tricks, a name based, virtual host entry is created under Apache2, pointing to a directory. This is followed by an entry in the **hosts** (c:\windows\system32\drivers\etc\hosts) file as follows:

172.16.9.66                      sct.tipsntricks.com                      *(IP Address may differ on individual machine bases)*

The following entries are made in the httpd.conf file as follows:

```
NameVirtualHost 172.16.9.66
<VirtualHost 172.16.9.66>
    ServerAdmin webmaster@sct.com
    DocumentRoot c:\sct\tipsntricks
    ServerName sct.tipsntricks.com
    ScriptAlias /cgi-bin/ "c:\sct\tipsntricks\cgi-bin"
</VirtualHost>
```

### Note



The default ScriptAlias tag of Apache2 should be disabled by commenting the line in the httpd.conf. For example: #ScriptAlias /cgi-bin/ "C:/Program Files/Apache Group/Apache2/cgi-bin/"

To start a session of the Tips And Tricks system, follow these steps:

- Open a browser
- Type in the url:

**http://sct.tipsntricks.com/**





```

    &LoginPage;
    close($Db);
    exit;
}
}
print "Content-type: text/html\n\n";
print "<HTML><BODY><SCRIPT Language=JavaScript>";
print "alert('ACCESS DENIED'); history.back()";
print "</SCRIPT></BODY></HTML>";
close($Db);
exit;
sub LoginPage {
    print "<HTML><FRAMESET Cols=21%,* FrameSpacing=NO>";
    print "<FRAME BgColor=YELLOW FrameBorder=NO Name=list Src=../menu.htm>";
    print "<FRAMESET Rows=10%,80%,* FrameSpacing=NO>";
    print "<FRAME FrameBorder=NO Name=head Src=header.pl>";
    print "<FRAME FrameBorder=NO Name=show Src=../display.htm>";
    print "<FRAME FrameBorder=NO Name=foot Src=../footer.htm>";
    print "</FRAMESET></FRAMESET></HTML>";
}

```

### Source Code For cgi-lib.pl

```

#!/perl/bin/perl.exe
# ReadParse - Reads in GET or POST data, converts it to unescaped text,
# and puts one key=value in each member of the list @in. Also creates
# key/value pairs in %in, using '\0' to separate multiple selections.
# Returns TRUE if there was input, FALSE if there was no input.
# UNDEF may be used in the future to indicate some failure.
# Now that cgi scripts can be put in the normal file space,
# it is useful to combine both the form and the script in one place.
# If no parameters are given (i.e., ReadParse returns FALSE),
# then a form could be output.
# If a variable-glob parameter (e.g., *cgi_input) is passed
# to ReadParse, information is stored there,
# rather than in $in, @in, and %in.
sub ReadParse {
    local (*in) = @_ if @_;
    local ($i, $key, $val);
# Read in text
    if (&MethGet) { $in = $ENV{'QUERY_STRING'}; }
    elsif (&MethPost) { read(STDIN,$in,$ENV{'CONTENT_LENGTH'}); }
    @in = split(/&|;/,$in);
    foreach $i (0 .. $#in) {
# Convert plus's to spaces
        $in[$i] =~ s/\+//g;
# Split into key and value.
        ($key, $val) = split(/=/,$in[$i],2); # splits on the first =.
# Convert %XX from hex numbers to alphanumeric
        $key =~ s/%(..)/pack("c",hex($1))/ge;
        $val =~ s/%(..)/pack("c",hex($1))/ge;
    }
}

```

```

# Associate key and value
  in{$key} .= "\0" if (defined($in{$key})); \0 is the multiple separator
  in{$key} .= $val;
}
return scalar(@in);
}

# PrintHeader - Returns the a line which tells WWW that the information
# to follow should be treated as an HTML document.
sub PrintHeader {
  print "Content-type: text/html\n\n";
  return 1;
}

# MethGet - Return true if this cgi call was using the GET request,
# false otherwise
sub MethGet {
  return ($ENV{'REQUEST_METHOD'} eq "GET");
}

# MethPost - Return true if this cgi call was using the POST request,
# false otherwise
sub MethPost {
  return ($ENV{'REQUEST_METHOD'} eq "POST");
}

# FormEncoded - Return true if this CGI call was having CONTENT_TYPE
# "applicatio....."
sub FormEncoded {
  return ($ENV{'CONTENT_TYPE'} eq "application/x-www-form-urlencoded");
}

# MyURL - Returns a URL to the script
sub MyURL {
  local ($port);
  $port = ":" . $ENV{'SERVER_PORT'} if $ENV{'SERVER_PORT'} != 80;
  return 'http://' . $ENV{'SERVER_NAME'} . $port . $ENV{'SCRIPT_NAME'};
}

# CgiError - Prints out an error message,
# which contains appropriate headers, markup, etcetera.
# Parameters: If no parameters, gives a generic error message.
# Otherwise, the first parameter will be the title
# and the rest will be given as different paragraphs of the body
sub CgiError {
  local (@msg) = @_;
  local ($i,$name);
  if (!@msg) {
    $name = &MyURL;
    @msg = ("Error: script $name encountered fatal error");
  };
  print &PrintHeader;
  print "<HTML><HEAD><TITLE>$msg[0]</TITLE></HEAD>\n";
}

```

```

print "<BODY><H1>$msg|0|</H1>\n";
foreach $i (1 .. $#msg) { print "<P>$msg|$i|</P>\n"; }
print "</BODY></HTML>\n";
}

# CgiDie - Identical to CgiError, but also quits
# with the passed error message.
sub CgiDie {
    local (@msg) = @_ ;
    &CgiError (@msg);
    die @msg;
}

# PrintVariables - Nicely formats variables in an associative array
# passed as a parameter and returns the HTML string.
sub PrintVariables {
    local (%in) = @_ ;
    local ($old, $out, $output);
    $old = $*;
    $* = 1;
    $output .= "\n<DL Compact>\n";
    foreach $key (sort keys(%in)) {
        foreach (split("\0", $in{$key})) {
            ($out = $_) =~ s/\n/<BR>\n/g;
            $output .= "<DT><B>$key</B>\n <DD><I>$out</I><BR>\n";
        }
    }
    $output .= "</DL>\n";
    $* = $old;
    return $output;
}

# PrintVariablesShort - Now obsolete; just calls PrintVariables
sub PrintVariablesShort {
    return &PrintVariables(@_);
}

1; #return true

```

### Source Code For menu.htm

```

<HTML><HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
<BODY Background=fade_bg.jpg LeftMargin=0 RightMargin=0><BR><BR><BR><BR>
<A HRef=ps.htm Target=_top><FONT Color=#23238e Face=Times New Roman Size=1>
<STRONG>Problems & Solutions</STRONG></FONT></A><BR><BR>
<A HRef=tip.htm Target=_top><FONT Color=#23238e Face=Times New Roman Size=1>
<STRONG>Tips & Tricks</STRONG></FONT></A><BR><BR>
<A HRef=ei.htm Target=_top><FONT Color=#23238e Face=Times New Roman Size=1>
<STRONG>Employee Information</STRONG></FONT></A><BR><BR>
<A HRef=display.htm Target="show"><FONT Color=#23238e Face=Times New Roman
Size=1><STRONG>Home</STRONG></FONT></A><BR><BR>
<A HRef=index.htm Target= top><FONT Color=#23238e Face=Times New Roman Size=1>
<STRONG>Logout</STRONG></FONT></A>
</BODY></HTML>

```

Source Code For style.css

```
body { font-family: Arial; color: 23238E; font-size: 8pt}
h4 { font-family: Arial; color: 23238e; font-size: 9pt}
h5 { font-family: Arial; color: 000000; font-size: 6pt}
vlink { color: rgb(153,102,0)}
alink { color: rgb(255,0,0)}
```

Source Code For header.pl

```
#!/c:/perl/bin/perl.exe
require "cgi-lib.pl";
&ReadParse;
&PrintHeader;
$cookie1 = $ENV{'HTTP_COOKIE'};
($fname, $lname, $login, $pswd) = split(/;/, $cookie1);
($fname1, $fname2) = split(/=/, $fname);
($lname1, $lname2) = split(/=/, $lname);
($login1, $login2) = split(/=/, $login);
($pswd1, $pswd2) = split(/=/, $pswd);
print "<HTML><BODY><CENTER><FONT Color=ff0000>Welcome $fname2 $lname2!</FONT>";
print "</CENTER></BODY></HTML>";
```

Source Code For display.htm

```
<HTML>
<HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
<BODY><BR>
<FONT Face=Arial Size=3><B><I>Have a Problem?<BR>Wish to post some suitable Tips and
Tricks !<BR>Looking for some Employee Information?</B></I></FONT><BR>
<MULTICOL Cols=1 Width=150>This system allows you to keep track of all problem areas,
errors and techniques identified and implemented during the course of software
projects.<P>Using these, you can post your Queries on the Intranet which can be accessed
by all other employees, and appropriate solutions can be posted back by any employee, if
the employee knows the solution. Besides, if users comes across some information that
might prove helpful to other employees, they can post it back as a Tip/Trick, where all
other employees can access it.<P>Further, any of these solutions or Tips can be used as
Standards to be maintained in the future.
<P><H4><I>So go ahead and click on the required option</I></H4>
</MULTICOL>
</BODY>
</HTML>
```

Source Code For footer.htm

```
<HTML><BODY>
<FONT Color=#2266FF>@ Copyright 2004 by</FONT>
<FONT Color=#FF6600><U><I>Silicon Chip Technologies</I></U></FONT>
</BODY></HTML>
```

Source Code For ps.htm

```

<HTML>
  <FRAMESET Cols="21%,*" FrameSpacing=NO>
    <FRAME BgColor=Yellow FrameBorder=NO Name="list" Src="probsol.htm">
    <FRAMESET FrameSpacing=NO Rows="10%,80%,*">
      <FRAME FrameBorder=NO Name="head" Src="/cgi-bin/header.pl">
      <FRAME FrameBorder=NO Name="show" Src="psabout.htm">
      <FRAME FrameBorder=NO Name="foot" Src="footer.htm">
    </FRAMESET>
  </FRAMESET>
</HTML>

```

Source Code For probsol.htm

```

<HTML>
  <HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
  <BODY Background=Image/fade_bg.jpg LeftMargin=0 RightMargin=0><BR><BR><BR><BR>
    <A HRef=home.htm Target=_top><FONT Color=#23238E Face=Times New Roman
      Size=1><STRONG>Problems & Solutions</STRONG></FONT></A><BR><BR>
    <IMG Src=Images/tball.gif>
    <A AccessKey="P" HRef=/cgi-bin/probadd.pl Target="show"><FONT Color=#23238E
      Face=Times New Roman Size=1><STRONG>Add
      Problems</STRONG></FONT></A><BR>
    <IMG Src=Images/tball.gif>
    <A AccessKey="S" HRef=/cgi-bin/plist_add.pl Target="show"><FONT Color=#23238E
      Face=Times New Roman Size=1><STRONG>Add
      Solutions</STRONG></FONT></A><BR>
    <IMG Src=Images/tball.gif>
    <A AccessKey="V" HRef=/cgi-bin/plist_view.pl Target="show"><FONT Color=#23238E
      Face=Times New Roman Size=1><STRONG>View Problems &
      Solutions</STRONG></FONT></A><BR><BR>
    <A HRef=tip.htm Target=_top><FONT Color=#23238E Face=Times New Roman
      Size=1><STRONG>Tips & Tricks</STRONG></FONT></A><BR><BR>
    <A HRef=ei.htm Target=_top><FONT Color=#23238E Face=Times New Roman
      Size=1><STRONG>Employee Information</STRONG></FONT></A><BR><BR>
    <A HRef=home.htm Target=_top><FONT Color=#23238E Face=Times New Roman
      Size=1><STRONG>Home</STRONG></FONT></A><BR><BR>
    <A HRef=index.htm Target=_top><FONT Color=#23238E Face=Times New Roman
      Size=1><STRONG>Logout</STRONG></FONT></A>
  </BODY>
</HTML>

```

Source Code For home.htm

```

<HTML>
  <FRAMESET Cols="21%,*" FrameSpacing=NO >
    <FRAME BgColor=Yellow FrameBorder=NO Name="list" Src="menu.htm">
    <FRAMESET FrameSpacing=NO Rows="10%,80%,*">
      <FRAME FrameBorder=NO Name="head" Src="/cgi-bin/header.pl">
      <FRAME FrameBorder=NO Name="show" Src="display.htm">
    </FRAMESET>
  </FRAMESET>

```

```

        <FRAME FrameBorder=NO Name="foot" Src="footer.htm">
    </FRAMESET>
</FRAMESET>
</HTML>

```

### Source Code For psabout.htm

```

<HTML>
  <HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
  <BODY><BR> <BR>
    <IMG Src=Images/blu-line.gif Width=100%>
    <FONT Size=2><B><CENTER>Have a Problem ??<BR><BR>or<BR><BR>
    Wish to post a solution ??<BR> <BR> <BR></CENTER>
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&.....Click on the required option</B></FONT>
    <IMG Src=Images/blu-line.gif Width=100%>
  </BODY>
</HTML>

```

### Source Code For probadd.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;
&PrintHeader;
$cookie1 = $ENV{'HTTP_COOKIE'};
($fname, $lname, $login, $pswd) = split(/;/, $cookie1);
($fname1, $fname2) = split(/=/, $fname);
($lname1, $lname2) = split(/=/, $lname);
($login1, $login2) = split(/=/, $login);
($pswd1, $pswd2) = split(/=/, $pswd);
$date=localtime(time);
@dt = split(" ", $date);
$date1 = $dt[2] . "-" . $dt[1] . "-" . $dt[4] . " " . $dt[3] . " " . $dt[0];
use Win32::ODBC;
if (!$db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
  $a = Win32::ODBC::Error();
  print "<HTML><BODY><P> $a </P></BODY></HTML>";
  exit;
}
print <<"HD";
<HTML><HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css" ></HEAD>
  <BODY><FORM Action=/cgi-bin/probadd_submit.pl Method=GET>
    <CENTER><H4>Problems (ADD)</H4></CENTER>
    <TABLE Align=Center Width=95%><TR>
      <TD Align=Left Width=20%><FONT Color=#23238E Face=Arial Size=1><B>Posted
      by : </B></FONT></TD>
      <TD Align=Left><FONT Color=#23238E Face=Arial Size=1> $fname2 $lname2
      </FONT></TD>
    </TR></TR>

```

```

<TD Align=Left Width=20%><FONT Color=#23238E Face=Arial Size=1><B>On :
  </B></FONT></TD>
<TD Align=Left><FONT Color=#23238E Face=Arial Size=1> $date1 </FONT></TD>
</TR><TR>
<TD Align=Left Width=20%><FONT Color=#23238E Face=Arial Size=1><B>Title :
  </B></FONT></TD>
<TD Align=Left><FONT Color=#23238E Face=Arial Size=1><INPUT Name=title
  MaxLenght=60 onBlur="this.value=this.value.toUpperCase();" Size=25
  Type=Text></FONT></TD>
</TR><TR>
<TD Align=Left Width=20%><FONT Color=#23238E Face=Arial
  Size=1><B>Description : </B></FONT></TD>
<TD Align=Left><FONT Color=#23238E Face=Arial Size=1><TEXTAREA Cols=35
  Name=prob_description onBlur="this.value=this.value.toUpperCase();" Rows=6>
  </TEXTAREA></FONT></TD>
</TR></TABLE><BR>
<CENTER><INPUT Name="save" Type=Submit Value="Submit">
  <INPUT Name="cancel" Type=Reset Value="Reset"></CENTER>
</FORM><BODY>
</HTML>

```

```
HD
```

```
close($Db);
exit;
```

### Source Code For mylib.pl

```

# file : mylib.pl
require "cgi-lib.pl";
sub month {
  my($a) = @_ ;
  if ($a eq "01") { $a = "JAN"; }
  elsif ($a eq "02") { $a = "FEB"; }
  elsif ($a eq "03") { $a = "MAR"; }
  elsif ($a eq "04") { $a = "APR"; }
  elsif ($a eq "05") { $a = "MAY"; }
  elsif ($a eq "06") { $a = "JUN"; }
  elsif ($a eq "07") { $a = "JUL"; }
  elsif ($a eq "08") { $a = "AUG"; }
  elsif ($a eq "09") { $a = "SEP"; }
  elsif ($a eq "10") { $a = "OCT"; }
  elsif ($a eq "11") { $a = "NOV"; }
  elsif ($a eq "12") { $a = "DEC"; }
  return($a);
}

sub encrypt {
  my($pw) = @_ ;
  my($pcount) = length($pw);
  my($pass);
  for ($i=1; $i <= $pcount; $i++) { $pass = $pass . "." . substr($pw, $i-1, 1); }
  $pass = substr($pass, 1);
}

```



```

my(@pwd) = split(",", $pass);
my(@rev_pwd) = reverse(@pwd);
my($sep);
for ($i=0; $i<$pcount; $i++) { $sep = $sep . ord($rev_pwd[$i]) - 7 . ord(","); }
$sep = $sep/8;
return($sep);
}

sub decrypt {
my($sepw) = @_;
$sepw = $sepw * 8;
@spw = split("44", $sepw);
$len = @spw;
@rpw = reverse(@spw);
for ($i = 0; $i < $len; $i++) { $pw = $pw . chr($rpw[$i]+7); }
return($pw);
}
1;

```

### Source Code For probadd submit.pl

```

#!/c:/perl/bin/perl.exe
require "cgi-lib.pl";
&ReadParse;
&PrintHeader;
$title = $in{title};
$pdesc = $in{prob_description};
$tlen = length($title);
$tdesc = length($pdesc);
if ($tlen eq "0") {
print "<SCRIPT Language=JavaScript> alert('Problem Title cannot be left empty');";
print "history.back(); </SCRIPT>";
exit;
}
else {
if ($tdesc eq "1" or $tdesc eq "0") {
print "<SCRIPT Language=JavaScript> alert('Problem Description cannot be left empty');";
print "history.back(); </SCRIPT>";
exit;
}
}
$cookie1 = $ENV{'HTTP_COOKIE'};
($fname, $lname, $login, $pswd) = split(/;/, $cookie1);
($fname1, $fname2) = split(/=/, $fname);
($lname1, $lname2) = split(/=/, $lname);
($login1, $login2) = split(/=/, $login);
($pswd1, $pswd2) = split(/=/, $pswd);
$date = localtime(time);
use Win32::ODBC;

```

```

if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
    print "<HTML><HEAD><TITLE>Database Test</TITLE></HEAD>";
    print "<BODY><H2>Error Connecting to Database</H2>";
    print "<P>Please Try Your Request at A Later Time</P></BODY></HTML>";
    exit;
}
if (!$Db->Sql("SELECT EMPLOYEE_NO FROM EMPLOYEE_INFO WHERE LOGIN_ID = \'$login2\'
AND PASSWORD=\'$pswd2\'")) {
    if ($Db->FetchRow()) {
        undef %Data;
        %Data = $Db->DataHash();
        $empno=$Data{'EMPLOYEE_NO'};
        $Db->sql("INSERT INTO PROBLEM_DEFINITION(EMPLOYEE_NO,
        PROBLEM_DATE_TIME, PROBLEM_TITLE, PROBLEM_DESCRIPTION)
        VALUES('$empno', to_date('$date','DY MON DD HH24:MI:SS YYYY'), '$title',
        '$pdesc')");
        $Db->sql("COMMIT");
        print "<SCRIPT Language=JavaScript> alert('Successfully Added'); </SCRIPT>";
        exec("perl", "post_prob.cgi");
        close($Db);
        exit;
    }
}
}

```

### Source Code For post\_prob.cgi

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;
&PrintHeader;
$cookie1 = $ENV{'HTTP_COOKIE'};
($fname, $lname, $login, $pswd) = split(/;/, $cookie1);
($fname1, $fname2) = split(/=/, $fname);
($lname1, $lname2) = split(/=/, $lname);
($login1, $login2) = split(/=/, $login);
($pswd1, $pswd2) = split(/=/, $pswd);
$date=localtime(time);
@dt = split(" ", $date);
$date1 = $dt[2] . "-" . $dt[1] . "-" . $dt[4] . "." . $dt[3] . "." . $dt[0];
use Win32::ODBC;
if ( !$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project") ) {
    $a = Win32::ODBC::Error();
    print "<HTML><BODY><P> $a </P></BODY></HTML>";
    exit;
}
print <<"HD";
<HTML><HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
<BODY><FORM Action="/cgi-bin/probadd.pl" Method="GET">
<CENTER><H4> Problems (ADD) </H4></CENTER>

```

```

<TABLE Align=Center Width=95%><TR>
  <TD Align=Left Width=20%><FONT Color="#23238E" Face="Arial" Size=1><B>
    Posted by : </FONT></B></TD>
  <TD Align=Left><FONT Color="#23238E" Face="Arial" Size=1> $fname2 $lname2
    </FONT></TD>
</TR><TR>
  <TD Align=Left Width=20%><FONT Color="#23238E" Face="Arial" Size=1><B> On :
    </FONT></B></TD>
  <TD Align=Left><FONT Color="#23238E" Face="Arial" Size=1> $date1
    </FONT></TD>
</TR><TR>
  <TD Align=Left Width=20%><FONT Color="#23238E" Face="Arial" Size=1><B>
    Title : </FONT></B></TD>
  <TD Align=Left><FONT Color="#23238E" Face="Arial" Size=1><INPUT
    Name="title" MaxLength=60 onBlur="this.value=this.value.toUpperCase();"
    Size=25 Type="Text"></FONT></TD>
</TR><TR>
  <TD Align=Left Width=20%><FONT Color="#23238E" Face="Arial" Size=1><B>
    Description : </FONT></B></TD>
  <TD Align=Left><FONT Color="#23238E" Face="Arial" Size=1><TEXTAREA
    Cols=35 Name="prob_description" onBlur="this.value=this.value.toUpperCase();"
    Rows=6></TEXTAREA></FONT></TD>
</TR></TABLE><BR>
<CENTER><INPUT Name="save" Type="Submit" Value="Submit">
  <INPUT Name="cancel" Type="Reset" value="Oops"></CENTER>
</FORM></BODY>
</HTML>

```

```

HD
close($Db);
exit;

```

### Source Code For plist add.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;
&PrintHeader;
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
  $a = Win32::ODBC::Error();
  print "<HTML><BODY><P> $a </P></BODY></HTML>";
  exit;
}
if (!$Db->Sql("SELECT PROBLEM_TITLE, EMPLOYEE_NO, PROBLEM_DATE_TIME,
  PROBLEM_DESCRIPTION FROM PROBLEM_DEFINITION ORDER BY
  PROBLEM_TITLE")) {
  print "<HTML><HEAD><LINK HRef=styles.css Rel=STYLESHEET Type=text/css></HEAD>";
  print "<BODY><CENTER><H4>Solutions (ADD)</H4></CENTER>";
  print "<TABLE Align=Center Width=100%><TR>";
  print "<TD><FONT Color=Blue Face=Arial Size=1><B>Problem Title</B></FONT></TD>";

```

```

print "<TD><FONT Color=Blue Face=Arial Size=1><B>Problem Description</B></FONT></TD>";
print "</TR>";
while($Db->FetchRow()) {
    undef %Data;
    %Data = $Db->DataHash();
    $pd = $Data{'PROBLEM_DESCRIPTION'};
    $pdt = $Data{'PROBLEM_DATE_TIME'};
    @pdtm = split(" ", $pdt);
    $pt = $Data{'PROBLEM_TITLE'};
    $send = $pdtm[0].$pdtm[1];
    print "<TR><TD Width=45%><A Href=sol_add.pl?go=$send><FONT Color=#23238E
        Face=Arial Size=1> $pt </FONT></A></TD><TD><FONT Color=#23238E Face=Arial
        Size=1> $pd </FONT></TD></TR>";
}
print "</TABLE></BODY></HTML>";
}
close ($Db);
exit;

```

### Source Code For sol\_add.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;
&PrintHeader;
$probdt = $in{'go'};
$cookie1 = $ENV{'HTTP_COOKIE'};
($fname, $name, $login, $pswd) = split(/:/, $cookie1);
($fname1, $fname2) = split(/=/, $fname);
($name1, $name2) = split(/=/, $name);
($login1, $login2) = split(/=/, $login);
($pswd1, $pswd2) = split(/=/, $pswd);
$date = localtime(time);
@dt = split(" ", $date);
$date1 = $dt[2] . "-" . $dt[1] . "-" . $dt[4] . "." . $dt[3] . "." . $dt[0];
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
    $a = Win32::ODBC::Error();
    print "<HTML><BODY><P> $a </P></BODY></HTML>";
    exit;
}
if (!$Db->Sql("SELECT EMPLOYEE_NO, PROBLEM_DATE_TIME, PROBLEM_TITLE,
    PROBLEM_DESCRIPTION FROM PROBLEM_DEFINITION WHERE
    to_char(PROBLEM_DATE_TIME, 'yyyy-mm-ddHH24:MI:SS') = '$probdt'")) {
    if ($Db->FetchRow()) {
        undef %Data;
        %Data = $Db->DataHash();
        if (!$Db->Sql("SELECT FIRST_NAME, LAST_NAME FROM EMPLOYEE_INFO WHERE
            EMPLOYEE_NO = '$Data{EMPLOYEE_NO}'")) {
            if ($Db->FetchRow()) {

```

```

undef %Data1;
%Data1 = $Db->DataHash();
$fname=$Data1{'FIRST_NAME'};
$lname=$Data1{'LAST_NAME'};
$name=$fname." ".$lname;
}
}
$Data{PROBLEM_DESCRIPTION} =~ s/;/\n/g;
$dt = $Data{PROBLEM_DATE_TIME};
$d = substr($dt, 0, 10);
@dd = split("-", $d);
$d = $dd[1] . "&month($dd[1]);
$d = $dd[2] . "-" . $dd[1] . "-" . $dd[0];
$t = substr($dt, 11);
$dt = $d . " " . $t;
@pdtm = split(" ", $dt);
$send = $pdtm[0].$pdtm[1];
print <<"HD";
<HTML>
<HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
<BODY><FORM Action=soladd_submit.pl Method=GET >
<CENTER><H4>PROBLEMS (VIEW)</H4></CENTER>
<TABLE Align=center Width=95%><TR>
<TD Align=Left Width=25%><FONT Color=#23238E Face=Arial Size=1>
<B>Title :</B></FONT></TD>
<TD Align=Left><FONT Color=#23238E Face=Arial
Size=1>$Data{PROBLEM_TITLE}</FONT></TD>
<INPUT Name=pt Type=Hidden Value=$Data{PROBLEM_TITLE}>
</TR><TR>
<Td Align=Left Width=25%><FONT Color=#23238E Face=Arial
Size=1><B>Posted by :</B></FONT></TD>
<TD Align=Left><FONT Color=#23238E Face=Arial Size=1> $name
</FONT></TD>
</TR><TR>
<TD Align=Left Width=25%><FONT Color=#23238E Face=Arial
Size=1><B>On :</B></FONT></Td>
<TD Align=Left><FONT Color=#23238E Face=Arial Size=1> $dt
</FONT></TD>
<INPUT Name=ptm Type=Hidden Value=$send>
</TR><TR>
<TD Align=Left Width=25%><FONT Color=#23238E Face=Arial
Size=1><B>Description :</B></FONT></TD>
<TD Align=Left><FONT Color=#23238E Face=Arial Size=1>
$Data{PROBLEM_DESCRIPTION} </FONT></TD>
</TR><TR>
<TD Align=Left Width=20%><FONT Color=#23238E Face=Arial
Size=1><B>Solution :</B></FONT></TD>
<TD Align=Left><FONT Color=#23238E Face=Arial Size=1><TEXTAREA
Cols=35 onBlur="this.value=this.value.toUpperCase();" Name=solution
Rows=4></TEXTAREA></FONT></TD>

```

```

</TR><TR>
  <TD Align=Left Width=20%><FONT Color=#23238E Face=Arial
    Size=1><B>Posted By :</FONT> </B></TD>
  <TD Align=Left Width=20%><FONT Color=#23238E Face=Arial Size=1>
    $fname2 \ $lname2 on $date1 </FONT></TD>
</TR></TABLE><BR>
<CENTER><INPUT Name="save" Type=Submit Value="Submit">
  <INPUT Name="cancel" Type=Reset Value="Reset"></CENTER>
</FORM></BODY>
</HTML>

```

```

HD
  }
}

```

### Source Code For soladd submit.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
&ReadParse;
&PrintHeader;
$sol = ${solution};
$pt = ${pt};
$pd = ${ptm};
if ($sol eq "1" or $sol eq "0") {
  print "<SCRIPT Language=JavaScript> alert('Problem Description cannot be left empty');";
  print "history.back();</SCRIPT>";
  exit;
}
$cookie1 = $ENV{'HTTP_COOKIE'};
($fname, $lname, $login, $pswd) = split(/,/, $cookie1);
($fname1, $fname2) = split(/=/, $fname);
($lname1, $lname2) = split(/=/, $lname);
($login1, $login2) = split(/=/, $login);
($pswd1, $pswd2) = split(/=/, $pswd);
$date = localtime(time);
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
  print "Content-type: text/html\n\n";
  print "<HTML><HEAD><TITLE>Database Test</TITLE></HEAD>";
  print "<BODY><H2>Error Connecting to Database</H2>";
  print "<P>Please Try Your Request at A Later Time</P></BODY></HTML>";
  exit;
}
if (!$Db->Sql("SELECT EMPLOYEE_NO FROM EMPLOYEE_INFO WHERE LOGIN_ID = \'$login2\'
  AND PASSWORD=\'$pswd2\'")) {
  if ($Db->FetchRow()) {
    undef %Data;
    %Data = $Db->DataHash();
    $empno=$Data{'EMPLOYEE_NO'};

```

```

$Db->sql("INSERT INTO PROBLEM_SOLUTIONS(EMPLOYEE_NO,
      PROBLEM_DATE_TIME, SOLUTION_DATE_TIME,
      SOLUTION_DESCRIPTION) VALUES('$empno', to_date('$pdt', 'DD-MON-
      YYYYHH24:MI:SS'), to_date('$date', 'DY MON DD HH24:MI:SS YYYY'), '$sol')");
if ($Db->Error()) {
    $a = Win32::ODBC::Error();
    print "<HTML><BODY><P> $a </P></BODY></HTML>";
    exit;
}
print "<SCRIPT Language='JavaScript'> alert('Successfully Added'); </SCRIPT>";
exec("perl", "post_sol.cgi");
close($Db);
exit;
}
}
}

```

### Source Code For post\_sol.cgi

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
    $a = Win32::ODBC::Error();
    print "<HTML><BODY><P> $a </P></BODY></HTML>";
    exit;
}
if (!$Db->Sql("SELECT PROBLEM_TITLE, EMPLOYEE_NO, PROBLEM_DATE_TIME,
      PROBLEM_DESCRIPTION FROM PROBLEM_DEFINITION ORDER BY
      PROBLEM_TITLE")) {
    print "<HTML><HEAD><LINK HRef=styles.css Rel=STYLESHEET Type=text/css></HEAD>";
    print "<BODY><CENTER><H4>Solutions (ADD)</H4></CENTER>";
    print "<TABLE Align=Center Width=100%><TR>";
    print "<TD><FONT Color=Blue Face=Arial Size=1><B>Problem Title</B></FONT></TD>";
    print "<TD><FONT Color=Blue Face=Arial Size=1><B>Problem Description</B></FONT></TD>";
    print "</TR>";
    while($Db->FetchRow()) {
        undef %Data;
        %Data = $Db->DataHash();
        $pd = $Data{'PROBLEM_DESCRIPTION'};
        $pdt = $Data{'PROBLEM_DATE_TIME'};
        @pdtm = split(" ", $pdt);
        $pt = $Data{'PROBLEM_TITLE'};
        $send = $pdtm[0].$pdtm[1];
        print "<TR><TD Width=45%><A HRef=sol_add.pl?go=$send><FONT Color=#23238E
            Face=Arial Size=1> $pt </FONT></A></TD><TD><FONT Color=#23238E Face=Arial
            Size=1> $pd </FONT></TD></TR>";
    }
}

```

```

print "</TABLE></BODY></HTML>";
}
close ($Db);
exit;

```

### Source Code For plist\_view.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;
&PrintHeader;
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
    $a = Win32::ODBC::Error();
    print "Content-type: text/html\n\n";
    print "<HTML><BODY><P> $a </P></BODY></HTML>";
    exit;
}
if (!$Db->Sql("SELECT PROBLEM_TITLE, PROBLEM_DESCRIPTION, PROBLEM_DATE_TIME
FROM PROBLEM_DEFINITION ORDER BY PROBLEM_TITLE")) {
    print <<"HD";
    <HTML><HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
    <BODY><FORM Action=prob_det.pl Method=GET>
        <CENTER><H4>Problems & Solutions</H4></CENTER>
        <TABLE Align=Center Width=100%><TR>
            <TD><FONT Color=Blue Face=Arial Size=1><B>Problem
                Title</B></FONT></TD>
            <TD><FONT Color=Blue Face=Arial Size=1><B>Problem
                Description</B></FONT></TD>
        </TR>
    </FORM>
    HD
    while($Db->FetchRow()) {
        undef %Data;
        %Data = $Db->DataHash();
        $pd = $Data{'PROBLEM_DESCRIPTION'};
        $pdt = $Data{'PROBLEM_DATE_TIME'};
        @pdtm = split(" ", $pdt);
        $pt = $Data{'PROBLEM_TITLE'};
        $send = $pdtm[0].$pdtm[1];
        print "<TR><TD Width=45%><A HRef=prob_det.pl?go=$send><FONT Color=#23238E
            Face=Arial Size=1> $pt </FONT></A></TD><TD><FONT Color=#23238E Face=Arial
            Size=1> $pd </FONT></TD></TR>";
    }
    print "</TABLE></FORM></BODY></HTML>";
}
close ($Db);
exit;

```



Source Code For prob det.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;
&PrintHeader;
$probdt = $in{'go'};
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
    $a = Win32::ODBC::Error();
    print "<HTML><BODY><P> $a </P></BODY></HTML>";
    exit;
}
if (!$Db->Sql("SELECT EMPLOYEE_NO, PROBLEM_DATE_TIME, PROBLEM_TITLE,
    PROBLEM_DESCRIPTION FROM PROBLEM_DEFINITION WHERE
    to_char(PROBLEM_DATE_TIME, 'yyyy-mm-ddHH24:MI:SS') = '$probdt'")) {
    if ($Db->FetchRow()) {
        undef %Data;
        %Data = $Db->DataHash();
        if (!$Db->Sql("SELECT FIRST_NAME, LAST_NAME FROM EMPLOYEE_INFO WHERE
            EMPLOYEE_NO = '$Data{EMPLOYEE_NO}'")) {
            if ($Db->FetchRow()) {
                undef %Data1;
                %Data1 = $Db->DataHash();
                $fname=$Data1{'FIRST_NAME'};
                $lname=$Data1{'LAST_NAME'};
                $name=$fname." ".$lname;
            }
        }
        $Data{PROBLEM_DESCRIPTION}=~/s;/\n/g;
        $dt = $Data{PROBLEM_DATE_TIME};
        $d = substr($dt, 0,10);
        @dd = split("-", $d);
        $dd[1] = &month($dd[1]);
        $d = $dd[2] . "-" . $dd[1] . "-" . $dd[0];
        $t = substr($dt, 11);
        $dt = $d . " " . $t;
        @pdtm = split(" ", $dt);
        $sendpt = $pdtm[0].$pdtm[1];
        print <<"HD";
        <HTML>
        <HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
        <BODY><CENTER><H4>Problem & Solutions (VIEW)</H4></CENTER>
        <TABLE Align=Center Width=95%><TR>
            <TD Align=Left Width =25%><FONT Color=#23238E Face=Arial
                Size=1><b> Problem Title :</B></FONT></TD>
            <TD Align=Left><FONT Color=#23238E Face=Arial Size=1>
                $Data{PROBLEM_TITLE} </FONT></TD>
        </TR><TR>

```

```

<TD Align=Left Width=25%><FONT Color=#23238E Face=Arial
Size=1><B>Posted by : </B></FONT></TD>
<TD Align=Left><FONT Color=#23238E Face=Arial Size=1> $name
</FONT></TD>
</TR><TR>
<TD Align=Left Width=25%><FONT Color=#23238E Face=Arial
Size=1><B>On : </B></FONT></TD>
<TD Align=Left><FONT Color=#23238E Face=Arial Size=1> $dt
</FONT></TD>
</TR><TR>
<TD Align=Left Width=25%><FONT Color=#23238E Face=Arial
Size=1><B>Description : </B></FONT></TD>
<TD Align=Left><FONT Color=#23238E Face=Arial Size=1>
$Data{PROBLEM_DESCRIPTION} </FONT></TD>
</TR><TR>
<TD Align=Left Width=25%><FONT Color=#23238E Face=Arial
Size=1><B>Solution/s : </B></FONT></TD>

```

HD

```

if (!$Db->Sql("SELECT * FROM problem_solutions WHERE
to_char(PROBLEM_DATE_TIME, 'yyyy-mm-ddHH24:MI:SS') = '$probdt' ")) {
    $no = 1;
    while ($Db->FetchRow()) {
        undef %Data1;
        %Data1 = $Db->DataHash();
        $sdt = $Data1{SOLUTION_DATE_TIME};
        $sd = substr($sdt, 0, 10);
        @sdd = split("-", $sd);
        $sdd[1] = &month($sdd[1]);
        $sd = $sdd[2] . "-" . $sdd[1] . "-" . $sdd[0];
        $st = substr($sdt, 11);
        $sdt = $sd . " " . $st;
        @sdtm = split(" ", $sdt);
        $sendst = $sdtm[0].$sdtm[1];
        $solno = "Solution" . $no;
        $godata = $solno . " : " . $sendst . " : " . $sendst;
        print "<TD><FONT Color=#23238E Face=Arial Size=1><A
        HRef=sol_det.pl?sd=$godata> $solno\n </A> </FONT></TD>";
        print "</TR><TR><TD></TD>";
        $no = $no + 1;
    }
}
print "</TR></TABLE></FORM></BODY></HTML>";
}
}

```

### Source Code For sol\_det.pl

```

#!/c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;

```

```

&PrintHeader;
$data = $in{'sd'};
@datas = split(":", $data);
$pdtt = $datas[1];
$sddt = $datas[2];
$sno = substr($datas[0], 8);
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
    $a = Win32::ODBC::Error();
    print "<HTML><BODY><P> $a </P></BODY></HTML>";
    exit;
}
if (!$Db->Sql("SELECT EMPLOYEE_NO, PROBLEM_DATE_TIME, SOLUTION_DATE_TIME,
    SOLUTION_DESCRIPTION FROM PROBLEM_SOLUTIONS WHERE
    to_char(PROBLEM_DATE_TIME, 'DD-MON-YYYYHH24:MI:SS') = '$pdtt' AND
    to_char(SOLUTION_DATE_TIME, 'DD-MON-YYYYHH24:MI:SS') = '$sddt'")) {
    if ($Db->FetchRow()) {
        undef %Data;
        %Data = $Db->DataHash();
        if (!$Db->Sql("SELECT FIRST_NAME, LAST_NAME FROM EMPLOYEE_INFO WHERE
            EMPLOYEE_NO = '$Data{EMPLOYEE_NO}'")) {
            if ($Db->FetchRow()) {
                undef %Data1;
                %Data1 = $Db->DataHash();
                $fname=$Data1{'FIRST_NAME'};
                $lname=$Data1{'LAST_NAME'};
                $name=$fname." ".$lname;
            }
        }
        $dt = $Data{SOLUTION_DATE_TIME};
        $d = substr($dt, 0, 10);
        @dd = split("-", $d);
        $dd[1] = &month($dd[1]);
        $d = $dd[2] . "-" . $dd[1] . "-" . $dd[0];
        $t = substr($dt, 11);
        $dt = $d . " " . $t;
        print <<"HD";
        <HTML>
        <HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
        <BODY><CENTER><H4>Solutions (VIEW)</H4></CENTER>
        <TABLE Align=Center Width=95%><TR>
            <TD Align=Left Width=25%><FONT Color=#23238E Face=Arial
                Size=1><B>Posted by : </B></FONT></TD>
            <TD Align=Left><FONT Color=#23238E Face=Arial Size=1> $name
                </FONT></TD>
        </TR><TR>
            <TD Align=Left Width=25%><FONT Color=#23238E Face=Arial
                Size=1><B>On : </B></FONT></TD>
            <TD Align=Left><FONT Color=#23238E Face=Arial Size=1> $dt
                </FONT></TD>
    
```

```

        </TR><TR>
          <TD Align=Left Width=25%><FONT Color=#23238E Face=Arial
            Size=1><B>Solution : </B></FONT></TD>
          <TD Align=Left><FONT Color=#23238E Face=Arial Size=1>
            $Data{ SOLUTION_DESCRIPTION } </FONT></TD>
        </TR></TABLE>
        <CENTER><INPUT Name=btnBack onClick="history.back();" Type=Button
          Value="Return To Previous List"></CENTER>
    </BODY>
</HTML>
HD
}
}

```

### Source Code For tip.htm

```

<HTML>
  <FRAMESET Cols="21%,*" FrameSpacing=NO>
    <FRAME BgColor=Yellow FrameBorder=NO Name="list" Src="tippg.htm">
    <FRAMESET FrameSpacing=NO Rows="10%, 80%, *">
      <FRAME FrameBorder=NO Name="head" Src="/cgi-bin/header.pl">
      <FRAME FrameBorder=NO Name="show" Src="ttabout.htm">
      <FRAME FrameBorder=NO Name="foot" Src="footer.htm">
    </FRAMESET>
  </FRAMESET>
</HTML>

```

### Source Code For tippg.htm

```

<HTML>
  <HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
  <BODY Background=Image/fade_bg.jpg LeftMargin=0 RightMargin=0><BR><BR><BR><BR>
    <A HRef=ps.htm Target=_top><FONT Color=#23238E Face=Times New Roman
      Size=1><STRONG>Problems & Solutions</STRONG></FONT></A><BR><BR>
    <A HRef=home.htm Target=_top><FONT Color=#23238E Face=Times New Roman
      Size=1><STRONG>Tips & Tricks</STRONG></FONT></A><BR><BR>
    <IMG Src=Images/tball.gif>
    <A AccessKey="A" HRef=/cgi-bin/tipadd.pl Target="show"><FONT Color=#23238E
      Face=Times New Roman Size=1><STRONG>Add Tips</STRONG></FONT></A><BR>
    <IMG Src=Images/tball.gif>
    <A AccessKey="V" HRef=/cgi-bin/tiplist.pl Target="show"><FONT Color=#23238E
      Face=Times New Roman Size=1><STRONG>View
      Tips</STRONG></FONT></A><BR><BR>
    <A HRef=ei.htm Target=_top><FONT Color=#23238E Face=Times New Roman
      Size=1><STRONG>Employee Information</STRONG></FONT></A><BR><BR>
    <A HRef=home.htm Target=_top><FONT Color=#23238E Face=Times New Roman
      Size=1><STRONG>Home</STRONG></FONT></A><BR><BR>
    <A HRef=index.htm Target=_top><FONT Color=#23238E Face=Times New Roman
      Size=1><STRONG>Logout</STRONG></FONT></A>
  </BODY>
</HTML>

```

Source Code For ttabout.htm

```
<HTML>
  <HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
  <BODY><BR> <BR>
    <IMG Src=Images/blu-line.gif Width=100%>
    <FONT Size=2><B><CENTER> Looking for some Tips & Tricks ??<BR><BR>
    or<BR><BR> Wish to post some Tips & Tricks ??<BR> <BR> <BR></CENTER>
    &nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&nbsp;&.....Click on the required option</B></FONT>
    <IMG Src=Images/blu-line.gif Width=100%>
  </BODY>
</HTML>
```

Source Code For tipadd.pl

```
#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
&ReadParse;
&PrintHeader;
$cookie1 = $ENV{'HTTP_COOKIE'};
($fname, $lname, $login, $pswd) = split(/,/,$cookie1);
($fname1, $fname2) = split(/=,$fname);
($lname1, $lname2) = split(/=,$lname);
($login1, $login2) = split(/=,$login);
($pswd1, $pswd2) = split(/=,$pswd);
$date = localtime(time);
@dt = split(" ", $date);
$date = $dt[2] . "-" . $dt[1] . "-" . $dt[4] . " " . $dt[3] . " " . $dt[0];
print <<"HD";
  <HTML><HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
  <BODY><FORM Action=tipssubmit.pl Method=GET>
    <CENTER><H4>Tips & Tricks (ADD)</H4></CENTER>
    <TABLE Align=Center Width=95%><TR>
      <TD Align=Left Width=20%><FONT Color=#23238E Face=Arial Size=1><B>Posted
        by : </B></FONT></TD>
      <TD Align=Left><FONT Color=#23238E Face=Arial Size=1> $fname2 $lname2
        </FONT></TD>
    </TR><TR>
      <TD Align=Left Width=20%><FONT Color=#23238E Face=Arial Size=1><B>On :
        </B></FONT></TD>
      <TD Align=Left><FONT Color=#23238E Face=Arial Size=1> $date </FONT></TD>
    </TR><TR>
      <TD Align=Left Width =20%><FONT Color=#23238E Face=Arial Size=1><b>Title
        :</B></FONT></TD>
      <TD Align=Left><FONT Color=#23238E Face=Arial Size=1><INPUT Name=title
        onBlur="this.value=this.value.toUpperCase();" Type=Text
        Size=25></FONT></TD>
    </TR><TR>
      <TD Align=Left Width =20%><FONT Color=#23238E Face=Arial
        Size=1><b>Description :</B></FONT></TD>
```

```

<TD Align=Left><FONT Color=#23238E Face=Arial Size=1><TEXTAREA Cols=35
  Name=tip_description onBlur="this.value=this.value.toUpperCase();"
  Rows=6></TEXTAREA></FONT></TD>
</TR><TABLE><BR>
<CENTER><INPUT Name="cancel" Type="Submit" Value="Submit">
<INPUT Name="cancel" Type="Reset" Value="Reset"></CENTER>
</FORM></BODY>
</HTML>

```

```

HD
close($Db);
exit;

```

### Source Code For tipsubmit.pl

```

#!/c:/perl/bin/perl.exe
require "cgi-lib.pl";
&ReadParse;
&PrintHeader;
$title = $in{title};
$desc = $in{tip_description};
$stlen = length($title);
$stdesc = length($desc);
if ($stlen eq "0") {
    print "<SCRIPT Language='JavaScript'> alert('Problem Title cannot be left empty');";
    print "history.back(); </SCRIPT>";
    exit;
}
else {
    if ($stdesc eq "1" or $stdesc eq "0") {
        print "<SCRIPT Language='JavaScript'> alert('Problem Description cannot be left empty');";
        print "history.back(); </SCRIPT>";
        exit;
    }
}
$cookie1 = $ENV{'HTTP_COOKIE'};
($fname, $lname, $login, $pswd) = split(/;/, $cookie1);
($fname1, $fname2) = split(/=/, $fname);
($lname1, $lname2) = split(/=/, $lname);
($login1, $login2) = split(/=/, $login);
($pswd1, $pswd2) = split(/=/, $pswd);
$date = localtime(time);
use Win32::ODBC;
if (!(($Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project"))) {
    print "Content-type: text/html\n\n";
    print "<HTML><HEAD><TITLE>Database Test</TITLE></HEAD>";
    print "<BODY><H2>Error Connecting to Database</H2>";
    print "<P>Please Try Your Request at A Later Time</P></BODY></HTML>";
    exit;
}

```

```

if (!$Db->Sql("SELECT EMPLOYEE_NO FROM EMPLOYEE_INFO WHERE LOGIN_ID = \'$login2\'
AND PASSWORD=\'$pswd2\'")) {
    if ($Db->FetchRow()) {
        undef %Data;
        %Data = $Db->DataHash();
        $empno=$Data{'EMPLOYEE_NO'};
        $Db->sql("INSERT INTO PROBLEM_SOLUTIONS(EMPLOYEE_NO,
            SOLUTION_DATE_TIME, SOLUTION_TITLE, SOLUTION_DESCRIPTION)
            VALUES('$empno', to_date('$date', 'DY MON DD HH24:MI:SS YYYY'), '$title',
            '$desc')");
        $Db->sql("COMMIT");
        print "<SCRIPT Language='JavaScript'> alert('Successfully Added'); </SCRIPT>";
        exec("perl", "post_tip.cgi");
        close($Db);
        exit;
    }
}

```

### Source Code For post\_tip.cgi

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
&ReadParse;
$cookie1 = $ENV{'HTTP_COOKIE'};
($fname, $lname, $login, $pswd) = split(/;/, $cookie1);
($fname1, $fname2) = split(/=/, $fname);
($lname1, $lname2) = split(/=/, $lname);
($login1, $login2) = split(/=/, $login);
($pswd1, $pswd2) = split(/=/, $pswd);
$date = localtime(time);
@dt = split(" ", $date);
$date = $dt[2] . "-" . $dt[1] . "-" . $dt[4] . "." . $dt[3] . "." . $dt[0];
print <<"HD";
<HTML><HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
<BODY><FORM Action=tipsubmit.pl Method=GET>
<CENTER><H4>Tips & Tricks (ADD)</H4></CENTER>
<TABLE Align=Center Width=95%><TR>
<TD Align=Left Width=20%><FONT Color=#23238E Face=Arial Size=1><B>Posted
by : </B></FONT></TD>
<TD Align=Left><FONT Color=#23238E Face=Arial Size=1> $fname2 $lname2
</FONT></TD>
</TR><TR>
<TD Align=Left Width=20%><FONT Color=#23238E Face=Arial Size=1><B>On :
</B></FONT></TD>
<TD Align=Left><FONT Color=#23238E Face=Arial Size=1> $date </FONT></TD>
</TR><TR>
<TD Align=Left Width =20%><FONT Color=#23238E Face=Arial Size=1><b>Title
:</B></FONT></TD>
<TD Align=Left><FONT Color=#23238E Face=Arial Size=1><INPUT Name=title
onBlur="this.value=this.value.toUpperCase();" Type=Text
Size=25></FONT></TD>

```

```

</TR><TR>
  <TD Align=Left Width =20%><FONT Color=#23238E Face=Arial
    Size=1><b>Description :</B></FONT></TD>
  <TD Align=Left><FONT Color=#23238E Face=Arial Size=1><TEXTAREA Cols=35
    Name=tip_description onBlur="this.value=this.value.toUpperCase();"
    Rows=6></TEXTAREA></FONT></TD>
</TR><TABLE><BR>
  <CENTER><INPUT Name="cancel" Type="Submit" Value="Submit">
  <INPUT Name="cancel" Type="Reset" Value="Reset"></CENTER>
</FORM></BODY>
</HTML>

```

```

HD
close($Db);
exit;

```

### Source Code For tiplist.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;
&PrintHeader;
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
  $a = Win32::ODBC::Error();
  print "<HTML><BODY><P> $a </P></BODY></HTML>";
  exit;
}
if (!$Db->Sql("SELECT SOLUTION_DATE_TIME, SOLUTION_TITLE, SOLUTION_DESCRIPTION
  FROM PROBLEM_SOLUTIONS ORDER BY SOLUTION_TITLE")) {
  print "<HTML>";
  print "<HEAD><LINK HRef=styles.css Rel=STYLESHEET Type=text/css></HEAD>";
  print "<BODY><FORM Action=tipinfo.pl Method=GET>";
  print "<CENTER><H4>Tips & Tricks</H4></CENTER>";
  print "<FONT Color=Blue Face=Arial Size=1><B>Tip Title</B></FONT><BR><BR>";
  print "<TABLE Align=Center Width=100%>";
  while($Db->FetchRow()) {
    undef %Data;
    %Data = $Db->DataHash();
    $sdt = $Data{'SOLUTION_DATE_TIME'};
    @sdtm = split(" ", $sdt);
    $st = $Data{'SOLUTION_TITLE'};
    $send = $sdtm[0].$sdtm[1];
    print "<TR><TD><A HRef=tipinfo.pl?go=$send><FONT Color=#23238E Face=Arial Size=1>
      $st </FONT></A></TD><TD></TD></TR>";
  }
  print "</TABLE></FORM></BODY></HTML>";
}
close ($Db);
exit;

```



Source Code For tipinfo.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;
&PrintHeader;
$soldt = $in{'go'};
use Win32::ODBC;
if (!( $Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project"))) {
    $a = Win32::ODBC::Error();
    print "<HTML><BODY><P> $a </P></BODY></HTML>";
    exit;
}
if (!( $Db->Sql("SELECT EMPLOYEE_NO, SOLUTION_DATE_TIME, SOLUTION_TITLE,
    SOLUTION_DESCRIPTION FROM PROBLEM_SOLUTIONS WHERE
    to_char(SOLUTION_DATE_TIME, 'yyyy-mm-ddHH24:MI:SS') = \'$soldt\'")) {
    if ( $Db->FetchRow() ) {
        undef %Data;
        %Data = $Db->DataHash();
        if ( ! $Db->Sql("SELECT FIRST_NAME, LAST_NAME FROM EMPLOYEE_INFO WHERE
            EMPLOYEE_NO = \'$Data{EMPLOYEE_NO}\'")) {
            if ( $Db->FetchRow() ) {
                undef %Data1;
                %Data1 = $Db->DataHash();
                $fname=$Data1{'FIRST_NAME'};
                $lname=$Data1{'LAST_NAME'};
                $name=$fname." ".$lname;
            }
        }
        $Data{SOLUTION_DESCRIPTION}=~/s/;\n/g;
        $dt = $Data{SOLUTION_DATE_TIME};
        $d = substr($dt, 0,10);
        @dd = split ("-", $d);
        $dd[1] = &month($dd[1]);
        $d = $dd[2] . "-" . $dd[1] . "-" . $dd[0];
        $t = substr($dt, 11);
        $dt = $d . " " . $t;
        print <<"HD";
        <HTML>
        <HEAD><LINK href="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
        <BODY><CENTER><H4>Tips & Tricks</H4></CENTER>
        <TABLE Align=Center Width=95%><TR>
            <TD Align=Left Width=25%><FONT Color=#23238E Face=Arial
                Size=1><B>Tip Title :</B></FONT></TD>
            <TD Align=Left> <FONT Color=#23238E Face=Arial Size=1>
                $Data{SOLUTION_TITLE} </FONT></TD>
        </TR><TR>

```

```

        <TD Align=Left Width=25%><FONT Color=#23238E Face=Arial
            Size=1><B>Posted by :</B></FONT></TD>
        <TD Align=Left> <FONT Color=#23238E Face=Arial Size=1> $name
            </FONT></TD>
    </TR><TR>
        <TD Align=Left Width=25%><FONT Color=#23238E Face=Arial
            Size=1><B>On :</B></FONT></TD>
        <TD Align=Left> <FONT Color=#23238E Face=Arial Size=1> $dt
            </FONT></TD>
    </TR><TR>
        <TD Align=Left Width=25%><FONT Color=#23238E Face=Arial
            Size=1><B>Description :</B></FONT></TD>
        <TD Align=Left> <FONT Color=#23238E Face=Arial Size=1>
            $Data{SOLUTION_DESCRIPTION} </FONT></TD>
    </TR></TABLE>
    <CENTER><INPUT Name=btnBack onClick="history.back();" Type=Button
        Value="Return To Previous List"></CENTER>
</BODY>
</HTML>
HD
}
}

```

### Source Code For ei.htm

```

<HTML>
  <FRAMESET Cols="21%,*" FrameSpacing=NO>
    <FRAME BgColor=Yellow FrameBorder=NO Name="list" Src="employee.htm">
    <FRAMESET FrameSpacing=NO Rows="10%, 80%, *">
      <FRAME FrameBorder=NO Name="head" Src="/cgi-bin/header.pl">
      <FRAME FrameBorder=NO Name="show" Src="eiabout.htm">
      <FRAME FrameBorder=NO Name="foot" Src="footer.htm">
    </FRAMESET>
  </FRAMESET>
</HTML>

```

### Source Code For employee.htm

```

<HTML>
  <HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>
  <BODY Background=Images/fade_bg.jpg LeftMargin=0 RightMargin=0><BR><BR><BR><BR>
    <A HRef=ps.htm Target=_top><FONT Color=#23238E Face=Times New Roman
      Size=1><STRONG>Problems & Solutions</STRONG></FONT></A><BR><BR>
    <A HRef=tip.htm Target=_top><FONT Color=#23238E Face=Times New Roman
      Size=1><STRONG>Tips & Tricks</STRONG></FONT></A><BR><BR>
    <A HRef=home.htm Target=_top><FONT Color=#23238E Face=Times New Roman
      Size=1><STRONG>Employee Information</STRONG></FONT></A><BR><BR>
    <IMG Src=Images/tball.gif>
    <A AccessKey="A" HRef=/cgi-bin/eiadd.pl Target="show"><FONT Color=#23238E
      Face=Times New Roman Size=1><STRONG>Add</STRONG></FONT></A><BR>

```





Source Code For eiadd\_submit.pl

```
#!/c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;
&PrintHeader;
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
    $a = Win32::ODBC::Error();
    print "<HTML><BODY><P> $a </P></BODY></HTML>";
    exit;
}
$empno = $in{'employee_no'};
$name = $in{'first_name'};
$lname = $in{'last_name'};
$desg = $in{'designation'};
$doj = $in{'date_of_joining'};
$doc = $in{'date_of_confirmation'};
$login = $in{'login_id'};
$password = $in{'password'};
$no = substr($empno, 1);
$no = $no + 1;
$lueno = "E" . $no;
if (length($name) eq "0") {
    print "<SCRIPT Language=JavaScript> alert('First Name cannot be left empty');";
    print "history.back(); </SCRIPT>";
    exit;
}
if (length($lname) eq "0") {
    print "<SCRIPT Language=JavaScript> alert('Last Name cannot be left empty');";
    print "history.back(); </SCRIPT>";
    exit;
}
if (length($doj) eq "0") {
    print "<SCRIPT Language=JavaScript> alert('Date Of Joining cannot be left empty');";
    print "history.back(); </SCRIPT>";
    exit;
}
if (length($doc) eq "0") {
    print "<SCRIPT Language=JavaScript> alert('Date Of Confirmation cannot be left empty');";
    print "history.back(); </SCRIPT>";
    exit;
}
if (length($login) eq "0") {
    print "<SCRIPT Language=JavaScript> alert('Login Name cannot be left empty');";
    print "history.back(); </SCRIPT>";
    exit;
}
```

```

if (length($passwd) eq "0") {
    print "<SCRIPT Language='JavaScript'> alert('Password cannot be left empty');";
    print "history.back(); </SCRIPT>";
    exit;
}
$db->sql("INSERT INTO employee_info VALUES('$empno', '$fname', '$lname', '$desig', '$doj', '$doc',
'$login', '$passwd')");
if ($db->Error()) {
    $a = Win32::ODBC::Error();
    if ($a = "[1858] [1] '[Oracle][ODBC Oracle Driver][Oracle OCI]ORA-01858: a non-numeric
character was found where a numeric was expected.'"") {
        print "<SCRIPT Language='JavaScript'> alert('Please ReEnter the date in the specified format');";
        print " history.back(); </SCRIPT>";
        close($db);
        exit;
    }
}
else {
    print "<SCRIPT Language='JavaScript'> alert('Successfully Added'); </SCRIPT>";
    $db->sql("UPDATE lu_eno SET employee_no=\'$lueno\'");
    exec("perl", "post_eiadd.cgi");
    close($db);
    exit;
}

```

### Source Code For post\_eiadd.cgi

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;
use Win32::ODBC;
if (!$db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
    $a = Win32::ODBC::Error();
    print "<HTML><BODY><P> $a </P></BODY></HTML>";
    exit;
}
if (!$db->sql("select employee_no from lu_eno")) {
    if ($db->FetchRow()) {
        undef%Data;
        %Data = $db->DataHash();
        $lu_eno = $Data{'EMPLOYEE_NO'};
    }
}
print <<"HD";
<HTML>
<HEAD><LINK Rel=STYLESHEET Type="text/css" HRef="styles.css"></HEAD>
<BODY><FORM Action=eiadd_submit.pl Method=GET>
<CENTER><H4>Employee Information (ADD) </H4></CENTER>
<TABLE><TR>
<TD><FONT Color=#23238E Face=Arial Size=1>Employee No.</FONT></TD>

```



```

print "<HTML><BODY><P> $a </P></BODY></HTML>";
exit;
}
print <<"HD";
<HTML>
  <HEAD><LINK Rel=STYLESHEET Type="text/css" HRef="styles.css"></HEAD>
  <BODY><FORM Action=eiedit_info.pl Method=GET>
    <CENTER><H4>Employee Information (EDIT)</H4></CENTER><BR>
    Employee Name to be edited : <BR><BR>
    <SELECT Name="emp_name" Size="7">
HD
if(!$Db->sql("SELECT first_name FROM employee_info ORDER BY first_name")){
  while ($Db->FetchRow()) {
    undef%Data;
    %Data = $Db->DataHash();
    $name = $Data{'FIRST_NAME'};
    print "<OPTION Value='$name'> $name </OPTION>";
  }
}
print <<"HDD";
  </SELECT><BR><BR>
  <CENTER><INPUT Name="save" Type=SUBMIT Value="Details">
    <INPUT Name ="cancel" Type=RESET Value="Reset"></CENTER>
  </FORM></BODY>
</HTML>
HDD
close($Db);
exit;

```

### Source Code For eiedit\_info.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;
&PrintHeader;
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
  $a = Win32::ODBC::Error();
  print "<HTML><BODY><P> $a </P></BODY></HTML>";
  exit;
}
$name = $in{emp_name};
$len = length($name);
if ($len eq "0") {
  print "<SCRIPT Language='JavaScript'> alert('Please Select Employee Name to be Edited');";
  print "history.back(); </SCRIPT>";
}
print <<"HD";
<HTML>
  <HEAD><LINK HRef="styles.css" Rel=STYLESHEET Type="text/css"></HEAD>

```







```

$nl = length($passwd);
$sl = length($passwd);
if (length($fname) eq "0") {
    print "<SCRIPT Language=JavaScript> alert('First Name cannot be left empty');";
    print "history.back(); </SCRIPT>";
    exit;
}
if (length($lname) eq "0") {
    print "<SCRIPT Language=JavaScript> alert('Last Name cannot be left empty');";
    print "history.back(); </SCRIPT>";
    exit;
}
if (length($doj) eq "0") {
    print "<SCRIPT Language=JavaScript> alert('Date Of Joining cannot be left empty');";
    print "history.back(); </SCRIPT>";
    exit;
}
if (length($doc) eq "0") {
    print "<SCRIPT Language=JavaScript> alert('Date Of Confirmation cannot be left empty');";
    print "history.back(); </SCRIPT>";
    exit;
}
if (length($login) eq "0") {
    print "<SCRIPT Language=JavaScript> alert('Login Name cannot be left empty');";
    print "history.back(); </SCRIPT>";
    exit;
}
if ( !($nl == 0 && $sl == 0) ) {
    if ($passwd eq $passwd) { $passwd= $passwd ; }
    else {
        print "<SCRIPT Language=JavaScript> alert('Confirm the new password');";
        print "history.back(); </SCRIPT>";
        exit;
    }
}
$db->sql("UPDATE employee_info SET first_name='$fname', last_name='$lname', designation='$desg',
        DATE_OF_JOINING='$doj', DATE_OF_CONFIRMATION='$doc', LOGIN_ID='$login',
        PASSWORD='$passwd' WHERE employee_no = '$empno'");
if ($db->Error() {
    $a = Win32::ODBC::Error();
    if ($a = "[1858] [1] \'[Oracle][ODBC Oracle Driver][Oracle OCI]ORA-01858: a non-numeric
        character was found where a numeric was expected.'") {
        print "<SCRIPT Language=JavaScript>";
        print "alert('Please ReEnter the date in the specified format'); history.back();</SCRIPT>";
        close($db);
        exit;
    }
}
else {
    print "<SCRIPT Language=JavaScript> alert('Successfully Edited'); </SCRIPT>";
}

```

```

    exec("perl", "post_eiedit.cgi");
}
close($Db);
exit;

```

### Source Code For post\_eiedit.cgi

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
&ReadParse;
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
    $a = Win32::ODBC::Error();
    print "<HTML><BODY><P> $a </P></BODY></HTML>";
    exit;
}
print <<"HD";
<HTML>
    <HEAD><LINK Rel=STYLESHEET Type="text/css" HRef="styles.css"></HEAD>
    <BODY><FORM Action=eiedit_info.pl Method=GET>
        <CENTER><H4>Employee Information (EDIT)</H4></CENTER><BR>
        Employee Name to be edited : <BR><BR>
        <SELECT Name="emp_name" Size="7">
HD
if (!$Db->sql("SELECT first_name FROM employee_info ORDER BY first_name")) {
    while ($Db->FetchRow()) {
        undef %Data;
        %Data = $Db->DataHash();
        $name = $Data{'FIRST_NAME'};
        print "<OPTION Value='$name'> $name </OPTION>";
    }
}
print <<"HDD";
    </SELECT><BR><BR>
    <CENTER><INPUT Name="save" Type=SUBMIT Value="Details">
        <INPUT Name="cancel" Type=RESET Value="Reset"></CENTER>
    </FORM></BODY>
</HTML>
HDD
close($Db);
exit;

```

### Source Code For eidel.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
&ReadParse;
&PrintHeader;
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
    $a = Win32::ODBC::Error();

```

```

print "<HTML><BODY><P> $a </P></BODY></HTML>";
exit;
}
print <<"HD";
<HTML>
  <HEAD><LINK Rel=STYLESHEET Type="text/css" HRef="styles.css"> </HEAD>
  <BODY><FORM Action=eidel_info.pl Method=GET>
    <CENTER><H4>Employee Information (DELETE)</H4></CENTER><BR>
    Employee Name to be Deleted : <BR><BR><SELECT Name="emp_name" Size="7">
HD
if(!$Db->sql("SELECT first_name FROM employee_info ORDER BY first_name")) {
  while ($Db->FetchRow()) {
    undef%Data;
    %Data = $Db->DataHash();
    $name = $Data{'FIRST_NAME'};
    print "<OPTION Value='$name'>$name</OPTION>";
  }
}
print <<"HDD";
  </SELECT><BR><BR>
  <CENTER><INPUT Name="save" Type=SUBMIT Value="Details">
  <INPUT Name="cancel" Type=RESET Value="Reset"></CENTER>
</FORM></BODY>
</HTML>
HDD
close($Db);
exit;

```

### Source Code For eidel\_info.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;
&PrintHeader;
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
  $a = Win32::ODBC::Error();
  print "Content-type: text/html\n\n<HTML><BODY><P> $a </P></BODY></HTML>";
  exit;
}
$name = $in{emp_name};
$len = length($name);
if ($len eq "0") {
  print "<SCRIPT Language='JavaScript'> alert('Please Select Employee Name to be Deleted');";
  print "history.back(); </SCRIPT>";
  close($Db);
  exit;
}

```

```

print <<"HD";
<HTML>
  <HEAD><LINK Rel=STYLESHEET Type="text/css" HRef="styles.css"></HEAD>
  <Body><Form Action=eidel_submit.pl Method=GET>
    <CENTER><H3><U>Employee Information (DELETE)</U></H3></CENTER>
HD
$name = $in{emp_name};
if (!$Db->sql("SELECT * FROM employee_info WHERE FIRST_NAME = '$name'" )) {
  if ($Db->FetchRow()) {
    undef%Data;
    %Data = $Db->DataHash();
    $doj = $Data{'DATE_OF_JOINING'};
    $doj = substr($doj, 0,10);
    @dojj = split("-", $doj);
    $dojj[1] = &month($dojj[1]);
    $doc = $Data{'DATE_OF_CONFIRMATION'};
    $doc = substr($doc, 0,10);
    @docc = split("-", $doc);
    $docc[1] = &month($docc[1]);
    $doj = $dojj[2] . "-" . $dojj[1] . "-" . $dojj[0];
    $doc = $docc[2] . "-" . $docc[1] . "-" . $docc[0];
    $yearj = (substr($dojj[0], 2, 2));
    $yearc = (substr($docc[0], 2, 2));
    $doj = $dojj[2] . "-" . $dojj[1] . "-" . $yearj;
    $doc = $docc[2] . "-" . $docc[1] . "-" . $yearc;
    print <<"HDD";
    <TABLE><TR>
      <TD><FONT Color=#23238E Face=Arial Size=1>Employee No.</FONT></TD>
      <TD><FONT Color=#23238E Face=Arial Size=1>
        <INPUT Name="employee_no" onFocus=blur(); Type=Text Size=6
          Value=$Data{'EMPLOYEE_NO'}></FONT></TD>
    </TR><TR>
      <TD><FONT Color=#23238E Face=Arial Size=1>First Name</FONT></TD>
      <TD><FONT Color=#23238E Face=Arial Size=1>
        <INPUT Name="first_name" onFocus=blur(); Type=Text Size=20
          Value=$Data{'FIRST_NAME'}></FONT></TD>
    </TR><TR>
      <TD><FONT Color=#23238E Face=Arial Size=1>Last Name</FONT></TD>
      <TD><FONT Color=#23238E Face=Arial Size=1>
        <INPUT Name="last_name" onFocus = blur(); Type=Text Size=20
          Value=$Data{'LAST_NAME'}></FONT></TD>
    </TR><TR>
      <TD><FONT Color=#23238E Face=Arial Size=1>Designation</FONT></TD>
      <TD><FONT Color=#23238E Face=Arial Size=1>
        <INPUT Name="designation" onFocus = blur(); Type=Text Size=20
          Value=$Data{'DESIGNATION'}></FONT></TD>
    </TR><TR>
      <TD><FONT Color=#23238E Face=Arial Size=1>Join Date (DD-MON-YY)
        </FONT></TD>

```



```

use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
    $a = Win32::ODBC::Error();
    print "<HTML><BODY><P> $a </P></BODY></HTML>";
    exit;
}
print <<"HD";
<HTML>
    <HEAD><LINK Rel=STYLESHEET Type="text/css" HRef="styles.css"> </HEAD>
    <BODY><FORM Action=eidel_info.pl Method=GET>
        <CENTER><H4>Employee Information (DELETE)</H4></CENTER><BR>
        Employee Name to be Deleted : <BR><BR><SELECT Name="emp_name" Size="7">
HD
if (!$Db->sql("SELECT first_name FROM employee_info ORDER BY first_name")) {
    while ($Db->FetchRow()) {
        undef%Data;
        %Data = $Db->DataHash();
        $name = $Data{'FIRST_NAME'};
        print "<OPTION Value='$name'$>$name</OPTION>";
    }
}
print <<"HDD";
    </SELECT><BR><BR>
    <CENTER><INPUT Name="save" Type=SUBMIT Value="Details">
        <INPUT Name="cancel" Type=RESET Value="Reset"></CENTER>
    </FORM></BODY>
</HTML>
HDD
close($Db);
exit;

```

### Source Code For eiview.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
&ReadParse;
&PrintHeader;
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
    $a = Win32::ODBC::Error();
    print "<HTML><BODY><P> $a </P></BODY></HTML>";
    exit;
}
print <<"HD";
<HTML><HEAD><LINK Rel=STYLESHEET Type="text/css" HRef="styles.css"></HEAD>
    <BODY><FORM Action=eiview_info.pl Method=GET>
        <CENTER><H4>Employee Information (VIEW)</H4></CENTER><BR>
        Employee Name to be Viewed : <BR><BR><SELECT Name="emp_name" Size="7">
HD

```



```

if(!$Db->sql("SELECT first_name FROM employee_info ORDER BY first_name")) {
    while ($Db->FetchRow()) {
        undef%Data;
        %Data = $Db->DataHash();
        $name = $Data{'FIRST_NAME'};
        print "<OPTION Value='$name'>$name</OPTION>";
    }
}
print <<"HDD";
    </SELECT><BR><BR>
    <CENTER><INPUT Name="save" Type=SUBMIT Value="Details">
        <INPUT Name="cancel" Type=RESET Value="Reset"></CENTER>
</FORM></BODY>
</HTML>
HDD
close($Db);
exit;

```

### Source Code For eiview info.pl

```

#!c:/perl/bin/perl.exe
require "cgi-lib.pl";
require "mylib.pl";
&ReadParse;
&PrintHeader;
use Win32::ODBC;
if (!$Db=new Win32::ODBC("dsn=PerlOracle;UID=project;PWD=project")) {
    $a = Win32::ODBC::Error();
    print "<HTML><BODY><P> $a </P></BODY></HTML>";
    exit;
}
$name = $in{emp_name};
$len = length($name);
if ($len eq "0") {
    print "<SCRIPT Language='JavaScript'> alert('Please Select Employee Name to be Viewed');";
    print "history.back(); </SCRIPT>";
}
print <<"HD";
    <HTML><HEAD><LINK Rel=STYLESHEET Type="text/css" HRef="styles.css"></HEAD>
    <BODY><FORM Method=GET>
        <CENTER><H3><U>Employee Information (VIEW)</U></H3></CENTER>
HD
$name = $in{emp_name};
if (!$Db->sql("SELECT * FROM employee_info WHERE FIRST_NAME = \'$name\'")) {
    if ($Db->FetchRow()) {
        undef%Data;
        %Data = $Db->DataHash();
        $doj = $Data{'DATE_OF_JOINING'};
        $doj = substr($doj, 0,10);
        @dojj = split("-", $doj);
    }
}

```



```
<CENTER><INPUT Name=btnBack onClick="history.back();" Type=Button
Value="Done"></CENTER>
</FORM></BODY>
</HTML>
HDD
}
}
close($Db);
exit;
```

## ANSWERS TO SELF REVIEW QUESTIONS

### 13. COMMON GATEWAY INTERFACE CONCEPTS

#### FILL IN THE BLANKS

1. Common Gateway Interface
2. Gateway
3. Hyper Text Transfer Protocol
4. GET
5. Environment Variables
6. QUERY\_STRING
7. CONTENT\_TYPE

#### TRUE OR FALSE

8. True
9. False
10. False

### 14. THE PERL LANGUAGE

#### FILL IN THE BLANKS

1. Practical Extraction and Report Language
2. Double-quoted strings
3. Scalars
4. Associative Arrays
5. Keys
6. %ENV

#### TRUE OR FALSE

7. False
8. True
9. False
10. False

### 15. PERFORMING OPERATIONS AND CONTROLLING PROGRAM Flow

#### FILL IN THE BLANKS

1. ++
2. if
3. <=>
4. Numeric
5. True
6. Loop
7. last

#### TRUE OR FALSE

8. True
9. False
10. False
11. True
12. False

### 16. PERL FUNCTIONS

#### FILL IN THE BLANKS

1. Function
2. chop()
3. .=
4. x
5. substr()
6. splice()

#### TRUE OR FALSE

7. False
8. False
9. True

**17. FILEHANDLING****FILL IN THE BLANKS**

1. Files, Directories
2. open()
3. >
4. Text, Binary
5. read(), sysread()
6. mkdir()
7. opendir()
8. readdir()
9. (-e)

**TRUE OR FALSE**

10. True
11. True
12. False
13. True

**18. REGULAR EXPRESSIONS****FILL IN THE BLANKS**

1. Regular Expression
2. Anchors, Characters & Modifiers
3. ^
4. Backslash
5. +
6. |
7. !
8. =
9. . (dot)

**TRUE OR FALSE**

10. False
11. True
12. False
13. True
14. True

**19. CREATING STRUCTURED PROGRAMS****FILL IN THE BLANKS**

1. Subroutine
2. sub
3. Arguments
4. my and local
5. return()
6. Library
7. Cgi-lib.pl
8. PrintVariables
9. ReadParse

**TRUE OR FALSE**

10. False
11. False
12. True
13. False
14. True

**20. GOING THE OBJECT WAY WITH PERL****FILL IN THE BLANKS**

1. Reference
2. Backslash
3. \$
4. Arrow
5. Package
6. main
7. require
8. use
9. Method

**TRUE OR FALSE**

10. False
11. True
12. False
13. True

**21. DATABASE CONNECTIVITY****FILL IN THE BLANKS**

1. ODBC
2. Win32::ODBC
3. Data Connection
4. new
5. FetchRow
6. Close
7. DataHash

**TRUE OR FALSE**

8. True
9. False
10. True
11. True
12. False

**22. DEBUGGING IN PERL****FILL IN THE BLANKS**

1. Debugging
2. -d
3. q
4. w
5. //
6. Subroutine
7. X
8. b
9. Trace mode

**TRUE OR FALSE**

10. False
11. False
12. True
13. False
14. True

**23. INSTALLING AND SETTING UP APACHE WEB SERVER****FILL IN THE BLANKS**

1. apachectl configtest, -t
2. Alias, ScriptAlias
3. container
4. Apache's API
5. MinSpareServers, MaxSpareServers
6. DocumentRoot
7. CAN-2004-0493, CAN-2004-0488
8. KeepAliveTimeout
9. apxs (APache eXtension)

**TRUE OR FALSE**

10. True
11. False
12. False
13. True
14. False
15. True
16. True
17. False
18. False
19. True

# SOLUTIONS TO HANDS ON EXERCISES

## 14. THE PERL LANGUAGE

1. Assign string values and numeric values to scalar variables:

```
#!/c:/perl/bin/perl.exe -w
$num_days_in_year = 365;
$book_title = "Programming Using Open Source Products On Windows";
print "$num_days_in_year ";
print "$book_title ";
```

2. Creating an indexed array named @week to populate it with the days of the week and print the elements of the array:

```
#!/c:/perl/bin/perl.exe -w
@week=("Monday", "Tuesday", "Wednesday", "Thursday", "Friday", "Saturday");
for($i=0; $i<=$#week; $i++) {
    print "Index[$i]=$week[$i] \n";
}
```

3. Creating an associative array %expenses containing information for John, Mary, Ed & Jane with their expenses respectively as \$250.50, \$195.00, \$345.25 and \$225.99 and printing the name and the expenses of each element using a for loop, a foreach loop and a while loop:

```
#!/c:/perl/bin/perl.exe -w
%expenses=("John"=>"250.50", "Mary"=>"195.00", "Ed"=>"345.25", "Jane"=>"225.99");
print "\n Printing the array \n\n";
print "\nUsing the for loop\n\n";
for $key(keys %expenses) {
    print "Name=$key Expenses=$expenses{$key} \n";
}
print "\nUsing the foreach loop\n\n";
foreach $name(keys %expenses) {
    print "$name=$expenses{$name} \n";
}
print "\nusing the while loop\n\n";
while (($key,$value)=each(%expenses)) {
    print "Name:$key Expenses:$value \n";
}
```

4. Creating an indexed array @names by assigning individual elements of that array and then print the value of each element using a for loop:

```
#!/c:/perl/bin/perl.exe -w
print "\ncreated array\n\n";
$name[0]="Ivan";
$name[1]="Cynthia";
$name[2]="Chriselle";
```

```

$name[3]="Chhaya";
$name[4]="Sharanam";
$name[5]="Hansel";
$name[6]="Anil";
$name[7]="Alex";
$name[8]="Mamta";
$name[9]="Ashwini";
for ($i=0; $i<=$#name; $i++) {
    print "Name [$i]=$name[$i] \n";
}

```

## 15. PERFORMING OPERATIONS AND CONTROLLING PROGRAM FLOW

### 1. Printing the value of \$ctr2, \$ctr3 and \$ctr4 when \$ctr=10:

```

$ctr=10;
$ctr2=$ctr+1;
$ctr3=$ctr++;
$ctr4=++$ctr;
print "ctr2=$ctr2 \n";
print "ctr3=$ctr3 \n";
print "ctr4=$ctr4 \n";

```

### 2. Printing the value of \$a when \$a=10 and \$b=2:

```

$a=10;
$b=2;
$a+=$b;
print "a=$a \n";
$a-=$b;
print "a=$a \n";
$a*=$b;
print "a=$a \n";
$a/=$b;
print "a=$a \n";

```

### 3. A loop to add numbers from 50 to 75:

```

for($i=50; $i<=75; $i++) {
    $sum+=$i; }
print "sum=$sum \n";

```

### 4. Using the if ... else statement to categorize the marks of a student:

```

if ($marks>75) {
    print "Congratulations, you have secured a Distinction \n"; }
elseif ($marks>=60 && $marks<75) {
    print "Congratulations, you have secured a First Class \n"; }
elseif ($marks>=45 && $marks<60) {
    print "Well done you have secured a Second Class \n"; }
elseif ($marks>=35 && $marks<45) {
    print "You have secured a Pass Class \n"; }
else {
    print "Sorry, you have failed \n"; }

```



**5. A foreach loop for printing the lowest number in a set:**

```
@numbers = (78, 87, 34, 45, 23, 64, 12, 32, 21, 54, 43);
$num = 0;
foreach $n1 (@numbers) {
    if ($n1 > $num) {
        $num = $n1;    }
}
foreach $n2 (@numbers) {
    if ($n2 < $num) {
        $num = $n2;    }
}
print "The lowest number is $num";
```

## 16. PERL FUNCTIONS

**1. A program to remove the last character from a string variable using chop() function and store it in the variable. Then print the value of original and new variables:**

```
$line = "Keep Your City Clean.";
$chop_line = chop($line);
print "$line \n";
print "$chop_line\n";
```

**2. A program using chomp() function to remove the newline character:**

```
$line = "Keep Your City Clean.\n";
chomp($line);
print "$line";
```

**3. A PERL program to concatenate two strings with space between them and append the third string using the concatenation operator storing the result in the same string:**

```
$firstname = "Jodie";
$secondname = "Foster";
# Concatenate the two strings
$fullname = $firstname . " " . $secondname;
# Concatenation using concatenation operator
$str = " is blonde.";
$fullname .= $str;
print "$fullname";
```

**4. Using the substr function to extract first five characters of a string and then printing those five characters:**

```
# Define a string
$line = "Play Light Music";
# Extract the part that follows "Play"
$extract = substr($line,0,5);
print "$extract\n";
```

5. A program to print the total length of a given string and then extract a string and count its length:

```
$line = "Jodie Foster is blonde.";
$count = length($line);
print "Length of the string \: $count\n";
$pos = index($line, "Foster");
$part = substr($line, $pos, 6);
print "Extracted string \: $part\n";
$len = length("$part");
print "Length of the extracted string \: $len";
```

6. A program to add elements to the beginning of the array and sort it:

```
@games = ("Golf", "Snooker", "Cricket", "Hockey");
unshift(@games, "Soccer", "Kabaddi");
print "@games\n ";
@sorted_names = sort(@games);
print "@sorted_names";
```

## 17. FILEHANDLING

1. A PERL program to open the file test-file1.txt and display appropriate message on success or failure:

```
if (open(TEST1, "test-file1.txt")) {
    print ("You are Successful in opening file");    }
else {
    print ("Cannot find file");    }
```

2. A PERL program to write to the text file:

```
open(TEST1, ">test-file1.txt") || die("cannot open file");
print TEST1 "This is good progress\n";
```

3. A PERL program to create a file test-file2.txt and then write and append to it accepting input from the user:

```
open(TEST2, ">>test-file2.txt");
$command = <STDIN>; {
    print TEST2 "$command";    }
```

4. A PERL program to create a directory called "Student\_Dir":

```
mkdir("Student_Dir", 840);
```

## 18. REGULAR EXPRESSIONS

1. A program, which will take in a string and check for the pattern 'Hello'. If found display an appropriate message for Uppercase, Lowercase or Proper case and also an appropriate message when match not found:

```
print "Enter a String containing hello:\n";
$str = <stdin>;
if ($str =~ /Hello/) {
    print "ProperCase Match! \n"; }
```

```

elsif ($str =~ /hello/) {
    print "LowerCase Match!\n"; }
elsif ($str =~ /HELLO/) {
    print "UpperCase Match!\n"; }
else {
    print "Hello not encountered or Mixed case match"; }

```

2. A program which defines a variable containing a string and matches a pattern to print an appropriate message:

```

$str = "Hello World";
if ($str =~ /Hello World/) {
    print "Case Match\n"; }
if ($str =~ /hello World/i) {
    print "Case Insensitive Match"; }

```

## 19. CREATING STRUCTURED PROGRAMS

1. Creating a subroutine, which will print the greeting and then invoking it:

```

sub message {
    print "Good Morning India! \n"; }
&message;

```

2. Creating a subroutine, which a name can be passed as an argument. The subroutine then prints a greeting:

```

sub message {
    print "Good Morning @_ \n"; }
&message("Roger");

```

3. Create a subroutine to calculate the tax on income @ 30% of income. Pass the income as an argument to the subroutine. The subroutine should then return the tax liability on that income.

```

$tax=0;
$income=5000;
$tax = &computetax($income);
print "Your tax liability is : " . $tax . "\n";
sub computetax {
    my($arg) = @_;
    $tax = ($arg * 0.30);
    return ($tax);
}

```

## 20. GOING THE OBJECT WAY WITH PERL

1. A program defines a scalar variable with a value and then creates a reference to scalar variable to print the contents of the reference:

```

#!c:/perl/bin/perl.exe -w
my $scalarVar = "Good Morning";
my $scalarRef = \$scalarVar;
print " Var : $scalarVar \n";
print "Ref : $$scalarRef \n";

```

2. A program to create the array of days of the week and then create a reference to this array and print out the contents of the array reference:

```
#!/c:/perl/bin/perl.exe -w
@weeks = qw (Sun Mon Tue Wed Thu Fri Sat);
$arrayRef = \@weeks;
for $month (@$arrayRef) {
    print " Month : $month \n"; }

```

3. A program to create an associative array and a reference the array to print its:

```
#!/c:/perl/bin/perl.exe -w
%who = ('Name' => 'John', 'Age' => 25, 'Height' => '182 cm', 'Weight' => '80 kg');
$hashRef = \%who;
for $key (sort keys %$hashRef) {
    $value = $hashRef->{$key};
    print " Key : $key, Value : $value \n"; }

```

4. Create a subroutine. Create a reference to that subroutine and then de-reference the reference to call the subroutine:

```
#!/c:/perl/bin/perl.exe -w
sub callBack {
    my ($mesg) = @_;
    print "$mesg\n";
}
$codeRef = \&callBack;
&$codeRef ("Hi, Mike!");
&$codeRef ("How are you?");

```

5. Create a package with a subroutine:

```
package pack1;
sub callback {
    my ($mesg) = @_;
    print "$mesg \n";
}
1;

```

Call a subroutine of that package in a program:

```
#!/c:/perl/bin/perl.exe -w
require "mylib.pl";
$var = "Good Morning India!";
pack1::callBack($var);

```

## *SECTION - V: Appendix*

### APPENDIX - A

#### HTML Color Codes

Color	RRGGBB	Color	RRGGBB	Color	RRGGBB
White	FFFFFF	Red	FF0000	Green	00FF00
Blue	0000FF	Magenta	FF00FF	Cyan	00FFFF
Yellow	FFFF00	Black	000000	Aliceblue	F0F8FF
Aquamarine	70DB93	Baker's chocolate	5C3317	Blue violet	9F5F9F
Brass	B5A642	Bright gold	D9D919	Brown	A62AA2
Bronze	8C7853	Bronze II	A67D3D	Cadet blue	5F9F9F
Cool bopper	D98719	Copper	B87333	Coral	FF7F00
Cornflower blue	42426F	Dark brown	5C4033	Dark green	2F4F2F
Dark green copper	4A766E	Dark olive green	4F4F2F	Dark orchid	9932CD
Dark purple	871F78	Dark slate blue	6B238E	Dark slate grey	2F4F4F
Dark tan	97694F	Dark turquoise	7093DB	Dark wood	855E42
Dim grey	545454	Dusty rose	856363	Feldspar	D19275
Firebrick	8E2323	Forest green	238E23	Gold	CD7F32
Goldenrod	DBDB70	Gray	C0C0C0	Green copper	527F76
Green yellow	93DB70	Hunter green	215E21	Indian red	4E2F2F
Khaki	9F9F5F	Light blue	C0D9D9	Light gray	A8A8A8
Light steel blue	8F8FBD	Light wood	E9C2A6	Light green	32CD32
Mandarin orange	E47833	Maroon	8E236B	Medium aquamarine	32CD99
Medium blue	3232CD	Medium forest green	6B8E23	Medium goldenrod	EAEAAE
Medium orchid	9370DB	Medium sea green	426F42	Medium slate blue	7F00FF
Medium turquoise	70DBDB	Medium violet red	DB7093	Medium wood	A68064
Midnight blue	2F2F4F	Navy blue	23238E	Neon blue	4D4DFF
Neon pink	FF6EC7	New midnight blue	00009C	New tan	EBC79E
Old gold	CFB53B	Orange	FF7F00	Orange red	FF2400
Orchid	DB70DB	Pale green	8FBC8F	Pink	BC8F8F
Plum	EAAD6A	Quartz	D9D9F3	Rich blue	5959AB
Salmon	6F4242	Scarlet	8C1717	Sea green	238E68
Semi sweet chocolate	6B4226	Sienna	8E6B23	Silver	E6E8FA
Sky blue	3299CC	Slate blue	007FFF	Spicy pink	FF1CAE
Spring green	00FF7F	Steel blue	236B8E	Summer sky	38B0DE
Tan	DB9370	Thistle	D8BFD8	Turquoise	ADEAEA
Very dark brown	5C4033	Very light gray	CDCDCD	Violet	4F2F4F
Violet red	CC3299	Wheat	D8D8BF	Yellow green	99CC32

#### Resource On World Wide Web

The world wide web (w3) conitorium	<a href="http://www.w3.org/">http://www.w3.org/</a>
Home to general information on the World Wide Web and links to all areas of WWW and HTML.	

<b>The WWW Section of Yahoo!</b>	<a href="http://www.yahoo.com/Computers/World Wide Web">http://www.yahoo.com/Computers/World Wide Web</a>
Links to hundreds of pages regarding WWW and HTML.	

<b>OpenText</b>	<a href="http://www.opentext.com/">http://www.opentext.com/</a>
An extensive database of Web sites.	

### Resources On HTML Editors

<b>FrontPage</b>	<a href="http://www.microsoft.com/msoffice/frontpage/default.html">http://www.microsoft.com/msoffice/frontpage/default.html</a>
Microsoft's Web page and site development application.	

<b>GNNPress</b>	<a href="http://www.tools.gnn.com/press/index.html">http://www.tools.gnn.com/press/index.html</a>
A multiplatform Web page design tool.	

<b>HomeSite</b>	<a href="http://www.dexnet.com/homesite.html">http://www.dexnet.com/homesite.html</a>
A full-featured shareware Web page designer.	

<b>HTML Assistant Pro</b>	<a href="http://www.brooknorht.com/istar.html">http://www.brooknorht.com/istar.html</a>
A commercial HTML editor based on popular shareware editor.	

<b>HTML Writer</b>	<a href="http://www.lal.cs.byu.edu/people/nosack/index.html">http://www.lal.cs.byu.edu/people/nosack/index.html</a>
An HTML editor that allows to edit more than one document at a time.	

<b>InContext Spider</b>	<a href="http://www.incontext.ca/products/spider1.html">http://www.incontext.ca/products/spider1.html</a>
A commercial Web page tool that supports advanced HTML features.	

<b>Symposia</b>	<a href="http://www.grif.fr/prod/sympro.html">http://www.grif.fr/prod/sympro.html</a>
A web design application for UNIX and Windows.	

<b>Webber</b>	<a href="http://www.csdcorp.com/webber.htm">http://www.csdcorp.com/webber.htm</a>
An HTML editor that includes a validator program to look for invalid HTML markup.	

<b>WebMedia Publisher</b>	<a href="http://www.wbmedia.com/publisher/">http://www.wbmedia.com/publisher/</a>
A web page tool that supports the latest Microsoft and Netscape HTML extensions.	

<b>Web Wizard</b>	<a href="http://www.halcyon.com/webwizard/index.html">http://www.halcyon.com/webwizard/index.html</a>
An HTML editor available in 16-bit and 32-bit versions	

### Resources On HTML Document Development

<b>HTML 2.0 Specification</b>	<a href="http://www.w3.org/hypertext/www/MarkUp/html-spec/index.html">http://www.w3.org/hypertext/www/MarkUp/html-spec/index.html</a>
A full description of the final draft of HTML 2.	

<b>HTML 3.2 Draft Specification</b>	<a href="http://www.w3.org/pub/www/MarkUp/">http://www.w3.org/pub/www/MarkUp/</a>
A full description of the current draft of HTML 3.2.	

<b>HTML Style Guide</b>	<a href="http://www.w3.org/hypertext/www/Provider/Style/Overview.html">http://www.w3.org/hypertext/www/Provider/Style/Overview.html</a>
Tim Berners-Lee's excellent guide to designing WWW documents.	

<b>NetSpace Guide for HTML Developers</b>	<a href="http://netspace.org/netspace/wwwdoc.html">http://netspace.org/netspace/wwwdoc.html</a>
Lots of links to documents on creating effective HTML documents.	

<b>URL Descriptions</b>	<a href="http://www.w3.org/hypertext/www/Addressing/Addressing.html">http://www.w3.org/hypertext/www/Addressing/Addressing.html</a>
Definitions of the various types of URLs, as well as discussion of URNs and URIs.	

<b>WWW Development Page (Virtual Library)</b>	<a href="http://www.charm.net/web/Vlib/">http://www.charm.net/web/Vlib/</a>
Links to a wide range of HTML development documents.	
<b>Net Tips for Writers and Designers</b>	<a href="http://www.dsiegel.com/tips/">http://www.dsiegel.com/tips/</a>
Design tips for effective Web pages from a leading designer.	
<b>Frames Tutorial</b>	<a href="http://www.newbie.net/frames/">http://www.newbie.net/frames/</a>
A step by step tutorial for creating pages using frames.	

## APPENDIX - B

### Resources On JavaScript

<b>Netscape JavaScript Information</b>	<a href="http://home.netscape.com/eng/mozilla/Gold/handbook/javascript/">http://home.netscape.com/eng/mozilla/Gold/handbook/javascript/</a>
JavaScript information from the company that brought it to the Web.	
<b>Gamelan JavaScript Scripts</b>	<a href="http://www.gamelan.com/">http://www.gamelan.com/</a>
A repository of many JavaScript scripts.	
<b>JavaScript Scripts</b>	<a href="http://www.geocities.com/SiliconValley/9000/">http://www.geocities.com/SiliconValley/9000/</a>
A set of useful JavaScript scripts by the same set of authors.	
<b>JavaScript Tip of the Week</b>	<a href="http://www.webreference.com/javascript">http://www.webreference.com/javascript</a>
Numerous tips and tricks on using JavaScript.	
<b>Babyak's JavaScript Resources</b>	<a href="http://www.epix.com/~mbabyak/">http://www.epix.com/~mbabyak/</a>
Links to JavaScript resources across the Web	
<b>JavaScript Overview</b>	<a href="http://home.netscape.com/comprod/products/navigator/version2.0/script/index.html">http://home.netscape.com/comprod/products/navigator/version2.0/script/index.html</a>
<b>JavaScript Documentation</b>	<a href="http://home.netscape.com/eng/mozilla/Gold/handbook/javascript/index.html">http://home.netscape.com/eng/mozilla/Gold/handbook/javascript/index.html</a>
<b>JavaScript Index</b>	<a href="http://www.c2.org/~andreww/javascript/">http://www.c2.org/~andreww/javascript/</a>
<b>JavaScript Library</b>	<a href="http://www.c2.org/~andreww/javascript/lib/">http://www.c2.org/~andreww/javascript/lib/</a>
<b>Learning JavaScript with Windows Help</b>	<a href="http://www.webconn.com/javahelp/javahelp.htm">http://www.webconn.com/javahelp/javahelp.htm</a>

## APPENDIX C

### HTTP

HTTP replies and requests both consist of a header and a body. Scripts that want to return HTTP directly to a client must create their own header. This header starts on a line distinguishing it as HTTP and expresses the status of the HTTP request. This line has the form:

**HTTP/version status\_code message**

The message is often OK or an error message. This error message can be used by the client to tell the user what happened. The status code is an integer between 200 and 599. It should be one of the status code listed in the HTTP specification. The valid codes are listed in the table below.

Code	Meaning
200	The request was successful
201	The POST request was successful

Code	Meaning
202	The request was received but the result was unknown
203	The GET request was accepted, but only partially fulfilled
204	The request was successful, there is no body information to update the client
300	The request can be provided from multiple locations at the client's choice, the location fields should be used to make this choice
301	The requested resource has moved and can be found at the URL in the location: field of the header. The browser should retrieve the new URL automatically
302	The requested resource is not at the specified location and can be found at the URL in the location: field of the header. The browser should retrieve the new URL automatically
304	The resource requested with a GET request and an If-Modified-Since field is not modified and will not be returned
400	The request has a wrong syntax
401	The requested resource requires authentication. The header should contain WWW-authenticated fields to allow the user to negotiate with the server for authentication
402	The requested resource has a cost and the client did not send a valid Charge-to:field in the request header
403	The requested resource is forbidden
404	The requested resource could not be found
405	The requested resource does not support the type of request made
406	The requested resource is not one of the client's accepted types or encoding
410	The requested resource was available, but is not any longer
500	There was a server error
501	The sever does not support the type of request that was made
502	The request required that the server retrieve information from another server and this retrieval
503	The requested service is not available at this time
504	Same as 502, but the retrieval timed out instead of failed

After the first line, the header can contain a number of lines, including the content type of the HTTP body. Following the header is a blank line and the body, HTML for a reply with the content type text/HTML. A list of the standard HTTP response header fields is provided in the table below.

Field	Meaning
Allow:method_list	A comma-delimited list of the HTTP request methods supported by the requested resource. These methods can be any of GET, HEAD, POST, PUT, DELETE, LINK and UNLINK. As one would expect, CGI scripts usually allow GET and/or POST
Content-Encoding:encoding	The encoding used on the message body. Currently this can be either compress or gzip. Only one of these fields is allowed. If supported by the browser, this allows data to be compressed during transfer and automatically decompressed by the browser without the user knowing
Content-length:length	The length in bytes of the message
Content-Transfer-Encoding:type	The encoding method used by the method, this is uncommon in HTTP requests, but is used in MIME
Content-type:general/specific	The MIME type for the message, often text/HTML
Date:date	The date and time that the message is sent. The format for this field is week day, day month year hours:minutes:seconds time zone. The time zone should be Greenwich Mean Time(GMT) for compatibility. For example, this field could be WED, 01 Apr 1995 13:13:13 GMT
Derived-From:version	The version of the information from which the resource came
Expires:date	The date and time that this reply should become invalid. Clients should use this information to refresh a page if necessary



Field	Meaning
Forwarded:by url for domain	Used by proxy Web servers to tell the client that a proxy was used. If multiple proxies are used, this field will be present multiple times
Last-modified:date	The date and time when this resource was last modified. This value should be in GMT
Link:thelink	Similar to the HTML link tag
Location:url	The URL for a resource that the Web server should return instead of this one.
MIME-version:version	The version of the MIME protocol that is supported
Public:methods	A list of non standard methods supported by this resource
Retry-after:date	If a resource is unavailable, the status code 503 should be returned and this field should have the date and time or number of seconds that the client should wait before retrying
Server:app/version	The Web server application name and version
Title:title	The title of the resource
URL:url	The Uniform Resource Identifier for a resource that should be returned instead of the requested one. This field is replacing Location:
Version:version	The version of the resource itself
WWW-Authentication:scheme message	This field is used to support user authentication

## PERL Error Messages

There are seven classifications of error messages that Perl can generate. These classifications are listed below.

Error Class	Meaning
W	An optional warning
D	An optional deprecation
S	A mandatory severe warning
F	A fatal error

Error Class	Meaning
P	An internal error
X	A very fatal error
A	An alien error message indicating an error not generated by Perl

The string \*\*\*\*\* in the error messages below means that an object from the code in question will replace \*\*\*\*\*, while the string NNNNN in the error message means that a numeric object from the code in question will replace NNNNN.

The following abbreviations are also used in the messages below:

- SV Scalar variable
- HV Hash variable
- AV Array variable

The following table displays the various Error messages with the type of Error class they belong to and what these error messages mean.

Error Message	Meaning
"my" variable ***** can't be in a package (F)	Occurs when one tries to declare a lexically scoped variable within a package. To localize a package variable, use local.
"no" not allowed in expression (F)	Perl displays this message when the no keyword is recognized by the compiler but does not return a meaningful value.
"use" not allowed in expression (F)	Perl displays this message when the use keyword is recognized by the compiler but does not return a meaningful value.

Error Message	Meaning
**** argument is not a HASH element (F)	Perl displays this error message when the argument is not a hash element.
**** did not return a true value (F)	This appears when the file loaded with the require or use doesn't load properly.
**** found where operator expected (S)	Occurs when the Perl lexer expects an operator but gets something else instead.
**** had compilation errors (F)	This message is displayed as the summary message when Perl -c fails.
**** had too many errors (F)	Issued when the parser gives up trying to parse a program. This occurs after ten tries.
**** matches null string many times (W)	Issued when the pattern one tries to match ends up in an infinite loop.
**** syntax OK (F)	Appears as a summary message when Perl -c is successful
500 Server Error	Indicates a CGI error and not a PERL error.
?+* follows nothing in regexp (F)	Perl issues this when a regular expression is used with a Quantifier.
Ambiguous use of **** resolved as **** (W) (S)	Issued when Perl interprets an expression in a way one might not have intended.
Argument **** isn't numeric (W)	Perl issues this message when it expected a numeric argument to an operator but found a string instead.
Array @**** missing in the @ in argument **** of ****() (P)	This message is displayed when one leaves the @ off of an array variable name.
Assignment to both a list and scalar (F)	Perl displays this message when the second and third arguments to a conditional operator are either both lists or both scalars.
Attempt to free unreferenced scalar (W)	This message occurs when Perl tries to decrement the reference count of a scalar and finds the reference count is already at 0.
Bad associative array (P)	This message is displayed when a null HV pointer is passed to one of Perl's internal hash routines.
Bad filehandle:**** (F)	Perl issues this message when a symbol was passed to something that wanted a filehandle.
Bad name after**** : (F)	This message is displayed when a symbol that is named by a package prefix isn't finished
Bad symbol for array (P)	Perl displays this message this message when an internal request tried to add an array entry to an object that wasn't a symbol table entry.
Bad symbol for a filehandle (P)	Perl displays this error message when an internal request tried to add a filehandle entry to an object that wasn't a symbol table entry.
Bad symbol for hash (P)	Perl displays this error when an internal request tried to add a hash entry to an object that wasn't a symbol table entry.
Callback called exit (F)	This message is displayed when a subroutine invoked from an external package exited by calling the exit function
Can't "last" outside a block (F)	This message is caused by trying to break out of the current block by issuing the last statement outside of the current block.
Can't "next" outside a block (F)	This message is caused by trying to reiterate the current block with the next statement outside the current block.
Can't break at that line (S)	This debugger warning message indicates that the line number specified contains a statement that can't be stopped.
Can't call method "*****" in empty package "*****" (F)	This message is displayed when one tries to call a method in a package that doesn't have anything in it.
Can't call method "*****" on unblesed reference (F)	This message is displayed when one tries to call a method without supplying an object reference.
Can't call method "*****" without a package or object reference (F)	This message could be displayed when trying to call a method by supplying an expression that doesn't evaluate to a package or an object reference.

Error Message	Meaning
Can't chdir to ***** (F)	Perl displays this message when one specifies a directory that can not be got with the chdir command.
Can't coerce ***** to integer in ***** (F)	This message will be displayed when one tries to force certain scalar values to be an integer.
Can't coerce ***** to number in ***** (F)	This message will be displayed when one tries to force certain scalar values to be a number.
Can't do inplace edit on *****:***** (S)	This message is displayed when creation of a new file fails because it can't be edited in place.
Can't emulate -***** On #! Line (F)	This message is displayed when the #! Line specifies a switch that doesn't make sense in the current context.
Can't localize a reference (F)	This message is displayed when one tries to make a reference variable local
Can't locate object method "*****" via package "*****" (F)	Perl displays this message when one tries to call a method that doesn't exist in the package.
Can't modify ***** in *****	Perl displays this message when one tries to make an assignment to the item *****.
Can't modify non-existent substring (P)	This message is displayed when Perl's internal routine that does assignment to a substr was handed a null pointer.
Can't take log of NNNNN (F)	This message is displayed when one tries to take the logarithm of a number that is either not positive, or real, or both.
Can't take sqrt of NNNNN (F)	This message is displayed when one tries to take the square root of a negative number.
Can't undef active subroutine (F)	Perl displays this message when one tries to undefine a subroutine that is currently running.
Can't unshift (F)	This message occurs when trying to unshift on array that can't be unshifted
Can't use "my*****" in sort comparison (F)	Perl displays this message when the variables \$a and \$b are used in a comparison after they have been declared as lexical variables with my.
Can't use ***** for loop variable (F)	This message occurs when trying to use something other than a simple scalar as a loop variable on a foreach.
Can't use ***** ref as ***** ref (F)	This message is displayed when one mixes up the reference types.
Can't use \l to mean \$l in expression (W)	This message is displayed when one tries to use the \l back reference to a matched substring outside a regular expression.
Can't use global ***** in "my" (F)	This message is displayed when one tries to declare a special variable as a lexical variable.
Close on unopened file ***** (W)	Perl issues this message when trying to close a file that was never opened.
Deep recursion on subroutine ***** (W)	This message is displayed when a subroutine calls itself more than 100 times more than it has returned.
Did you mean &***** instead? (W)	This message is displayed when one refers to a subroutine with a dollar sign, as in \$sub when one meant &sub.
Did you mean \$ or @ instead of %? (W)	Perl issues this message when referring to an individual hash item with %, as in %hash{\$key}.
Do you need to predeclare*****? (S)	This message occurs when a subroutine or module name is referenced when it hasn't yet been declared.
Elsif should be elsif (S)	This message occurs when one uses Elsif instead of elsif
Filehandle ***** never opened (W)	This message occurs when I/O operation is attempted on a filehandle that was never initialized.
Filehandle ***** only opened for input (W)	This message occurs when one attempts to write to a read-only filehandle.
Final \$ should be \ \$ or \$name (F)	Perl displays this message when a \$ comes at the end of a string and Perl cannot decide if what to follow should be a reference or a scalar.

Error Message	Meaning
Final @ should be \@ or @name (F)	Perl displays this message when a \$ comes at the end of a string and Perl cannot decide if what to follow should be an array or an array name.
Found = in conditional, should be == (W)	This message is displayed when one assigns and meant test for equality.
Illegal division by zero (F)	Perl displays this message when trying to divide a number by 0.
Illegal modulus zero (F)	Perl displays this message when trying to divide a number by 0 to get the remainder.
Illegal octal digit (F)	This message occurs when one uses an 8 or 9 in an octal number.
Illegal octal digit ignored (W)	This warning message occurs when Perl finds an 8 or 9 in an octal number and interpretation of the number stops before 8 or 9.
Internal disaster in regexp (P)	This message occurs when something goes wrong with the regular expression parser.
Junk on the end of regexp (P)	This message occurs when the regular expression parser is mixed up.
Label not found for "last*****" (F)	This message is displayed when trying to break out of a loop by name but are not currently in the named loop.
Label not found for "next*****" (F)	This message is displayed when trying to continue in a loop one is not currently in.
Label not found for "redo*****" (F)	This message is displayed when trying to restart a loop one is not currently in.
Missing \$ on loop variable (F)	This message occurs when a loop variable is introduced without a \$.
Missing right bracket (F)	This message appears when the Perl lexer counts more opening curly braces than closing curly braces.
Modification on read-only value attempted (F)	Perl issues this message when trying to modify a constant.
Not a GLOB reference (F)	This message is issued because Perl was looking for a reference to a typeglob but got something else instead
Not a HASH reference (F)	This message is issued because Perl was looking for a reference to a hash but got something else instead.
oops:oopsAV (S)	An internal Perl warning that indicates the grammar is messed up.
panic:ck_split (P)	This message is displayed when an internal consistency check fails while trying to compile a split.
panic:do_split (P)	This message occurs when a split is not set up properly.
panic:pp_iter (P)	Perl issues this message when the foreach iterator is called in a nonloop context.
Parens missing around "*****" list (W)	This warning indicates to do assignment with parentheses instead of without them.
Precedence problem: open ***** should be open(*****) (S)	This message indicates that one has used open...   die, when open(...)    die should have been used.
print on closed filehandle***** (W)	This warning indicates that one has tried to print on a closed filehandle.
printf on closed filehandle***** (W)	This warning indicates that one has tried to print on a closed filehandle.
Recompile Perl with -DDEBUGGING to use -D switch	This message indicates that one needs to recompile Perl with -DDEBUGGING inorder to use the -D switch in the code.
Semicolon seems to be missing (W)	This warning is issued when a syntax error was caused by leaving off a semicolon.
Send on closed socket (W)	This warning is issued when trying to send a filehandle that is closed.
Sequence (?#... not terminated (F)	Perl generates this error when a regular expression comment isn't closed with a parenthesis.
Sever error	Also known as "500 Server error", this is a CGI error and not a Perl error.

Error Message	Meaning
substr outside of string (W)	This warning is issued when trying to reference a substr that is outside of a string.
Syswrite on closed filehandle (W)	This warning is generated when the filehandle one is trying to write is closed.
Undefined sort subroutine "*****" called (F)	Perl generates this error when it can't find the sort subroutine called.
Undefined subroutine &***** called (F)	Perl generates this error when it can't find the subroutine called because the subroutine either wasn't defined initially or was undefined.
Unrecognized switch: -***** (F)	This error is issued when one gives Perl an illegal option switch.
Use of implicit split to @_ is deprecated (D)	This message is displayed when one should have explicitly assigned the results of a split to an array.
Warning: use of "*****" without parens is ambiguous (S)	This message indicates that one has forgotten to enclose an argument to a function in parentheses.

### Resources On CGI And PERL

<b>CGI Documentation</b>	<a href="http://www.yahoo.ncsa.uiuc.edu/cgi/">http://www.yahoo.ncsa.uiuc.edu/cgi/</a>
Information on the Common Gateway Interface, Including CGI scripts in the programs.	
<b>CGI Programs (Perl)</b>	<a href="http://www.seas.upenn.edu/~mengwong/perlhtml.html">http://www.seas.upenn.edu/~mengwong/perlhtml.html</a>
A library of commonly used CGI routines, written in Perl.	
<b>Specification for CGI</b>	<a href="http://www.yahoo.ncsa.uiuc.edu/cgi/">http://www.yahoo.ncsa.uiuc.edu/cgi/</a>
The NCSA site also provides a set of tutorials and a page with numerous links to HTTP-specific specifications. These pages are available at; <a href="http://www.yahoo.ncsa.uiuc.edu/docs/tutorials/">http://www.yahoo.ncsa.uiuc.edu/docs/tutorials/</a> and <a href="http://www.yahoo.ncsa.uiuc.edu/docs/Library.html">http://www.yahoo.ncsa.uiuc.edu/docs/Library.html</a>	
<b>CGI Examples</b>	<a href="http://www.eff.org/~erict/Scripts/">http://www.eff.org/~erict/Scripts/</a>
<b>Perl Home Page</b>	<a href="http://www.perl.com">http://www.perl.com</a>
<b>Win32 Distribution of Perl</b>	<a href="http://www.activestate.com">http://www.activestate.com</a>



## INDEX

\$

\$\_ ..... 264

<

< and > Commands ..... 312  
<DIV>...</DIV> ..... 215  
<ILAYER>...</ILAYER> ..... 219  
<LAYER>...</LAYER> ..... 217  
<LINK> ..... 213  
<SPAN>...</SPAN> ..... 212  
<STYLE>...</STYLE> ..... 205  
    Type=text/css ..... 205

A

a and A Commands ..... 311  
Action ..... 158  
Action attribute ..... 159  
ALIGN ..... 24, 37, 44, 45  
ALIGN = BOTTOM ..... 37  
ALIGN = CENTER ..... 37  
ALIGN = LEFT ..... 37  
ALIGN = MIDDLE ..... 37  
ALIGN = RIGHT ..... 37  
ALIGN = TOP ..... 37  
ALINK ..... 53  
ALT ..... 37  
ALT attribute ..... 39  
anchors ..... 54  
Anchors ..... 272  
Apache 2 ..... 315  
    .htaccess ..... 321  
    httpd.conf ..... 319  
        <Directory> ..... 321  
        <Files> ..... 321  
        <FilesMatch> ..... 321  
        Alias ..... 321  
        CustomLog ..... 320, 321  
        DirectoryIndex ..... 321  
        DocumentRoot ..... 320, 321  
        ErrorLog ..... 320, 321  
        KeepAlive ..... 320  
        KeepAliveTimeout ..... 320  
        LoadModule ..... 323

MaxKeepAliveRequests .....	320
NameVirtualHost .....	322
PidFile .....	320
ScoreBoardFile .....	320
ScriptAlias .....	321
ServerAdmin .....	320
ServerName .....	320, 321
ServerRoot .....	320
Timeout .....	320
TransferLog .....	321
VirtualHost .....	321
Installation .....	316
mine.types .....	319
<i>Apache Modules</i>	
apxs .....	323, 412
<i>arithmetic operations</i> .....	244
<i>Arithmetic Operators</i> .....	127
<i>ARPAnet</i> .....	1
<i>Array</i> .....	239, 258, 260, 261
<i>array methods</i> .....	125
<i>arrays</i> .....	258
<i>Arrays</i> .....	124
<i>arrow operator</i> .....	291
<i>Assignment Operators</i> .....	128
<i>Associative Arrays</i> .....	240
<i>asterisk (*)</i> .....	275
<i>Auto-Decrement</i> .....	244
<i>Auto-Increment</i> .....	244
<b>B</b>	
<i>b Command</i> .....	310
<i>BACKGROUND</i> .....	20
<i>Backward Pattern Search (??) Command</i> .....	306
<i>BGCOLÖR</i> .....	20
<i>BGCOLOR attribute</i> .....	47
<i>Binary files</i> .....	268
<i>binding operators</i> .....	276
<i>BODY</i> .....	35
<i>Bold</i> .....	25
<i>BORDER</i> .....	37, 44
<i>BORDER attribute</i> .....	37
<i>button</i> .....	159
<i>button element</i> .....	162
<b>C</b>	
<i>c Command</i> .....	310
<i>Caption</i> .....	44
<i>caret (^)</i> .....	272
<i>Cascading Style Sheets (CSSs)</i> .....	205



<i>CELLPADDING</i> .....	44
<i>CELLPADDING</i> attribute.....	46
<i>CELLSPACING</i> .....	44
<i>CELLSPACING</i> attribute.....	46
<i>Centering</i> .....	27
<i>CGI</i> .....	228
<i>CGI</i> object.....	294
<i>CgiDie</i> .....	286
<i>CgiError</i> .....	286
<i>cgi-lib.pl</i> .....	285
<i>Character</i> .....	272
<i>checkbox</i> .....	166
<i>checkBox</i> .....	159
<i>chomp</i> .....	See
<i>chop</i> .....	254
<i>class</i> .....	293
<i>Client / Server</i> .....	12
<i>Client IP Address</i> .....	6
<i>Close</i> .....	267
<i>Close Method</i> .....	299
<i>COLOR</i> .....	28
<i>COLSPAN</i> .....	44
<i>COLSPAN</i> and <i>ROWSPAN</i> attributes.....	47
<i>communicating on the Internet</i> .....	2
<i>Comparison Operators</i> .....	128, 245
<i>concatenate</i> .....	255
<i>Conditional statements</i> .....	245
<i>Connect String</i> .....	296
<i>Connection Method</i> .....	297
<i>CONTENT_LENGTH</i> .....	231
<i>cookies</i> .....	184

**D**

<i>d</i> and <i>D</i> Commands.....	311
<i>data connection</i> .....	296
<i>Data Method</i> .....	298
<i>Data Rows</i> .....	44
<i>data storage system</i> .....	296
<i>DataHash Method</i> .....	298
<i>Date object</i> .....	175
<i>DD</i> .....	33
<i>debugging</i> .....	303
<i>Declaring functions</i> .....	133
<i>Defining a Reference</i> .....	290
<i>Definition description</i> .....	33
<i>Definition Lists</i> .....	33
<i>Definition term</i> .....	33
<i>dense array</i> .....	125
<i>dereferencing</i> .....	290
<i>directories</i> .....	266, 269, 270

<i>DIRHANDLE</i> .....	269, 270
<i>DL</i> .....	33
<i>Document Body</i> .....	20
<i>Document Head</i> .....	19
<i>Document Object Model(DOM)</i> .....	143
<i>dollar sign (\$)</i> .....	272
<i>Domain Name Extension</i> .....	5
<i>dot</i> .....	274
<i>Double-quoted strings</i> .....	236
<i>Drawing Lines</i> .....	24
<i>DSN</i> .....	296
<i>Dynamic Shared Object (DSO)</i> .....	323
<b>E</b>	
<i>each function</i> .....	242
<i>Environment Variables</i> .....	230
<i>Establishing connectivity on the internet</i> .....	5
<i>eval() function</i> .....	132
<i>Event handlers</i> .....	150
<i>excludes</i> .....	274
<i>Expressions</i> .....	126
<b>F</b>	
<i>FetchRow Method</i> .....	298
<i>filehandle</i> .....	264
<i>Filehandle</i> .....	282
<i>FILEHANDLE</i> .....	266, 267, 268, 269
<i>Files</i> .....	266
<i>FONT</i> .....	33, 40, 48, 62, 63, 64, 73, 74, 75, 76
<i>Font Attributes</i> .....	206
<i>FONTFACE</i> .....	28
<i>Footer</i> .....	21
<i>for Loop</i> .....	250
<i>FOR loop</i> .....	131
<i>foreach Loop</i> .....	251
<i>Forms</i> .....	153
<i>Frames</i> .....	68
<i>FRAMESET</i> .....	68
<i>FTP</i> .....	10
<i>functions</i> .....	254, 258, 261, 262. See
<i>Functions</i> .....	132
<b>G</b>	
<i>gateway</i> .....	228
<i>Get</i> .....	158
<b>H</b>	
<i>H Command</i> .....	313
<i>Header Rows</i> .....	44

IDX	INDEX	PAGE v
<i>HEIGHT</i> .....		37
<i>hidden object</i> .....		159
<i>Hierarchy</i> .....		144
<i>HREF</i> .....		53
<i>HSPACE</i> .....		37
<i>HTML</i> .....		15, 35
<i>HTML Command</i>		
GET.....		229
POST.....		229
<i>HtmlBot</i> .....		285
<i>HtmlTop</i> .....		285
<i>HTTP</i> .....		14
<i>HTTP</i> .....		229
<i>Hyperlinks</i> .....		53
<b>I</b>		
<i>If - then - else</i> .....		130
<i>if statement</i> .....		245
<i>Image maps</i> .....		58
<i>IMG</i> .....		37
<i>Immediate if (Conditional expression)</i> .....		131
<i>Index</i> .....		257
<i>Indexed Arrays</i> .....		239
<i>inheritance</i> .....		294
<i>Initializing Local Variables</i> .....		280
<i>Instance</i> .....		144, 178
<i>Instance Hierarchy</i> .....		143
<i>Instantiation</i> .....		178
<i>Interactive event handlers</i> .....		150
<i>Internet</i> .....		1
<i>Internet Address</i> .....		8
<i>Internet Domains</i> .....		4
<i>INTERNET PROTOCOL</i> .....		9
<i>InterNIC</i> .....		2
<i>Interpreted language</i> .....		119
<i>IP Addressing</i> .....		7
<i>ISP</i> .....		1, 6
<i>Italics</i> .....		25
<b>J</b>		
<i>JSSS DOM</i> .....		144
<b>K</b>		
<i>keys function</i> .....		241
<b>L</b>		
<i>l Command</i> .....		304
<i>L Command</i> .....		310
<i>Layers</i>		
Z-INDEX.....		218

<i>Length</i> .....	257
<i>Length property</i> .....	126
<i>LI</i> .....	32
<i>library</i> .....	284
<i>Line Breaks</i> .....	22
<i>Live Wire</i> .....	118
<i>Logical Operators</i> .....	127
<i>Loops</i> .....	249
<b>M</b>	
<i>math object</i> .....	175
<i>Meta-characters</i> .....	272
<i>MethGet</i> .....	285
<i>Method</i> .....	158
<i>methods</i> .....	293
<i>Methods</i> .....	146
<i>mkdir</i> .....	269, 270
<i>Modifiers</i> .....	272
<i>Modifying the Argument Value</i> .....	281
<i>Modules</i> .....	293
<i>Multiple Virtual Domains</i> .....	3
<i>MySQL</i> .....	324
<i>MyURL</i> .....	285
<b>N</b>	
<i>n Command</i> .....	308
<i>NAME</i> .....	69
<i>NAME=value</i> .....	184
<i>Network Class</i> .....	9
<i>Non Interactive event handlers</i> .....	150
<i>Null</i> .....	123
<b>O</b>	
<i>object</i> .....	293
<i>ODBC</i> .....	296
<i>OL</i> .....	32
<i>OnChange( )</i> .....	160
<i>onSelect( )</i> .....	160
<i>open</i> .....	264, 266, 267, 268, 269, 270
<i>Open Source Domain</i> .....	324
<i>opendir</i> .....	269
<i>operators</i> .....	126
<i>Ordered Lists</i> .....	32
<b>P</b>	
<i>P</i> .....	33, 40, 48, 62, 73
<i>p Command</i> .....	313
<i>package</i> .....	292
<i>package declaration</i> .....	292

*paired* ..... 15

*Paragraph breaks* ..... 21

*parseFloat() function* ..... 133

*parseInt() function* ..... 132

*Passing parameters* ..... 134

*password* ..... 159

*password element* ..... 161

*pattern* ..... 272

*Pattern Search(//) Command* ..... 306

*period (.)* ..... 274

*Perl* ..... 233

*Perl -d* ..... 303

*Post* ..... 158

*printf* ..... 265, 266

*PrintHeader* ..... 285

*PrintVariables* ..... 286

*Processing Form Information* ..... 231

*Programming Statements* ..... 129

*Properties* ..... 145

**R**

*r Command* ..... 308

*R Command* ..... 312

*radio* ..... 159

*radio button* ..... 168

*readdir* ..... 270

*ReadParse* ..... 285

*Recursive Functions* ..... 136

*redirection operators* ..... 266

*reference* ..... 290

*Registering A Virtual Domain* ..... 4

*Regular expression* ..... 272

*Repeating Strings* ..... 255

*REQUEST\_METHOD* ..... 231

*require* ..... 284, 292

*reset* ..... 159

*reset button* ..... 164

*Resolving Domain Names* ..... 7

*Return Function* ..... 284

*Return values* ..... 135

*rewinddir* ..... 270

*rmdir* ..... 270

*ROWSPAN* ..... 44

**S**

*s Command* ..... 307

*S Command* ..... 306

*scalar variable* ..... 238

*select* ..... 159

*select object* ..... 171

<i>Servers</i> .....	13
<i>Servers and Clients</i> .....	1
<i>set-cookie</i> .....	184
<i>shift</i> .....	254
<i>shortcut operators</i> .....	245
<i>Single quoted strings</i> .....	237
<i>singular</i> .....	15
<i>SIZE</i> .....	24, 27
<i>sort</i> .....	260
<i>Spacing</i> .....	27
<i>Special Operators</i> .....	129
<i>splice</i> .....	260, 261
<i>standard input</i> .....	264
<i>standard output</i> .....	264
<i>STANDARD OUTPUT</i> .....	231
<i>START</i> .....	32
<i>stdin</i> .....	264
<i>STDIN</i> .....	264, 265
<i>stdout</i> .....	264
<i>STDOUT</i> .....	265
<i>String</i> .....	122
<i>String Comparisons</i> .....	246
<i>string object</i> .....	174
<i>String Operators</i> .....	128
<i>strings</i> .....	236
<i>Styles</i> .....	24
<i>submit</i> .....	159
<i>submit button</i> .....	164
<i>Subroutine Arguments</i> .....	279
<i>Subroutines</i> .....	279
<i>substring</i> .....	256
<i>Super Controlled - endless loops</i> .....	131
<i>sysread</i> .....	268
<i>syswrite</i> .....	268
<b>T</b>	
<i>T Command</i> .....	313
<i>TABLE</i> .....	44
<i>TARGET</i> .....	70
<i>TCP/IP</i> .....	2, 8
<i>TD</i> .....	44
<i>TELNET</i> .....	11
<i>text</i> .....	159
<i>TEXT</i> .....	20
<i>Text fields</i> .....	161
<i>Text files</i> .....	267
<i>textarea</i> .....	170
<i>textArea</i> .....	159
<i>The World Wide Web</i> .....	10
<i>time</i> .....	254, 262

<i>Title</i> .....	21
<i>TR</i> .....	44
<i>Trace(t) command</i> .....	311
<i>TRANSMISSION CONTROL PROTOCOL</i> .....	10
<i>TYPE</i> .....	27, 32
<i>Typeglob</i> .....	282
<b>U</b>	
<i>UL</i> .....	32
<i>undef</i> .....	297
<i>Underline</i> .....	25
<i>UNIX</i> .....	271
<i>unless statement</i> .....	248
<i>Unordered List</i> .....	32
<i>Until Loop</i> .....	249
<i>use function</i> .....	293
<i>USEMAP</i> .....	59
<i>Using ! Command</i> .....	313
<b>V</b>	
<i>v Command</i> .....	309
<i>VALIGN</i> .....	44
<i>VALUE</i> .....	32
<i>values function</i> .....	241
<i>variable</i> .....	238
<i>Variable</i> .....	121
<i>Variable Scope</i> .....	135
<i>Virtual Domain</i> .....	3
<i>VLINK</i> .....	53
<i>VSNL</i> .....	6
<i>VSPACE</i> .....	37
<b>W</b>	
<i>w Command</i> .....	305
<i>Web Server</i> .....	1
<i>WHILE loop</i> .....	132
<i>WIDTH</i> .....	24, 37, 44
<i>WIN32</i> .....	
<i>ODBC</i> .....	296
<i>word boundary</i> .....	273
<b>X</b>	
<i>X Command</i> .....	309

60970c

3232