

**KWAME NKRUMAH UNIVERSITY OF SCIENCE AND
TECHNOLOGY, KUMASI**



INSTITUTE OF DISTANCE LEARNING

(BSC COMPUTER SCIENCE, 3)

**CSM 399: WEB-BASED CONCEPTS AND
DEVELOPMENT**

Credit: 3

E. O. OPPONG

J.K. PANFORD

Publisher Information

© IDL, 2010

All rights reserved. No part of this book may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage and retrieval system, without the permission from the copyright holders.

For any information contact:

Dean
Institute of Distance Learning
New Library Building
Kwame Nkrumah University of Science and Technology
Kumasi, Ghana

Phone: +233-51-60013
 +233-51-61287
 +233-51-60023
















Fax: +233-51-60014

E-mail: idldean@kvcit.org
 idloffice@kvcit.org
 kvcit@idl-knust.edu.gh
 kvcitavu@yahoo.ca

Web: www.idl-knust.edu.gh
 www.kvcit.org

Publisher's notes to the Learner:

1. Icons: - the following icons have been used to give readers a quick access to where similar information may be found in the text of this course material. Writer may use them as and when necessary in their writing. Facilitator and learners should take note of them.

Icon #1  Learning Objective	Icon #2  Activity	Icon #3  Assignments	Icon #4  Information	Icon #5  Summary
Icon #6  Time For Activity	Icon #7  Self Assessment	Icon #8  Group Discussion	Icon #9  Read	Icon #10  New Terms
Icon #11  Well Done	Icon #12  Note/Learning Tip	Icon #13  Pause	Icon #14  Question	Icon #15  Online

2. Guidelines for making use of learning support (virtual classroom, etc.)

This course material is also available online at the virtual classroom (v-classroom) Learning Management System. You may access it at www.kvcit.org

Course writers:

Emmanuel Ofori Oppong

BSc Computer Science, MSc Ind. Mathematics (Computer Solutions to Scientific problems/Optimization)

Lecturer - Computer Programming and Web Applications
Computer Science Department, KNUST

J. K. Panford

BSc Computer Science, MSc Computer Science

Lecturer – Computer Science

Computer Science Department, KNUST

Course developer(s):

Departmental Board

Computer Science Department, KNUST

Course Introduction

This Web-based Concepts and Development is designed for students of BSc Computer Science 3

COURSE OVERVIEW

This course shall emphasise the foundation of web-based programming, foundation in web-based technologies, web page creation, web-based architectures – client/server, Distributed, Web servers and Web hosting – IIs, Apache, Domain name registration, Static and dynamic web content and creation – HTML, DHTML, XML, Server side and Client side Scripting, Trading and security on the web e-commerce, e-business, encryption, signatures, Scripting languages – Open source (perl, php, python, jsp, servlets) and Microsoft based languages (ASP, ASP.NET).



COURSE OBJECTIVES

On completion of this course, you would be able to:

1. Explain web-based technologies
2. Apply XHTML for web page creation
3. Distinguish between server-side and client-side scripting
4. Use scripting languages
5. Apply ASP.Net to the development of web applications

COURSE OUTLINE

- Unit 1: Web-Based Concepts
- Unit 2: Cascading Style Sheets, Spans and Divisions
- Unit 3: Tables, Frames and Forms
- Unit 4: Client-side/server-side scripting and ASP.Net

GRADING

Continuous assessment: 30%

End of semester examination: 70%

RESOURCES

You will require web browser and Visual Studio for this course.

REFERENCES

1. Deitel, H.M and P. J. Deitel (2005). Visual Basic How to Program
2. Deitel, H.M and P. J. Deitel (2005). Java How to Program
3. Douglas Bock (2008). ASP.Net programming. www.siue.edu
4. Gottlerberg, T. T and T. N. Trainor (1991). More Excellent HTML
5. Osborne/McGraw-Hill (1991). Oracle web application Handbook
6. ASP.Net Programming. www.siue.edu

READING LIST / RECOMMENDED TEXTBOOKS / WEBSITES / CD ROM

1. Web application architecture by Leon Shklar and Richard Rosen
2. XML How to Program by Deitel, Deitel, Nieto, Lin and Dahnu
3. More Excellent HTML by T. T. Gottlerber and T. N trainor
4. www.siue.edu/dbock
5. www.w3schools.com

Table of Contents

<i>Publisher Information</i>	ii
<i>Course writers:</i>	iv
<i>Course Introduction</i>	v
<i>Table of Contents</i>	vii
<i>List of Figures</i>	ix
Unit 1	1
WEB-BASED CONCEPTS	1
SESSION 1-1: THE WORLD WIDE WEB	2
1-1.1 The Web	2
1-1.2 The Internet	3
SESSION 2-1: A WEB PAGE	7
2-1.1 Markup Languages	7
2-1.2 XHTML Tag Types	11
2-1.3 Container and Empty Tags	12
2-1.3 Creating a Web page	15
2-1.4 Basic document layout	18
2-1.5 Horizontal Rules	23
2-1.6 List Structures	24
2-1.7 Linking Pages	30
Unit 2	37
CASCADING STYLE SHEETS, SPANS AND DIVISIONS	37
SESSION 1-2: CASCADING STYLE SHEETS	38
1-2.1 Creating Style Sheets	38
1-2.2 Applying Style Sheets	41
1-2.3 Style Selectors and classes	59
1-2.4 Style Classes	62
SESSION 2-2: SPANS AND DIVISIONS	64
2-2.1 Introduction	64
2-2.2 The tag	65
2-2.3 The <div>Tag	65
Unit 3	68
TABLES, FRAMES AND FORMS	68
SESSION 1-3: TABLES	69
1-3.1 Creating Tables	69
1-3.2 thead, tfoot and tbody tags	89
SESSION 2-3: FRAMES	91
2-3.1 Set of frames	91
2-3.2 The Frameset Document	91
2-3.3 Nested Frames	96
SESSION 3-3: FORMS	101
3-3.1 Creating Forms	101

Unit 4	117
CLIENT-SIDE, SERVER-SIDE SCRIPTING AND ASP.NET	117
SESSION 1-4: SCRIPTING LANGUAGES	118
1-4.1 PHP and VB Script.....	118
1-4.2 Client side scripting with JavaScript.....	119
1-4.3 Server side scripting- Common Gateway Interface (CGI).....	154
1-4.4 Servlets and Java Server Pages	157
SESSION 2-4: Introduction to web application using Active Server Pages.Net (ASP.NET)	
.....	160
2-4.1 ASP.Net.....	160
2-4.2 Differences between Static and Dynamic Web Pages	161
2-3.4 Working with ASP.NET Web Sites	164
2-4.4 Validation Controls	176
2-4.5 Multi-Page Application	184

List of Figures

Figure 1: The OSI Model.....	4
Figure 2: Simple Request-Response Mode.....	6
Figure 3: Uniform Resource Locator.....	6
Figure 4: Evolving HTML Versions.....	8
Figure 5: Appearance of <title> tag on Browser Title Bar	15
Figure 6: Browser Display of First Web Page	17
Figure 7: Applying the <blockquote> tag to a paragraph	19
Figure 8: Nested blockquote	21
Figure 9: Browser display of <h1> tags.....	22
Figure 10: Default horizontal rules	23
Figure 11: Web page with default margins.....	42
Figure 12: Web page with 25-pixel margins surrounding the page.....	44
Figure 13: Setting paragraph margins.....	47
Figure 14: Using margin settings to increase vertical spacing	48
Figure 15: Various paragraph alignments displayed in browser window.....	50
Figure 16: A floated image	52
Figure 17: Text indentions applied to paragraphs.....	54
Figure 18: A dynamically resized images.....	56
Figure 19: Application of class styles	63
Figure 20: The example browser window is divided into three separate frames.....	91
Figure 21: Browser display of a frameset with two column frames	93
Figure 22: Browser display of a frameset with two row frames.....	93
Figure 23: A nested frameset	97
Figure 24: How CGI works.....	154
Figure 25: WelcomeServlet.java.....	157

WEB-BASED CONCEPTS

Introduction

The World Wide Web (WWW) was developed in research laboratories and universities, as a means by which researchers could easily share information about various research efforts. The Web was developed to run on the TCP/IP, the networking protocol of the internet, a global network that links computers and users from systems all over the world. However, in contrast to previous internet based services that tended to be purely text based and confusing to novice users, the Web has designed to support richer data types and be easier to use.



Learning Objectives

After reading this unit you should be able to:

1. Explain the origin of the world wide web and the internet, and the protocols
2. Create web pages with HTML/XHTML

Unit content

Session 1-1: THE WORLD WIDE WEB

1-1.1 The Web

1-1.2 The Internet

Session 2-1: A web page

2-1.1 Markup languages

2-1.2 XHTML Tag Types

2-1.3 Creating a web page

2-1.4 Basic document format

SESSION 1-1: THE WORLD WIDE WEB

1-1.1 The Web

The Web consists of a collection of computers acting as servers that uses the Hypertext Transfer Protocol, or HTTP, to serve request for information from programs on end users' computers, known as browsers because they let a user browse through content. These browsers are designed to interpret and display pages of information written using the Hypertext Markup Language, or HTML and other markup languages. These HTML pages use both tags to affect the appearance of the text on the page (whether onscreen or printed) and to allow graphics and other contents to be embedded in the page.

The programs on the servers that send requests to the browsers are known either as HTTP servers or as listeners, because they listen for a request to which to respond.

The most critical feature of the HTML is the ability to embed links to other documents inside a page. These links, when activated by the user, direct the browser to fetch and display a new "page" of information (one or more screens deep). In most GUI web browsers, the user simply clicks the linking using her mouse to jump to the new page. These links can direct the browsers to fetch new pages from any HTTP server on the Internet anywhere in the world. This allows the user, for example, to retrieve three different pages from three different machines on three different continents – without ever needing to know from where the information is to be accessed, what the time zone is, or what language humans speak there.

Some information managers saw the promise of web technology and implemented HTTP servers that were accessible only from within their own organization. Such systems have come to be known as intranets, in contrast to the publicly accessible internet. Other organizations have extended their intranet to form an extranet, one or more web applications accessible to a select group of users external to the organization. Often, such applications are used by distributors or current customers to access data that is not truly public but that needs to be available to a wider audience than just a company's employees.

In addition to delivering relatively static documents, the Web can be used as an environment in which to implement functional applications. These applications use HTML to display the GUI, or graphical user interface, on a remote user's screen (say, a paralegal large law firm with staff located in different buildings across town) while the computation and processing occur only on the server machine. This process is carried out by programs that are executed by the HTTP listener program in response to certain requests from the browser.

1-1.2 The Internet

The internet is a world-wide collection of different heterogeneous networks interconnected by routers using TCP/IP. There are many types of computer networks, including the internet and your Local Area Network (LAN). The purpose of networks is to allow one computer to communicate with others. Communications from computer to computer are said to be layered, meaning several communications format and components are used for any given network.

A popular representation of the layers of a network is the OSI Reference model, which breaks the protocols used into seven layers.

For example layer 7 is the Physical layer (the wire, fiber or frequency upon which bits are transmitted). Layer 4 is the Transport layer in which data packets are sent across the network. When communications take place (from your computer to a server for example), your application program and operating system begin the process of transmitting data being sent by breaking them down and adding header information at each layer. When the data reach layer 1, they are sent across the wire (or through the air). When the data reach their destination the header information is stripped off and the packets are reassembled for use at the other end. The diagram below shows how this process works (Figure 1).

TCP/IP is a set of protocols used at layer 4 to break data into datagrams (sets of bits) of the appropriate size from transmission and reassembly. The IP part of the TCP/IP refers to the addressing scheme used to ensure that datagrams arrive at the right computer on the Internet.

Layer 1 is the Application layer, and it is the layer that HTTP operates. HTTP is designed specifically for use with the Hypertext documents. Note that communications between computers use all of the layers, with additional bits being added to the front end (and the back end) of a data packet or datagram for each layer.

When computers (through your browsers) communicate with a Web server, the browser generates a request to the Web server. Protocols (such as HTTP, TCP and IP) running on your computer break the request down into datagrams and packets, adding headers (extra bits on the front end) and footers to each little chunk of data. The header tells the various routers, computers and programs what to do with each packet so that the request can be properly received and reassembled at the Web server. After processing your request, the Web server sends out a response using the same “break down and reassemble at the other end” process that your computer did.

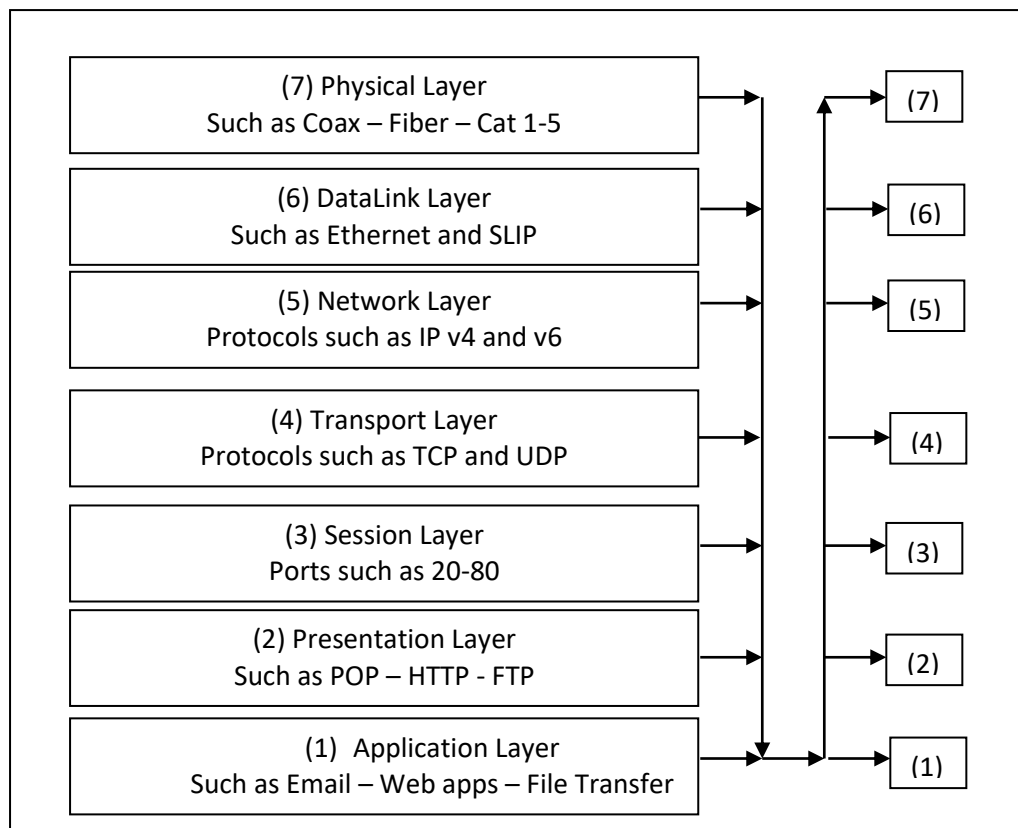


Figure 1: The OSI Model

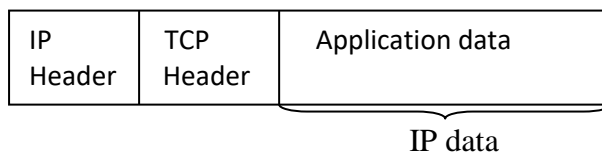
IP (INTERNET PROTOCOL)

- It uses “packet switching”. The data or message is divided in pieces called packets or IP datagram. IP specifies layout of the datagrams.
- It is connectionless, meaning that there is no connection between source and destination. Connection is established; the packets are sent independently and may use different paths.
- It is unreliable: Packets may be lost (e.g. by congestion or by buffer overflow in the router)
- A packet consists of HEADER and DATA. The header is at least 20 bytes long and contains the source and destination IP address. The maximum packet length is 64KB
- IP address is a uniformly assigned 32 bit-number. Mostly represented in dotted decimal form. E.g. 193.16.5.254

ICMP – Internet Control Message Protocol

ICMP is used for status, errors and event messages like Echo/Echo reply (used in ping). It breaks data stream up into packets or segments and adds TCP header. The TCP header for example contains sequence number checksum.

ICMP messages are sent in several situations: for example, when a datagram cannot reach its destination, when the gateway does not have the buffering capacity to forward a datagram, and when the gateway can direct the host to send traffic on a shorter route. The Internet Protocol is not designed to be absolutely reliable. The purpose of these control messages is to provide feedback about problems in the communication environment, not to make IP reliable. There are still no guarantees that a datagram will be delivered or a control message will be returned.



UDP (USER DATAGRAM PROTOCOL)

- It is connectionless and unreliable
- Basically an application interface to IP, working as multiplexer using ports to direct datagrams
- It is used mainly for applications that needs fast deliver and small fraction of data loss:

HTTP AND HTML

History of the World Wide Web (WWW)

1989 – First proposal by Tim Berner-Lee (CERN – European Center for Nuclear Research, Switzerland)

1991 – First text-based prototype

1993 – Mosaic, first graphical browser (NCSA – National Center for SuperComputing applications) authorized by More Andreessen.

1994 – CERN and MIT founded the World Wide Web Consortium (<http://www.w3.org>)

1994 – More Andreessen founds Netscape Corporation

1995 – Java was developed by SUN Corporation

REQUESTING A WEBPAGE

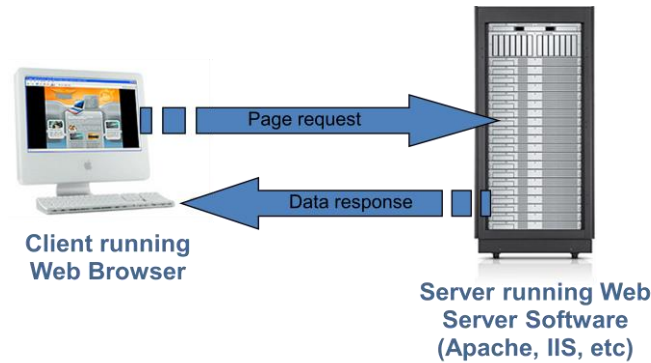


Figure 2: Simple Request-Response Mode

The request/response process is achieved by using HTTP (Hypertext Transfer Protocol)

For request, you send:

- Address
- Resource number

HTTP is stateless. All HTTP requests are independent from another; no state-information is kept.

- The browser requests the IP address of host www.w3.org through DNS
- Name server replies “18.23.0.23”
- The browser starts a TCP connection to port 80 on 8.23.0.23
- The browser sends a request function and a document name. i.e. “GET /hyp..... HTTP/1
- The server www.w3.org sends the file “The Project.html”
- The TCP connection is released
- The browser renders (interprets, displays) the contents of the file.

For every in-line (embedded) image (icon, drawing, photo), a separate TCP connection has to be established to get it. The browser also does caching therefore it checks if the page is already in the cache.

Each Web page has an address, called a URL (Uniform Resource Locator)

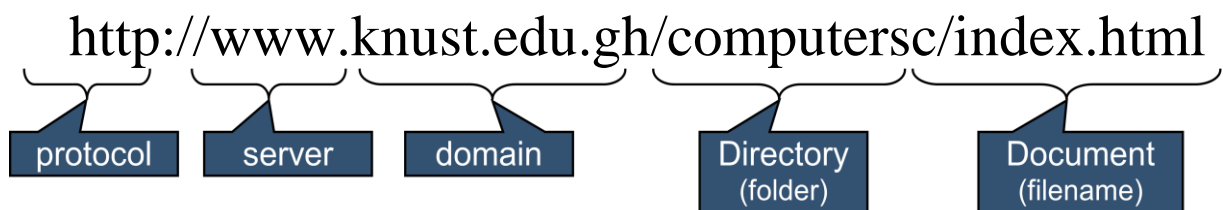


Figure 3: Uniform Resource Locator



Self Assessment 1-1

1. List the protocols used in the OSI model



Answer tips

Study the OSI model

SESSION 2-1: A WEB PAGE

2-1.1 Markup Languages

A Web page originates as a standard text file containing the information to be displayed along with formatting instructions for its presentation on the computer screen. These formatting instructions are written in a special markup language, so-called because it is used to "mark up" the information on the page to describe its arrangement and appearance when rendered in a Web browser.

From the beginning, the markup language for Web pages has been the HyperText Markup Language (HTML). It works by surrounding text and graphic content appearing on the page with special codes instructing the browser on how to arrange and display this content. As a very simple example the line of code shown below surrounds the text string Format this line of text with the HTML markup codes `<h2>` and `</h2>` (heading 2 style) in order to display the text in the style shown.

```
<h2>Format this line of text.</h2>
```

Format this line of text.

These markup codes instruct the browser to format the text by applying sizing and boldness to the characters appearing between them. Markup codes, also called HTML tags or elements, are always enclosed inside "<" and ">" symbols to set them apart from the text content to which they apply. Normally, an "opening" tag indicates the beginning point of formatting (`<h2>` in the above example) and a separate "closing" tag indicates the ending point of formatting (`</h2>` in the above example). By learning the available HTML tags you can create your own Web pages to present your text and graphics in about any way you please.

Evolving HTML Standards

The World Wide Web Consortium (W3C) maintains standards for Internet technologies, including standards for Web markup languages. The earliest standard for HTML appeared in

1995, followed by versions HTML 3.0, HTML 3.2, and culminating in HTML 4.01 in 1999. Subsequent to this latest HTML standard, however, an extensive reformulation of markup languages occurred.

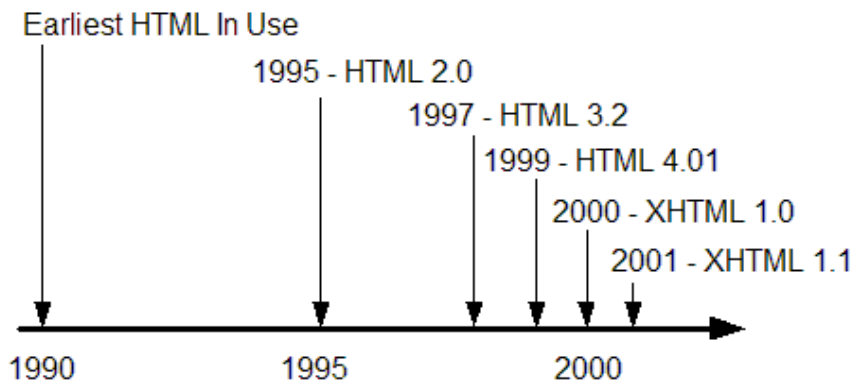


Figure 4: Evolving HTML Versions

Recent W3C efforts have focused on defining a new XML (eXtensible Markup Language) language for use as a universal markup language, replacing older languages, and with standards for creating future languages for special markup needs

Recent effort to reformulate HTML as an XML-based language has led to its current incarnation as the eXtensible HyperText Markup Language (XHTML). The initial version, XHTML 1.0, appeared in the year 2000 as a transitional standard that still recognized some of the popular features of HTML. By 2001, however, it had evolved into version XHTML 1.1 and completely abandoned the hold-over features of previous HTML standards. The language currently is in revision towards XHTML 2.0. HTML is derived from SGML (Standard General Markup Language)

HTML and XHTML are most identical, but XHTML is compliant with the XML specification, so certain rules must be followed when writing XHTML documents. These rules give some insight into what it means to write well formed XML documents.

XHTML documents must be XML-compliant. The following rules they ensure they are. Once you get used to writing Web page documents with these rules, you'll find that it's easy and not much different from writing ordinary HTML documents

1. All XHTML documents must have a DOCTYPE declaration before the root element in the document, such as
`<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" http://www.w3.org/TR/xhtml1/DTD/transitional.dtd>`

2. The root element of the document must be <html>. There may be only one element at this level, and no other elements may come before the starting <html> tag or after the ending </html> tag.
3. XHTML elements and attributes must be written lowercase. XML is case sensitive, and the accepted convention is to write all XHTML elements and attributes lowercase (only <html> is acceptable).
4. Attributes values must be encased in quotes. Attributes are inserted inside the tags that make up element (in the starting tag only) and normally are written in the name of the attribute, an equal assign and the value of the attribute. For example, to set the width of an image to 120 pixels you would write the following:
.
In XHTML, putting quotes around attribute values is required as shown in the following:
.
5. Attribute values may not be minimized. In XHTML all attributes must include a value
6. Leading and trailing spaces in attribute values will be stripped. This means any blank spaces before or after the value part of an attribute (a name-value pair) will be removed.
7. Only the id attribute can be used to identify an element uniquely.
8. Nonempty elements must be terminated (they must have an ending tag, or be terminated with a slash, like this: <br?>)
9. Elements must be properly nested, not overlapping, if they are contained within another element
10. SCRIPT and STYLE elements must be marked as CDATA areas, as shown in the following code example:

```
<script language="Javascript">
<![CDATA[
function buttonalert() {
alert("You clicked a button")
}
]]>
</script>
```

Following the rules above makes you XHTML documents compliant with XML, and you'll find that many of the rules above apply to any language that intends to be XML-compliant (languages created using XML DTDs or XML schemas)

XML

XML stands for extensible markup language. XML was developed around 1996 and is a subset of SGML. Its documents conform to SGML. XML was made less complicated than SGML to enable its use on the web

- You can define your own elements and thereby support a much wider variety of information display. XML may be used to describe chemical structures, or other scientific or artistic data that cannot be readily displayed using HTML.
- Documents may be better organized into structures to allow for easier reference using and generating items like a table of contents.
- XML allows elements to be used to sort through database information readily

To create a document, it must contain elements. Let's assume that I want to create a document with the elements LAND, FOREST, TREE, MEADOW, and GRASS. Here is how I would use these elements:

```
<LAND>
  <FOREST>
    <TREE>Oak</TREE>
    <TREE>Pine</TREE>
    <TREE>Maple</TREE>
  </FOREST>
  <MEADOW>
    <GRASS>Bluegrass</GRASS>
    <GRASS>Fescue</GRASS>
    <GRASS>Rye</GRASS>
  </MEADOW>
</LAND>
```

Each element is enclosed in <> brackets. The ending element has the '/' character before its name. As you can see, there is one element that contains all others, <LAND>.

XML requires one element that contains all others. This single element, which in this case is "LAND", is called the root element. The FOREST element contains several TREE elements, and the MEADOW likewise contains several elements of GRASS. Each element that is contained in another ends in that same element and therefore each element is properly nested.

Elements that are included in another element are considered nested. The TREE elements in the above example are nested in the FOREST element. The FOREST element is the parent element to the TREE element and the TREE element is also called the sub-element to the FOREST element. These relationships hold true as you move up and down the element hierarchy. The FOREST and MEADOW elements are sub-elements to the LAND root element.

2-1.2 XHTML Tag Types

XHTML formatting codes, or tags, surround the material to be formatted and are enclosed inside angled brackets ("<" and ">") to identify them as markup instructions. On the basis of these markup tags, the browser displays the page in the specified layout and style.

Various types of XHTML tags are used for different layout and styling purposes. The list below is one way to categorize tags according to their primary usages.

- a. **Document Layout Tags.** These tags are used to structure the XHTML document. They organize the information content on a page so that text and graphical elements appear where you want them to appear. These are the tags you use to design the overall physical and visual relationships between page elements.
- b. **Text Formatting Tags.** These tags are used to apply font faces, styles, sizes, and colors to the text appearing on the page. They package and decorate the text content of the page
- c. **List Formatting Tags.** Certain tags are used to organize text information into lists. List structures include bulleted lists, numbered lists, and others.
- d. **Graphic Formatting Tags.** These tags are used to position, size, and style drawings and pictures that appear on the page.
- e. **Linking Tags.** Web pages are hypertext documents meaning there are linkages between them. With the click of a mouse you can jump from one page to another or from one page to a particular location on another page, immediately. It is not necessary to traverse the pages sequentially. Therefore, certain tags are used to set up these linkages between pages and between different sections of a single page.
- f. **Table Tags.** Web pages provide information to visitors; they display data. Often time's data need to be organized in tabular form, into rows and columns, for better presentation, readability, and understanding. Table tags permit you to arrange data into tables; they also can be used as structuring devices for laying out the entire content of Web pages.
- g. **Frame Tags.** Web pages also can be organized into frames or into separate windows each containing a different Web page that is accessible at the same time. Frame tags permit this structural option for laying out information on a page.
- h. **Form Tags.** Forms are data collection devices. They are used to collect information from visitors in order to capture data for processing or to solicit visitor preferences about content displays. Forms are the mechanisms through which visitors interact with your Web page. A number of different form tags permit various means of user interaction.
- i. **Multimedia Tags.** Modern Web pages provide a multimedia experience, presenting audio and video information and entertainment as well as text and graphic images. Certain tags are used to link to and play audio and video files through special media players that can be embedded on the Web page.

2-1.3 Container and Empty Tags

XHTML tags are special *keywords* surrounded by "<" and ">" angle brackets. These element "names" indicate the purpose of the tag and direct the browser to treat the enclosed content in a specified fashion.

The <html> Root Element

The root element (the opening tag) of an XHTML page should be an <html> tag indicating the namespace of the applicable standard, that is, the Web location of the XHTML standard being applied to the page. In all cases, the root element is coded as shown below

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
```

The <html> tag surrounds all the content on the page and indicates that this is an HTML document that the browser should treat as such. Likewise, <body> tags enclose all page content that is displayed inside the browser window. In conformance with XHTML standards, all keywords *are typed in lower-case characters*.

Most XHTML tags are coded as *pairs* of opening and closing tags, called container tags. The opening tag is the keyword itself appearing inside < and > symbols; the closing tag is in the same format with the keyword preceded by a forward slash (/). This pair of container tags encloses the information to which the formatting applies. Thus, the pair of container tags <html>...</html> encloses the entire HTML document to indicate that everything in between is coded in HTML. In the same way the pair of <body>...</body> container tags encloses all page content that is displayed inside the browser window.

Certain other tags are a not container tag; that is, they do not "surround" information to be formatted but are coded as single, stand-alone tags. These empty tags are coded in a special way to conform to XHTML standards. They must include a forward-slash (/) character *immediately before the closing angle bracket*. For example, the single tag to display a horizontal line across the page is coded as <hr/> and the single tag to create a line break in a text paragraph is coded as
. This special coding indicates that the tag serves as *both* the opening and closing tag.

XHTML tags are identifiers and containers of text and graphic information appearing on a Web page. Their primary use is to describe the *structure* of that content, to package it for placement on the page. However, most Web pages are more than straight text and in-line images placed there by XHTML tags. The information on the page has *styling* applied to make it more attractive and readable. Various typefaces, text sizes, colors, and other formatting characteristics make the page more inviting to read and, when used judiciously, can make the information more understandable.

Under previous versions of HTML, most of the styling of page content took place through attributes coded as part of its tags. Attributes provided additional formatting specifications beyond those implied by the tag name itself. For example, font styles could be assigned to text information by surrounding it with a tag containing attributes to set the font face, size, and color: . Various tags supplied various attributes to format their contained content in various ways.

Under current versions of XHTML, tag attributes have all but disappeared, being deprecated (deprecated in use) in favor of Cascading Style Sheets (CSS), or just "style sheets" for short. A style sheet is a set of styling characteristics associated with XHTML tags. These styling characteristics are composed of style *properties* and *values*, written in the special syntax of style sheets, and assigned to the tags that will inherit these styles. For instance, one way to assign a style sheet to a tag is to include it as a style attribute inside the tag.

```
<p style="font-family:arial; font-size:14pt; color:red; font-weight:bold">
```

This is a text paragraph.

```
</p>
```

In this example, a paragraph of text, surrounded by a <p> (paragraph) tag is given an Arial type face of size 14pt (14 points), colored red, and displayed in bold characters. Four *property: value* specifications (e.g., font-family:arial) apply these styles. This is an example of including a style sheet as part of the tag to which it applies.

Working with XHTML Documents

XHTML documents have a simple, common structure that forms the basis for designing all Web pages. This basic structure of tags is shown in the following listing with associated tags described in the following sections.

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

```
<!DOCTYPE html
```

```
  PUBLIC "-//W3C//DTD XHTML 1.1//EN"
```

```
  "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
```

```
<head>
```

```
  <title>page title goes here</title>
```

```
</head>
```

```
<body>
```

```
  .
```

```
  page content goes here
```

```
  .
```

```
</body>
```

```
</html>
```

As you begin coding an XHTML page you can start with this template. In fact, you may wish to create this document and save it as a template file. Then, when you start a new page, simply open this document, save it under the name of the new page, and continue coding for the particular information to appear on that page.

All XHTML documents should begin with the prolog lines shown below. The first line indicates that this document is based on XML version 1.0. The remaining lines point to the Document Type.

The <html> Tag

The <html> container tag surrounds all XHTML coding in the document. This tag indicates that the enclosed information contains XHTML coding and should be rendered as such in the browser. In conformance with XHTML standards the opening tag includes a reference to the location namespace of the validation standards to be applied to this document along with attributes that specify the language used, English (en) in this case.

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
```

The <head> Tag

The <head> container tag encloses the head section of the XHTML document. The head section supplies a title for the document (see below) along with other information related to formatting and indexing of the document. For present purposes only a title appears in the head section. Other tags that can be included in the head section are discussed as needed.

```
<head>
  <title>page title goes here</title>
</head>
```

The <title> Tag

The <title> container tag gives a title to the document. This tag encloses a string of text that appears in the browser's Title Bar when the page is opened. The <title> tag provides helpful page identification information to the person visiting the various pages of your Web site. Note that <head> and <title> sections are required for conformance with XHTML 1.1 standards.

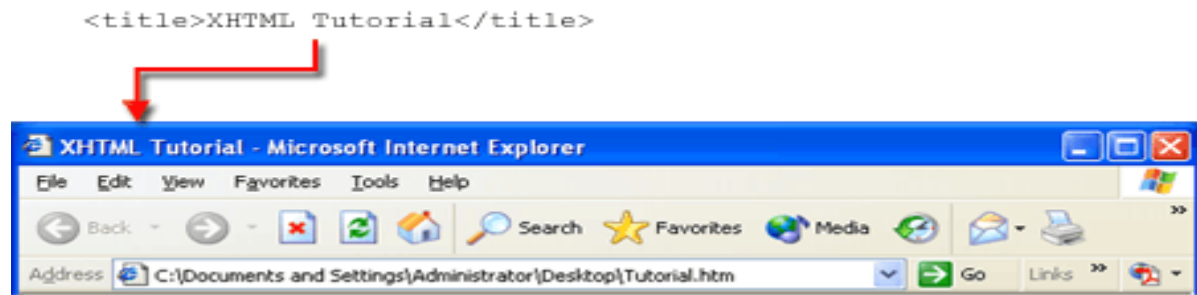


Figure 5: Appearance of <title> tag on Browser Title Bar

The <body> Tag

The bulk of the coding of an XHTML document appears in the body section surrounded by the <body> container tag. Only information appearing inside this tag is displayed in the browser window. In its simplest form the body section contains plain text that is displayed in the default font style inside the browser window. Browsers normally display text using Times New Roman font face at approximately 12-point size.

All Web pages begin with this basic document structure. Then the <body> of the document is expanded with text and other page elements that are to be displayed inside the browser window. Various arrangements of these display elements and control over their appearance are accomplished by enclosing them within additional XHTML tags.

In order to view your work it is not necessary to be connected to the Internet or to be linked to a server on the World Wide Web. You can do all your work locally. If you happen to have an account with an Internet Service Provider that provides personal home directories then you can copy documents to your directory for viewing on the Web. For purposes of this tutorial, though, you can create Web pages on your computer's hard drive or on a removable thumb drive or diskette and view the pages through your browser.

2-1.3 Creating a Web page

Text Editing

XHTML documents are created with text editors or with special HTML editors designed for that purpose. For this course a simple text editor such as Windows Notepad will be used to type the HTML document.

You also need to make sure that you save the document with the .htm or .html file extension. This extension identifies the document as a Web page and is needed for the browser to recognize it as such.

The saved HTML document with the .htm extension is now ready to be viewed in your browser. You can open the document directly in the browser by double-clicking its icon, or you can open your browser and use the File menu to navigate to the correct drive, folder, and document. When the document is loaded into the browser the address appearing in the Address box of your browser indicates this path to the document.

Incidentally, there is a quick way to open a Web document for editing in Notepad if you happen to be viewing the page in your browser. From the browser's View menu choose Source, meaning to view the XHTML source coding for the page. The page automatically opens in the Notepad editor.

Creating Your First Web Page

It is now time to create your first Web page and view it in your browser. This page isn't very fancy but it will familiarize you with the process of coding, editing, saving, and retrieving a page for display in your browser. So, begin by opening the Notepad editor and enter the text and code shown in the following listing.

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

<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
  <title>First Web Page</title>
</head>
<body>

<!-- Here is a paragraph for browser display -->

<p>This is my first attempt at creating a Web page. I don't quite understand
yet what I'm doing, but it seems easy enough. Perhaps when I learn a few
more XHTML tags and CSS styles, I'll start to get comfortable and can dazzle
you with my skills.</p>
</body>
</html>
```

XHTML Comments

It is usually a good idea to place comments in your Web document to describe its various sections. Comments are general descriptions or explanations of XHTML code. These serve as useful reminders of the purposes or contents of sections of code when you or someone else

returns at a later time to edit the document. In the above example, a general comment has been placed at the beginning of the <body> section.

```
<!-- Here is a paragraph for browser display -->
```

Comments are coded between a pair of <! -- and --> tags. These tags can enclose a single-line comment as illustrated above, or they can enclose multiple lines of XHTML coding. Any code or text appearing between these symbols is ignored by the browser and does not show up on the page. Besides their use for including comments in a document, comment tags can be used to temporarily suspend browser display of a section of code.

A second method of commenting a small section of HTML code is by using the exclamation mark (!) within a tag. This symbol can be used to suspend display of an entire tag -- by adding it to the beginning of the tag -- or one of its attributes -- by adding it in front of the attribute. In the following example this symbol is used to remove formatting from the line of text by commenting its surrounding <p> tag.

```
<!p style="font-family:arial; color="red">  
Format this line in red.  
</p>
```

Incidentally, the exclamation mark is not a formal, approved comment symbol. Any other character can be used since it simply corrupts the tag name or attribute name, making it meaningless to the browser. An exclamation mark is used here simply for compatibility with the standard comment tag.

Saving and Displaying Your Web Page

After coding the page, you must save the document so that it can be retrieved by your browser. Where you save the document depends on your development environment. Also remember to save the document with the special file extension .htm or .html. Save this current document under the name FirstPage.htm or any other name of your choosing.

Now, open your browser and retrieve the page. If you had coded your document as in the example above, your Web page should appear in the browser window similar in style to that shown below.

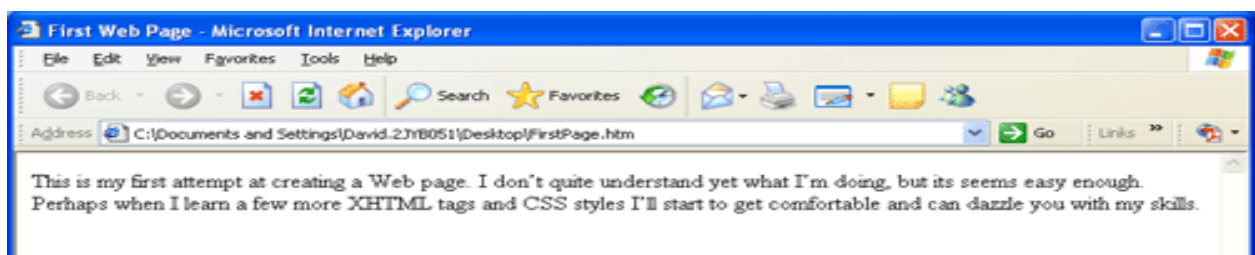


Figure 6: Browser Display of First Web Page

2-1.4 Basic document layout

XHTML tags have the primary purpose of structuring and arranging text and graphic content on a Web page. They are not designed to style or decorate the content, only to arrange it so that information is presented in a visual order for ease of reading and in a logical order for enhanced understanding.

In addition to arranging content on the page you can apply visual styles to it. You can apply different sizes, colors, and font styles to text and you can size and decorate graphic images. Throughout these courses you are introduced to styling methods that add these kinds of visual appeal to text and graphic elements. First, however, it is necessary to become familiar with the basic tags that bring layout order to page content.

Paragraphs

Most Web pages are text documents, and the most common way to arrange text on a Web page is in paragraph layout. For purposes of a Web page, a paragraph is a block of text *preceded and followed by a single blank line*. A paragraph is aligned at the left margin of the page and is word-wrapped inside the browser window.

The <p> Tag

In order to format a block of text as a paragraph, the <p> (paragraph) tag surrounds the enclosed text. The general format is <p>text</p>

Any amount of text can appear between the opening and closing tags. When displayed in the browser the enclosed text is single spaced and preceded and followed by a single blank line to separate it from surrounding page content. A <p> tag is known as a block-level tag. It creates a line break and begins displaying its content on a new line in the browser. After its content is displayed, another line break takes place, creating a block of text set off from surrounding content in the browser.

The <blockquote> Tag

The <blockquote> tag offers a way to indent a block of text, to offset it from surrounding content for special emphasis as would be done for a quoted citation from which the tag gets its name. This container tag indents its enclosed text a fixed number of pixels (approximately 40 pixels) from the left and right margins. Its enclosed text is word-wrapped and preceded and followed by a blank line. The general format for the <blockquote> tag is

```
<blockquote>
  <p>text</p>
</blockquote>
```

Note that the `<blockquote>` tag is *not* considered to be a block-level tag under XHTML 1.1 standards, even though it produces line breaks and blank lines before and after its enclosed text. It does not take the place of the block-level `<p>` tag required to enclose a block of text and set it off from surrounding content. The `<blockquote>` tag simply surrounds the paragraph block, including its enclosing `<p>` tag.

In the following example a `<blockquote>` tag offsets the middle paragraph from surrounding text. Browser output is shown.

`<p>`Here are three paragraphs. This first paragraph is formatted with a standard paragraph tag and is blocked at the left margin of the page.`</p>`

`<blockquote>`

`<p>`This second paragraph is formatted with a blockquote tag to indent its left and right margins approximately 40 pixels from the edges of the page.`</p>`

`</blockquote>`

`<p>`This final paragraph is coded like the first one. It is enclosed within a standard paragraph tag and is blocked at the left margin.`</p>`

Listing 2-4. A text block formatted with a `<blockquote>` tag.

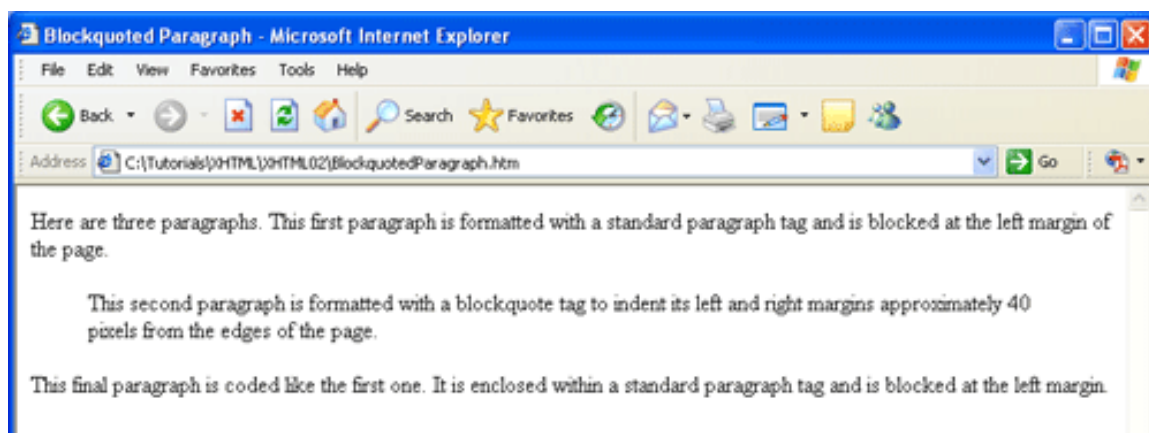


Figure 7: Applying the `<blockquote>` tag to a paragraph

Nested Blockquotes

In the following example, `<blockquote>` tags are used to offset three paragraphs, the middle one of which is further indented inside its outer blockquoted paragraphs.

`<p>`Here are five paragraphs. This first paragraph is formatted with a standard paragraph tag and is blocked at the left margin of the page.`</p>`

```
<blockquote>
<p>This second paragraph is formatted with a blockquote tag to indent its
left and right margins approximately 40 pixels from the edges of the page.</p>
```

```
<blockquote>
<p>This paragraph is also surrounded by blockquote tags. It is further
indented within the margins produced by its outer blockquote tag.</p>
</blockquote>
```

```
<p>This paragraph is aligned like the second paragraph since it also appears
inside the outermost blockquote tag.</p>
</blockquote>
```

```
<p>This fifth paragraph is coded like the first one. It is enclosed within
a standard paragraph tag and blocked at the left margin.</p>
```

In this example, the two `<blockquote>` tags are nested; that is, a `<blockquote>` tag appears *inside* a `<blockquote>` tag.

Line Break

Coding paragraphs with `<p>` tags is the most common method of structuring text on a Web page.

Most Web pages are text pages, and paragraphs are convenient and readable methods of text presentation. Other varieties of text structuring, however, are available.

The `
` Tag

The `
` (line break) tag causes the browser to end a line of text and continue display on the next line in the browser window. It does not, as in the case of paragraphs, leave blank lines before or after the ended text line.

The `
` tag is *not* a container tag; it does not enclose text and it does not have an accompanying closing tag. This empty tag is simply placed within the text wherever a line break should occur. The tag is handy for displaying lists of items, lines of verse, or other groupings of individual, single-spaced text lines. The following code uses line breaks, for example, to end individual lines of verse packaged inside a `<blockquote>` tag.

```
<p>Here is a tale about Mary and a pesky little lamb which followed her
anywhere and everywhere she went.</p>
```

```
<blockquote>
<p>Mary had a little lamb,<br/>
Its fleece was white as snow;<br/>
And everywhere that Mary went,<br/>
```

The lamb was sure to go.</p>
</blockquote>

<p>Mary had an awkward social life. It's awfully difficult to date with sheep trailing around after you all the time.</p>

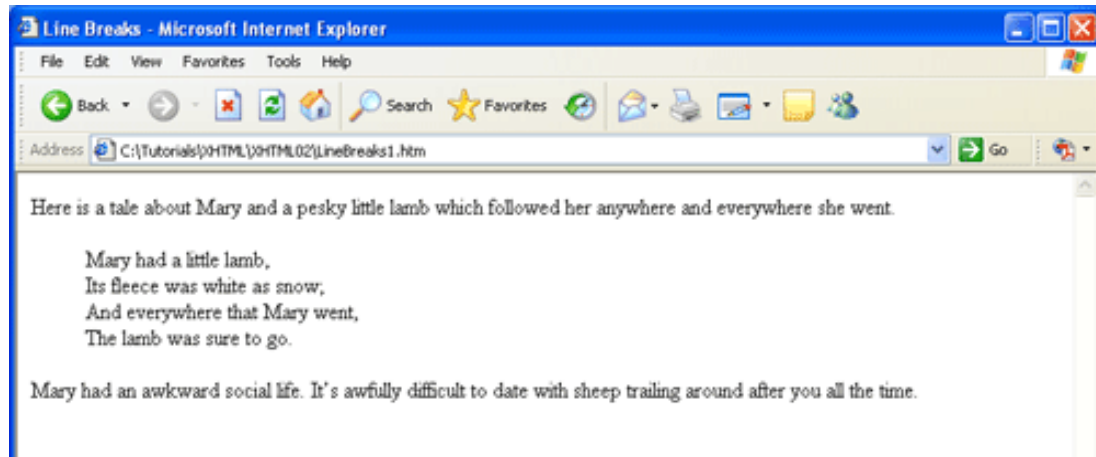


Figure 8: Nested blockquote

The four lines of verse are enclosed inside a blockquoted paragraph to offset and indent them from surrounding paragraphs. Each line of verse appears on a separate text line a single space below the preceding line.

It is important to remember that a `
` tag is an in-line tag, not a block-level tag; therefore, under XHTML 1.1 standards it cannot appear on a line by itself or as a stand-alone tag. It must be enclosed inside a `<p>` or other block-level tag as is done in the previous example.

All browsers still recognize the `
` tag, coded without the closing slash required for an XHTML empty tag. Future versions of XHTML will not recognize this tag, so it is best to avoid its use.

Headings

Text appearing on a Web page is displayed in a default font style and size unless specified otherwise. Most browsers display text in the Times New Roman font face, or font family, at approximately 12-point size. In order to add visual variety to your pages you will likely want to choose other font faces and sizes for different sections of your document. This is a topic that is taken up in more detail later; however, there are simple ways to change text sizes to permit you to add various levels of headings to your pages.

The `<h1>` (heading) tag is a block-level tag that encloses a string of text for display in one of six heading styles.

General format for <h*n*> tag

The number *n* in the tag ranges from 1 (largest) to 6 (smallest). These six heading levels are coded in the following example. A standard paragraph is displayed for comparison.

```
<h1>This is Heading Level 1</h1>
```

```
<h2>This is Heading Level 2</h2>
```

```
<h3>This is Heading Level 3</h3>
```

```
<h4>This is Heading Level 4</h4>
```

```
<h5>This is Heading Level 5</h5>
```

```
<h6>This is Heading Level 6</h6>
```

```
<p>This is a normal text paragraph. </p>
```

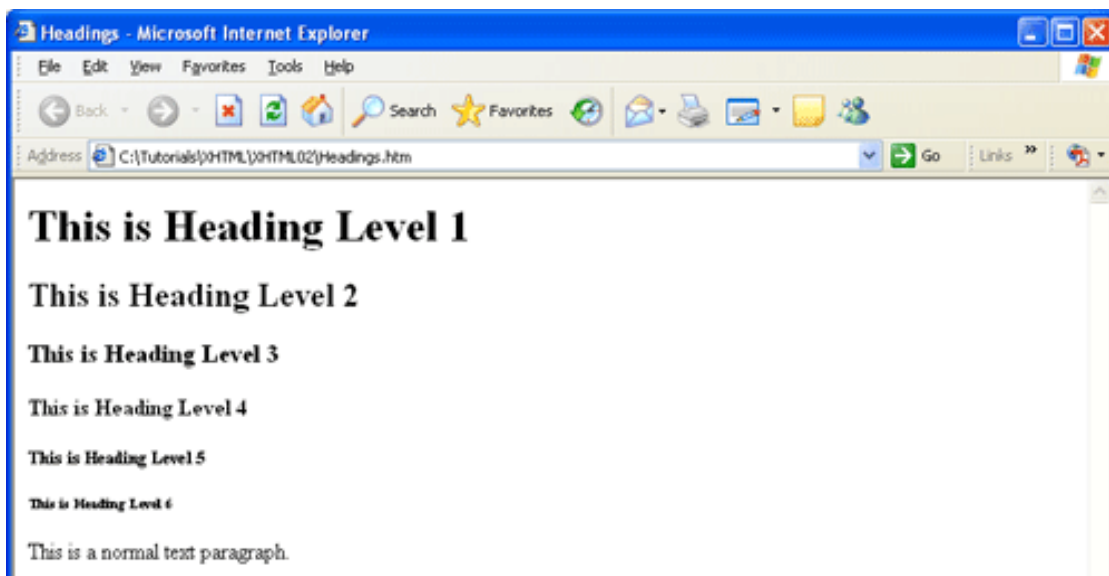


Figure 9: Browser display of <h*n*> tags

Headings are displayed in bold characters in the default font face. They appear on a line by themselves preceded and followed by a single blank line. Thus, the <h*n*> tag cannot be used to change text sizes in the middle of a line or middle of a paragraph since it forces a line break before and after its enclosed text. It is used for block-level, stand-alone text, most often as content headings to identify and set apart sections of a Web page.

Deprecated align Attribute

Headings can be aligned to the left (default) or right, or they can be centered on the line by applying the align="left|center|right" attribute to the <h*n*> tag. The following code, for example, centers a heading horizontally between page margins.

```
<h1 align="center">Centered Heading</h1>
```

Since align is a deprecated attribute under XHTML standards it should be used only as a temporary method to align headings. Later tutorials cover style sheet standards for aligning text horizontally on a page.

2-1.5 Horizontal Rules

The primary way to separate and identify sections of a Web page is by use of major and minor headings coded within `<h1>` tags. Different sections of a Web page also can be visually separated by displaying horizontal rules between them. This is done with the `<hr/>` (horizontal rule) block-level tag to draw a separator line across the page. The general format for this tag is shown in Figure 10.

The `<hr/>` tag causes a line break with the rule appearing on a line by itself. The default rule is 2 pixels in height and spans the width of the browser window

`<h2>Horizontal Rules</h2>`

`<p>`A horizontal rule is used to separate content sections of a page. The following is a default rule.`</p>`

`<hr/>`

`<p>`The rule is 2 pixels in height and spans the width of the browser window. It is preceded and followed by blank lines.`</p>`

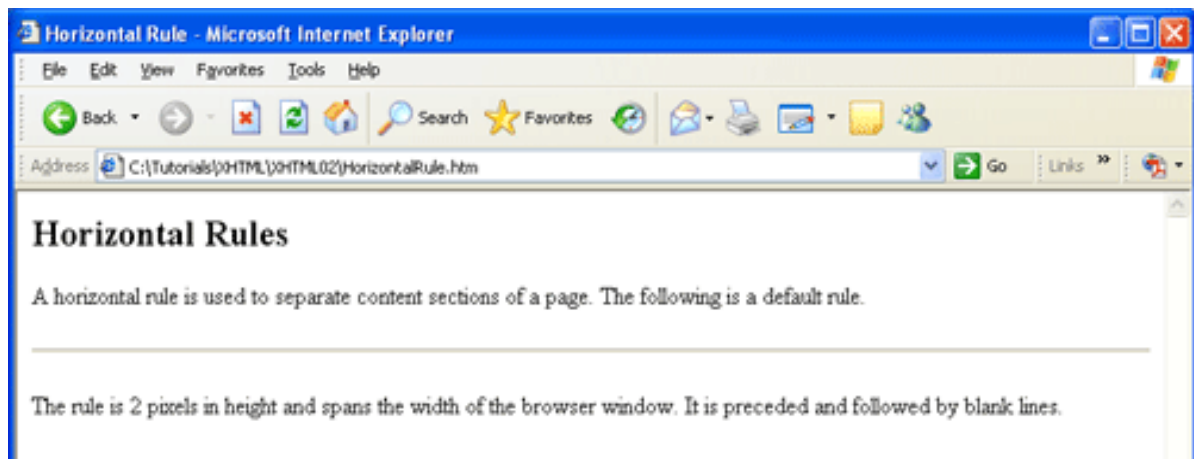


Figure 10: Default horizontal rules

In a later tutorial you learn how to style a horizontal rule by adjusting its height, width, color, alignment, and display characteristics. For present purposes the default rule can be visually effective in breaking up lengthy spans of text or indicating logical breaks in content.

Deprecated <hr> Tag and Attributes

The <hr> tag (without the closing slash) is still recognized by browsers and produces the same rule as the <hr/> tag. It is not valid under XHTML 1.1 standards. The tag also can include the following attributes to style the rule, attributes which also are recognized in the <hr/> tag but are deprecated.

`align="left|center|right"`

`size="n"`

`width="n|n%"`

`noshade`

`color="color"`

The align attribute positions the rule to the left (default), centered, or to the right across the line. The size attribute sets the height of the horizontal rule in pixels. The width attribute sets the rule's width in pixels or as a percentage of the window width. Rules can be displayed as a solid bar by coding noshade (without an associated value) and assigned a color with the color attribute. The color value is given as a color name or hexadecimal value, assignments which are discussed in a later tutorial.

The following rule is given by the code:

```
<hr align="center" size="5" width="50%" noshade color="grey">
```

Since the <hr> tag has been deprecated under current standards it should not be used. Similar alignment and styling attributes can be assigned to the newer <hr/> tag although they should be avoided in favor of newer styling methods discussed later.

2-1.6 List Structures

XHTML provides text structuring in the form of lists. You can produce lists of bulleted items, lists of numbered items, and lists of terms and definitions. The first two lists resemble single-spaced lines of text with the addition of prefixed bullets or numbers. The latter list is similar in display to a series of blockquoted paragraphs.

Unordered Lists

An unordered list is a series of items preceded by bullet characters and set off from surrounding text by blank lines. The list is single spaced and indented from the left margin.

List Item 1

List Item 2

List Item 3

An unordered list is created with the block-level container tag enclosing listed items identified with block-level (list item) container tags. The general format for an unordered list is

```

<ul>
  <li>list item</li>
  <li>list item</li>
  ...
</ul>

```

For example, the bulleted list shown above is given by the following code.

```

<ul>
  <li>List Item 1</li>
  <li>List Item 2</li>
  <li>List Item 3</li>
</ul>

```

Items in the list are single spaced and preceded by a bullet character. If text for a list item is wider than the width of the page it is word wrapped and indented inside the bullet character. Items can be enclosed inside <p> tags -- or
 tags can be coded between items -- to increase line spacing between them. The following list, for example, surrounds list items with <p> tags to leave blank lines between entries.

```

<ul>
  <li><p>This is the first item in the list. Text following the bullet
character is word wrapped inside the bullet. Paragraph tags are used to
leave blank lines between items in the list.</p></li>

  <li><p>This is the second item in the list. Text following the bullet
character is word wrapped inside the bullet. Paragraph tags are used to
leave blank lines between items in the list.</p></li>
</ul>

```

Unordered lists can be nested inside each other. For example, a bulleted list appearing inside a second bulleted list appearing inside a third bulleted list is produced by the following code and is displayed in the browser as shown below. Note that a nested unordered list is part of the list items of its container list. That is, the ... tags for the nested list appear *inside* a pair of ... tags of the enclosing list.

```

<ul>
  <li>List Item 1</li>
  <li>List Item 2
    <ul>
      <li>List Item 2a</li>
      <li>List Item 2b
        <ul>
          <li>List Item 2b1</li>
          <li>List Item 2b2</li>
        </ul>
      </li>
    </ul>
  </li>
</ul>

```

```
</ul>
</li>
<li>List Item 3</li>
</ul>
```

- List Item 1
- List Item 2
 - List Item 2a
 - List Item 2b
 - ListItem 2b1
 - List Item 2b2
- List Item 3

Each nested list is further indented inside its container list. Also, different bullet characters are used for nested lists. By default, a disc character marks the outer-most list, a circle marks the next inner-most list, and a square marks the inner list. Notice that when lists are contained inside other lists that no blank lines surround the interior lists as they do when a single list appears within the normal flow of text on the page.

Deprecated type Attribute

The type="disc|circle|square" attribute can be coded inside the opening tag in order to specify the style of bullet character to use if different from the default disc character. Current XHTML standards do not promote use of the type attribute and provide other means shown later for specifying bullet characters.

Ordered Lists

An ordered list is a series of items preceded by sequence numbers and set off from surrounding text by single blank lines. By default, the list is numbered with decimal numbers beginning with 1 and numbered consecutively through the last item in the list. The list is single spaced and indented from the left margin in the same way as an unordered list. An example ordered list is shown below.

1. List Item 1
2. List Item 2
3. List Item 3

An ordered list is created with the container tag enclosing list items identified with container tags. The general format for coding an ordered list is

```
<ol>
<li>list item</li>
```

For example, the ordered list shown above is

```
<li>List Item 2</li>
<li>List Item 3</li>
</ol>
```

Items in the list are single spaced and word-wrapped inside the numbering character. List items can be enclosed inside `<p>` tags or separated by `
` tags to increase line spacing between items.

Nested Ordered Lists

Ordered lists can be nested inside each other, with subordinate lists indented from the next outer-most list. All nested ordered lists use the *same* decimal numbering system beginning with decimal 1.

```
<ol>
  <li>List Item 1</li>
  <li>List Item 2
    <ol>
      <li>List Item 2.1</li>
      <li>List Item 2.2</li>
    </ol>
  </li>
  <li>List Item 3</li>
</ol>
```

Deprecated type and start Attributes

A type attribute can be coded inside the opening `` tag in order to specify one of five different numbering characters. The attribute value can be `type="1"` for decimal numerals (the default), `type="A"` for upper-case letters, `type="a"` for lower-case letters, `type="I"` for upper-case Roman numerals, and `type="i"` for lower-case Roman numerals. The tag `<ol type="A">`, for example, produces the following list of alphabetically ordered items.

- A. List Item 1
- B. List Item 2
- C. List Item 3

You can choose the beginning sequence value of an ordered list by coding the optional `start="n"` attribute for the `` tag. A starting sequence value is needed, for example, when an ordered list is interrupted by other elements on the page.

As shown in the output below two consecutively sequenced lists are separated by a paragraph. The first list begins with "A" and is sequenced through "E" since there are five

items in the list. The second list needs to begin its sequence with the sixth letter "F". If a starting value is not specified sequencing begins anew with "A".

This is the beginning of the list:

- A. List Item A
- B. List Item B
- C. List Item C
- D. List Item D
- E. List Item E

This is a continuation of the list:

- F. List Item F
- G. List Item G
- H. List Item H
- I. List Item I
- J. List Item J

Code for the above lists is shown below. The first ordered list uses upper-case letters (type="A") beginning with "A" and ending with "E". The second list overrides this default sequencing by specifying start="6" in its opening tag. Thus, the second list is ordered consecutively from "F" through "J".

<p>This is the beginning of the list:</p>

```
<ol type="A">
  <li>List Item A</li>
  <li>List Item B</li>
  <li>List Item C</li>
  <li>List Item D</li>
  <li>List Item E</li>
</ol>
```

<p>This is a continuation of the list:</p>

```
<ol type="A" start="6">
  <li>List Item F</li>
  <li>List Item G</li>
  <li>List Item H</li>
  <li>List Item I</li>
  <li>List Item J</li>
</ol>
```

Definition Lists

A definition list is a series of terms and definitions offset from surrounding text by blank lines. The terms in the list are blocked at the left margin; definitions are indented and word wrapped on the following lines. An example definition list is shown below.

Term 1

This is the Term 1 definition. The definition term appears on a line by itself and is followed by a definition text block. The definition is indented and word wrapped.

Term 2

This is the Term 2 definition. The definition term appears on a line by itself and is followed by a definition text block. The definition is indented and word wrapped.

The general formats for tags used to create definition lists are given below.

```
<dl>
  <dt>Term 1</dt>
    <dd>Definition text for Term 1</dd>
  <dt>Term 2</dt>
    <dd>Definition text for Term 2</dd>
  ...
</dl>
```

A definition list is enclosed inside <dl> tags and contains one or more <dt> (definition term) tags listing the items to be defined. Each definition term has an associated <dd> (definition description) tag enclosing the definition for the term.

```
<dl>
  <dt>Term 1</dt>
    <dd>This is the Term 1 definition. The definition term appears on
      a line by itself and is followed by a definition text block.
      The definition is indented and word wrapped.</dd>
  <dt>Term 2</dt>
    <dd>This is the Term 2 definition. The definition term appears on
      a line by itself and is followed by a definition text block.
      The definition is indented and word wrapped.</dd>
</dl>
```

When displayed in the browser, items in the list are single spaced with no blank lines appearing between the terms. If you wish to include additional line spacing you can code <p> tags surrounding the definitions or
 tags between them.

A definition list, of course, can be used for purposes other than defining terms. Any time you need a list of items (terms), each followed by an indented paragraph (definitions), you can use a definition list.

2-1.7 Linking Pages

The original purpose of the World Wide Web was to provide a mechanism for non-linear access to information. In order to activate this type of retrieval, information is packaged as a set of Web pages; then hypertext links, or hyperlinks, are created to navigate from one page to another in whatever sequence the reader prefers. Varieties of page links are discussed more fully. Here you are introduced to the basic concept of linking and how to set up text links for navigation between pages.

Most often a link from one page to another is made by creating a text “hot spot” on one page that, when clicked with the mouse, opens a second page which replaces the first page in the browser window. The string of text that is made sensitive to a mouse click can be a single word or several words, usually suggestive of the contents of the destination page. Normally, a second link appears on the linked page for return to the original page. It is considered poor Web citizenship to leave visitors at the end of a series of links with no way to get back to the starting page.

The <a> Tag

The most common type of link is a clickable word or phrase that transfers directly to the target page. A text link of this sort is created by surrounding the text string with an <a> tag, also called an anchor tag, specifying the location of the page to which to link. The basic format for coding this tag is `linking text`

The href (hypertext reference) attribute specifies the URL of the linked page. Links can be made to other pages at the local Web site or to pages at remote sites anywhere on the World Wide Web.

If the linked page is in the same location -- in the same directory -- as the linking page, then the URL is simply the page name. This local link is called a relative link since the path to the linked page is relative to the location of the linking page.

When linking to a remote site the URL must be a full Web reference including the "http://" protocol followed by the domain name of the site. This external link is called an absolute link since the path to the linked page is a full Web address.

The <a> tag surrounds a string of text that is to be made click-sensitive and that activates the link. Usually the text is descriptive of the destination link. It can be a single letter, a word, a

phrase, or any amount of text that can reasonably serve as a link. When this text is clicked, the linking page is immediately replaced by the linked page in the browser window.

The <a> tag is an in-line tag, meaning that it must appear inside a <p> tag or other block-level tag for compatibility with XHTML 1.1 standards.

```
<p>
<a href="Organize.htm">Topical Organization</a><br/>
<a href="Supplement.htm">Supplemental Information</a><br/>
</p>
```

```
<p>When creating links between Web pages it is important to provide return
links to the linking pages. The page author should not leave readers dangling
at the end of a series of links with no direct way to return to the home page.
</p>
```

If you know the URL of a Web site anywhere on the World Wide Web you can make a link to that site. This URL is normally in the form `http://domain name`, where the `http` protocol prefixes the site location (domain name). This type of absolute reference takes you to the home page of that site.

The following link code, for example, is used to open the home page at `www.knust.edu.gh`

```
<a href="http://www.knust.edu.gh">Go to KNUST</a>
```

Go to KNUST

The Deprecated target Attribute

By default, linked pages are opened in the same browser window as the linking page, replacing it. When linking to a remote site a risk is that your visitors will navigate the remote site and eventually lose contact with your site. They can repeatedly click the browser's Back button to get back to your site, but if they have wandered around the remote site to any great extent then backing up to your site may be impractical. You end up losing your visitors to the remote site.

One way around this problem is to *always* link to remote sites in a separate browser window. It is quite easy to open a page in a different window; just add the attribute `target="_blank"` to your <a> tag as shown below for the above link to the Google site.

```
<a href="http://www.knust.edu.gh.com" target="_blank">Go to KNUST</a>
```


With this technique visitors can browse the remote site as much as they wish in a second browser window, and by closing that window return immediately to your linking page in the original window.

It should be noted that the target attribute is not valid under XHTML 1.1 standards. Its use should be considered a temporary way to open Web pages in a new browser window. Later you are introduced to an alternative method to the target attribute that does not violate XHTML standards.

Creating text links as described here represent only a couple of the ways in which hyperlinks can tie together multiple Web pages. In a later tutorial other varieties of links and linking methods are discussed.

Displaying Pictures

In addition to presenting text content Web pages can display graphic images. Later tutorials discuss this topic more fully. Here, basic methods of placing pictures on a page are introduced.

Graphic Formats: There are three popular graphic formats used for Web images. GIF (Graphics Interchange Format) is the most widely supported. Pictures saved in this format have the .gif file extension. GIF format can display images in black and white, grayscale, or color. It is most often used for line drawings, logos, icons, and other “spot color” images, usually not for “continuous color” images such as photographs. Incidentally, the acronym GIF is pronounced “jiff.”

The PNG (Portable Network Graphics) format was designed as a replacement for the GIF format. PNG, pronounced “ping,” has the advantage over GIF of improving image quality while offering better image compression. It does not, however, support animated images. PNG files use the .png extension.

The JPEG (Joint Photographic Experts Group) format, pronounced “j-peg,” is designed for storing photographic images. It can represent millions of colors and uses compression techniques to compact all of this color information into small file sizes. Pictures saved in this format usually have the .jpg file extension.

You can display pictures taken with digital cameras or scanned from photographs, or you can copy images from public clip art collections on the Web. If you have available any of the popular graphic editing software packages, then it is usually a simple matter to edit, resize, and convert digital photographs or scanned images to Web formats. Just make sure that images are saved in GIF, PNG, or JPEG format for display on your Web pages. Other formats may display properly in the browser, however, their large file sizes may make it impractical for visitors to download these images during Web access.

The Tag

Both GIF and JPEG images are placed on Web pages by using the in-line tag. The general format for this tag is shown below.

```

```

An tag is coded on the Web page *at the location* at which the image is to be displayed. The src attribute gives the source, or location, of the image file. If the file is in the same directory as the Web page on which it is displayed, then only the file name is needed as the src URL.

The alt attribute is required under XHTML 1.1. Its "text" value gives a brief pop-up description of the image when the mouse is moved on top of the picture. This text string is displayed when users have graphic displays turned off in their browsers, thus giving a text description of missing pictures. Also, the text can be vocalized by special reader software used by visually impaired visitors who otherwise might not be able to see or focus clearly on images.

Note that the tag is an empty tag; it does not use a pair of opening and closing tags since it does not surround any content. It only marks the page location of an image. Also, as an in-line tag, it must be coded inside a block-level tag such as a <p> tag. By default, the picture is aligned at the left margin of the page.

The JPEG image shown in below is assumed to be located in the same directory as the Web page that displays it. The image is placed on the page at the location of the tag with the accompanying code. (Move your mouse pointer on top of the image to view its pop-up alt box.)

```
<p></p>
```



Notice that you do not physically place the image on the Web page. Rather, you include an tag containing a URL address for the image file. When the Web page is loaded into the browser this URL is used to locate the image file, which is then transmitted separately to the browser for display on the page. This is why image file sizes are important considerations. Displaying an image requires an additional data transmission from the Web server, one for each image appearing on the page. Therefore, try to keep the number of

images and their sizes as small as possible in order to reduce the number and length of transmissions needed to fill a Web page.

Deprecated Image Alignment

Horizontal placement of an image can be controlled by enclosing the `` tag inside a `<p>` tag which aligns the image to the left, to the right, or centered on the line. Paragraph alignment can use the deprecated `align="left|center|right"` attribute of the `<p>` tag. The picture below, for example, is centered on a line by itself and is preceded and following by blank lines with the enclosing `<p>` tag.

```
<p align="center"></p>
```



Later you will learn more about placing and aligning images on a page. For present purposes consider use of the paragraph's `align` attribute only as a quick fix for aligning images.

Deprecated width and height Attributes

A downloaded image may not be the exact size you need for your page. The best solution is to open it in a graphic editing program in order to size or crop the picture to the dimensions you need prior to placing it on the page. An alternative is to resize the picture dynamically, during browser display, by setting the `` tag's deprecated width and height attributes:

```
width="n"
```

```
height="n"
```

These *n* sizes are given in pixels. You need only to set one *or* the other dimension, but not both, to maintain proportionality. The browser automatically adjusts the other dimension to keep the picture in correct proportion.



Self Assessment

1. Which of this is the correct XHTML for a list?

- a. ``
- b. ``
- c. ``
- d. ``
- e. `<list><list>`

2. What is a correct XHTML tag for a line break?

- a. `<break/>`
- b. `
`
- c. `</br>`
- d. `<break>`
- e. `
`

r

3. What XHTML code is "well-formed"?

- a. `<p>A <i>short</i> paragraph`
- b. `<p>A <i>short</i> paragraph</p>`
- c. `<p>A <i>short</i> paragraph</p>`
- d. all of above
- e. none of these



Unit Summary

1. The internet is a world-wide collection of different heterogeneous networks interconnected by routers using TCP/IP. A Web page originates as a standard text file containing the information to be displayed along with formatting instructions for its presentation on the computer screen. These formatting instructions are written in a special markup language, so-called because it is used to "mark up" the information on the page to describe its arrangement and appearance when rendered in a Web browser.

2. XHTML formatting codes, or tags, surround the material to be formatted and are enclosed inside angled brackets (" $<$ " and " $>$ ") to identify them as markup instructions. On the basis of these markup tags, the browser displays the page in the specified layout and style. HTML and XHTML are most identical, but XHTML is compliant with the XML specification, so certain rules must be followed when writing XHTML documents.



Unit Assignments 1

Develop a web site for your former school

CASCADING STYLE SHEETS, SPANS AND DIVISIONS

Introduction

The preferred way to apply styling to page elements is with Cascading Style Sheets (CSS). A style sheet is a set of style properties and accompanying style values that describe the appearance of XHTML elements to which they are applied.



Learning Objectives

After reading this unit you should be able to:

1. Describe the types of cascading style sheets
2. Apply cascading styles to web pages.

Unit content

Session 1-2: Cascading Style Sheets

- 1-2.1 Creating style sheets
- 1-2.2 Applying the Style sheets

Session 2-2: Spans and Divisions

- 2-2.1 Uses of span and divisions
- 2-2.2 The span tag
- 2-2.3 The div tag

SESSION 1-2: CASCADING STYLE SHEETS

1-2.1 Creating Style Sheets

There are three methods of creating style sheets. These are

- Inline style sheets
- Internal or Embedded Style Sheets
- Linked or External Style Sheets

The in-line style sheet appears inside the tag to which its style declarations apply; an embedded or internal style sheet is a separate style section of a Web page which applies its styles to all designated tags on a page; a linked or external style sheet is an external document containing style settings that apply to all pages that link to it.

Inline style sheets

An in-line style sheet is coded inside a tag to apply styling to that particular tag. It is coded in this general format.

```
<tag style="property:value [; property:value] ...">
```

One or more *property:value* pairs give style settings within a style attribute coded in the tag. Multiple style properties settings are separated from each other with semicolons; spaces can be included between the settings to help with readability.

The properties and values that can be coded for a tag depend to a large extent on the particular tag being styled. Style settings for a horizontal rule, for instance, can include height, width, color, and text-align properties, identical in effect to the deprecated size, width, color, and align attributes they replace. A style attribute appears inside the `<hr/>` tag and specifies a set of properties and values to apply to the tag when it is displayed in the browser.

```
<hr style="height:10px; width:50%; color:red; text-align:center"/>
```

When applied as in-line style sheets, the properties and values pertain only to the tag in which they are coded. This means, for instance, that if there are several `<hr/>` tags on a page each would need to include the same in-line style sheet to share the same style. This may not present a problem with only a few tags; with many tags, though, it can be tedious and error-prone to code the same style sheet in all of the individual tags. A way is needed, then, to declare style settings *one time* and then share these settings among multiple tags.

Internal or Embedded Style Sheets

To avoid duplicate coding of in-line style sheets an embedded style sheet can be used. An embedded style sheet is defined within a `<style type="text/css">` tag, normally coded in the `<head>` section of the page. Within this `<style>` section is a list of tag names, called selectors,

to which style declarations (properties and values) are assigned. Once these declarations are made, they are automatically applied to the identified tags wherever they are located on the page. The general format for the <style> section of a document is shown in Figure 3-5.

```
<style type="text/css">
  selector {property:value [; property:value] ...}
  ...
</style>
```

A *selector* is a tag name (*without* the enclosing "<" and ">" symbols). Style properties and values for the tag are enclosed within a list of style declarations, separated by semicolons, all enclosed within a pair of braces "{ }". For example, the following code specifies style declarations for the <hr/> tag, supplying the same style settings as in the previous in-line style sheet.

```
<head>
  <title>Embedded Style Sheet Example</title>
  <style type="text/css">
    hr {height:10px; width:50%; color:red; text-align:center}
  </style>
</head>
```

The hr selector is associated with four *property:value* declarations to style a horizontal rule. Once these styles are defined within an embedded style sheet they are applied automatically wherever the browser encounters an <hr/> tag in the document. It is not necessary to code separate in-line style sheets within each and every <hr/> tag. The tag itself carries the styling described in the embedded style sheet.

Although shown above with a single selector for the <hr/> tag, an embedded style sheet can contain any number of entries depending on how many different tags are to be styled. It can be coded on a single line. Many Web page authors prefer to code *property:value* settings on separate lines for ease of reading and editing as shown below.

```
<style type="text/css">
hr {height:10px;
    width:50%;
    color:red;
    text-align:center}
</style>
```


Linked or External Style Sheets

If your Web site contains multiple pages, and most do, then a third style sheet alternative is probably the best solution. It permits you to conveniently apply the same styles to *all* pages without having to duplicate in-line or embedded style sheets on each page.

A linked style sheet is a *separate document* containing *property:value* settings in the same format as an embedded style sheet. The only difference is that the linked document does *not* include `<style>` tags surrounding the entries. For example, a linked style sheet document containing specifications for the `<hr/>` tag described above includes the entries shown.

Stylesheet.css (document)

```
hr {height:10px;  
    width:50%;  
    color:red;  
    text-align:center}
```

This separate document is created with a text editor and saved as a standard text file, usually with the file extension `.css` to identify it as a style sheet document. It is common practice to save the document under the name `Stylesheet.css` in the same directory as the Web pages to which it applies.

A linked style sheet document contains style settings that are applicable to all pages of the Web site. Therefore, all pages that use the style sheet must "link" to it to make the styles available to those pages. This linkage occurs with a `<link>` tag coded in the `<head>` section of the page.

The general format for the `<link>` tag is shown below.

```
<link  
  href="url"  
  type="text/css"  
  rel="stylesheet"  
>
```

The `href` (hypertext reference) attribute gives the location of the linked style sheet. If the style sheet appears in the same directory as the Web page to which it applies, then this entry is simply the name of that document. The `type` attribute identifies the type of document to which the link is made (always `"text/css"`). The `rel` attribute specifies the relationship of the external document to the current page (always `"stylesheet"`).

If you have a style sheet document named `Stylesheet.css` that is located in the same folder as your Web page, you link the Web page to this document using the following XHTML code.

```
<head>
  <title>Linked Style Sheet Example</title>
  <link href="Stylesheet.css" type="text/css" rel="stylesheet"/>
</head>
```

Now the Web page has available to it all of the style settings included in the Stylesheet.css document. This linked style sheet can serve to replace embedded or in-line style sheets that otherwise would appear on individual Web pages. As in the case with embedded style sheets, any tags identified with selectors in a linked style sheet carry with them the declared styles.

1-2.2 Applying Style Sheets

It would not be uncommon to employ all three types of style sheets for a single Web page. A linked style sheet would contain styles that are common to all Web pages; an embedded style sheet would contain styles that are pertinent to the particular page; in-line style sheets would apply to individual tags for which common styling is not needed. The browser handles these multiple style sheets in the following fashion.

- First, any linked style sheets are applied to the identified tag selectors on the page.
- Second, any embedded style sheets are applied. If the same tag selectors appear in both the linked and embedded style sheets then embedded styles override or augment linked styles.
- Third, any in-line style sheets are applied. If these settings pertain to the same tags that appear in either linked or embedded style sheets, then in-line styles override or augment both linked and embedded styles.

The general principle in force is that any lower-level style setting takes precedence over an equivalent higher-level style setting. In-line style sheets take precedence over embedded style sheets, which take precedence over linked or external style sheets.

Assume, for example, that a linked style sheet contains the following style declarations for horizontal rules.

```
hr {height:1px;
    width:50%;
    color:red;
    text-align:center}
```

All pages that link to this style sheet display horizontal rules in this style. Now assume that one particular page needs differently styled rules, say, ones that are blue rather than red. So, an embedded style sheet is coded on this single page in order to *override* the color declaration in the linked style sheet:

```
<style type="text/css">
  hr {color:blue}
</style>
```

All horizontal rules on this page are blue; however, they retain the height, width, and text-align properties from the linked style sheet. Tags on this page inherit the properties of the linked style sheet unless modified by the embedded style sheet.

Assume further that one particular horizontal rule on this page needs to be green and extend to the width of the page. So, an in-line style sheet is added to this particular rule.

```
<hr style="color:green; width:100%" />
```

Green styling overrides the blue color that is inherited from the embedded style sheet; a width of 100% overrides the 50% width inherited from the linked style sheet. This particular rule, however, retains its 1 pixel height and centered alignment inherited from the linked style sheet.

Style inheritance is what makes style sheets "cascading." Linked styles are inherited by, or cascade across, all Web pages that link to them; embedded styles are inherited by, or cascade across, an entire Web page containing a <style> section; in-line styles are inherited by, or cascade across, individual tags containing in-line style sheets.

Margin Styles

When text is displayed in the browser window it spans from border to border with about a quarter of an inch margin surrounding the text on the page. You may have noticed these margins on Web pages created earlier. They are shown in the simple single-paragraph Web page in Figure 11.

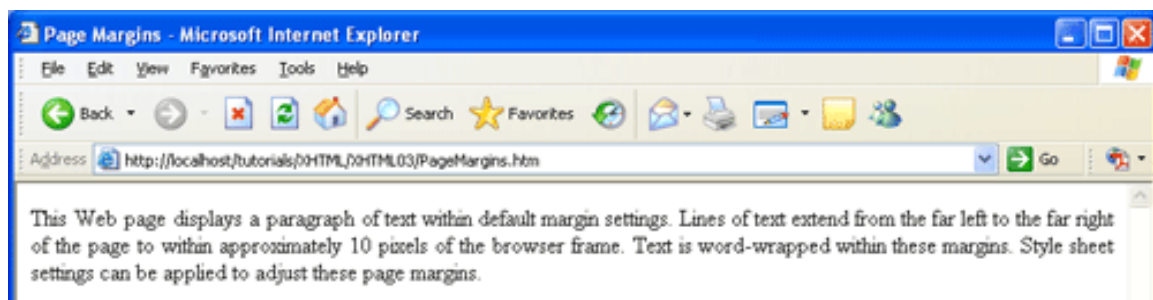


Figure 11: Web page with default margins

In some cases these margin settings work perfectly well. However, if there is a large amount of text on the page these narrow margins make for a crowded look and they increase the difficulty of reading the page. It can become tiresome plodding through line after line of text

spanning from border to border. A Web page is more inviting, is more readable, and is easier on the eyes if more white space appears around the text. In other cases, however, you may wish to reduce or remove default margin settings so that certain page elements appear even closer to the window borders.

Recall that the `<body>` tag encloses all page content that appears in the browser window. This tag can also be used to control margins around the borders of the page. The way to do this is to apply a style sheet to the `<body>` tag to adjust page margins.

Margin Style Properties

There are five style properties that can be used to set margins around page elements. These properties specify the amount of space to leave between the outside edges of the element and any surrounding content. When applied to the `<body>` tag they give the amount of white space to leave around displayed content on the page.

Property	:	Values
margin		<i>npx</i>
margin-top		<i>n%</i>
margin-right		auto
margin-bottom		
margin-left		

The margin property sets the same margin width around *all* sides of the element; the margin-top, margin-right, margin-bottom, and margin-left properties set margins individually for each of the four sides. The auto setting causes margins to revert to their default widths when overriding previous margin settings.

Measurement Values

The margin properties, like various other style properties, require a measurement value. In most cases you are required to specify whether the measurement is in pixels (px) or as a percentage (%) of the width of the browser window. These are the two most common measurements of distance on a computer screen. If you do not specify the measurement type, then pixels are assumed.

Applying Margins with an In-Line Style Sheet

You can set margins surrounding the content on a Web page by applying a style sheet to the `<body>` tag. The following in-line style sheet, for example, sets a 25-pixel margin around the entire page, using the margin property and specifying 25 pixels of white space between the contents on the page and the borders of the browser window.

```
<body style="margin:25px">
```

```
<p>This Web page displays a paragraph of text within margin settings of  
25 pixels surrounding the page. These margins are produced by an in-line  
style sheet applied to the body of the document. Lines of text extend  
between and are word-wrapped within these left, right, top, and bottom  
margin settings.</p>
```

```
</body>
```

This styling produces the page display shown. The text paragraph appears less crowded than in Figure 11, making it a more inviting read and making the text more readable by shortening the lines.

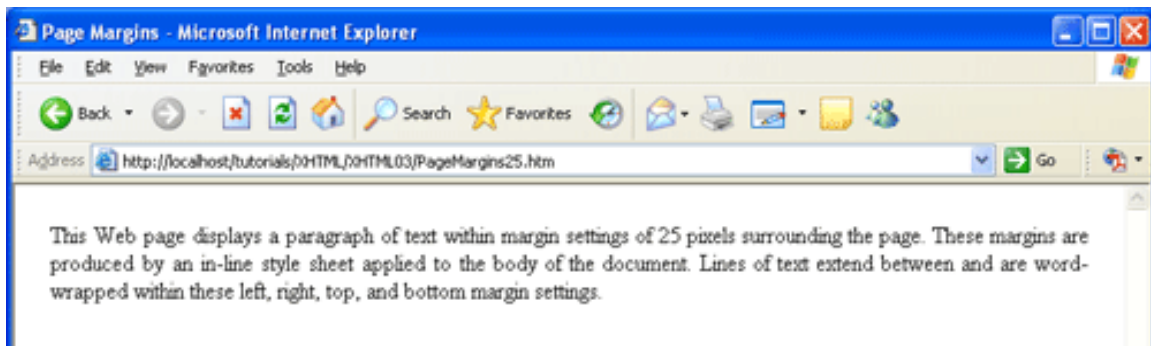


Figure 12: Web page with 25-pixel margins surrounding the page

Applying Margins with an Embedded Style Sheet

In the above example the `<body>` tag includes an in-line style sheet to set margins for this particular page. You can, instead, code an embedded style sheet to produce the same results. The following `<style>` entry supplies a body selector along with a margin style property and value to apply to this tag.

```
<head>  
  <title>Page Margins</title>  
  <style type="text/css">  
    body {margin:25px}  
  </style>  
</head>
```

```
<body>
```

```
<p>This Web page displays a paragraph of text within margin settings of  
25 pixels surrounding the page. These margins are produced by an embedded  
style sheet applied to the body of the document. Lines of text extend
```

between and are word-wrapped within these left, right, top, and bottom margin settings.</p>

</body>

The <body> tag, through the body selector, takes on the 25-pixel margin setting without having to code a style sheet inside the tag itself. Also, by using an embedded style sheet, style settings are isolated within a separate section of the page, making styles easy to locate and change if necessary. Your first instinct, in fact, should be to create an embedded style sheet, adjusted to in-line or linked style sheets as situations dictate.

Applying Margins with a Linked Style Sheet

If you have a Web site with multiple pages, and all pages need to share the same body margins, then a linked style sheet should be used. Place body styling in this separate document and link to it from all pages to which the style should apply.

The following code shows a Stylesheet.css document in which margin styling for the <body> tag is specified. This document appears in the same Web directory as the pages which share this style.

```
Stylesheet.css  
body {margin:25px }
```

Once this style sheet document has been created any Web page can apply its margin settings by linking to it. Code to adopt this linked style sheet is shown below.

```
<head>  
  <title>Any Page</title>  
  <link href="Stylesheet.css" type="text/css" rel="stylesheet"/>  
</head>  
...
```

Applying Individual Page Margins

In the above examples the margin property is applied to set the same margin widths surrounding all four side of the page. You can, instead, selectively apply different widths to each of the sides by using individual margin properties. In the following embedded style sheet these different margin settings are applied to a page.

```
<style type="text/css">
  body {margin-top:50px; margin-left:30px; margin-right:30px;
        margin-bottom:50px}
</style>
```

Note that the properties can appear in any order without affecting their application. Just make sure that *property:value* pairs are connected with a colon (:) and that declarations are separated with semicolons (;).

Deprecated <body> Attributes

Although not recognized under XHTML there are margin attributes, rather than margin properties, that can be used to set margin spacing for a page. These deprecated <body> attributes are listed below.

```
leftmargin="n"
rightmargin="n"
topmargin="n"
bottommargin="n"
```

```
marginwidth="n"
marginheight="n"
```

Margin settings are given by a value (*n*) representing the number of pixels of blank space to leave between the page content and the borders of the browser window. Settings apply individually to each of the borders (leftmargin, rightmargin, topmargin, bottommargin), to both sides (marginwidth), or to both the top and bottom (marginheight) margins of the page, for example,

```
<body leftmargin="30" rightmargin="30" topmargin="30" bottommargin="30">
```

or

```
<body marginwidth="30" marginheight="30">
```

Although these deprecated <body> attributes still work in modern browsers--and you are likely to encounter them on existing Web pages--you should not use them when creating your own pages. Rely instead on margin style properties using linked, embedded, or in-line style sheets.

Paragraph Margins

Margin styles are applicable to many XHTML tags, not just to the <body> tag. In fact, any container tag can take on margin settings. The <p> tag is case in point. By declaring margin, margin-top, margin-right, margin-bottom, and/or margin-left style properties for this tag, a paragraph can have its margins adjusted on all sides of the text block or on each side individually.

The following code displays three paragraphs, the middle one of which uses an in-line style sheet to set its left and right margins to 40 pixels.

`<p>Here are three paragraphs. This first paragraph is formatted with a standard paragraph tag with default style settings.</p>`

`<p style="margin-left:40px; margin-right:40px">This second paragraph is formatted with the style setting "margin-left:40px; margin-right:40px" to adjust its left and right margins to offset the paragraph from surrounding paragraphs. This styling produces a paragraph similar in style to a blockquote tag.</p>`

`<p>This third paragraph is coded like the first one. It is enclosed within a standard paragraph tag containing default styles.</p>`

Browser output of this coding is shown in Figure 13.

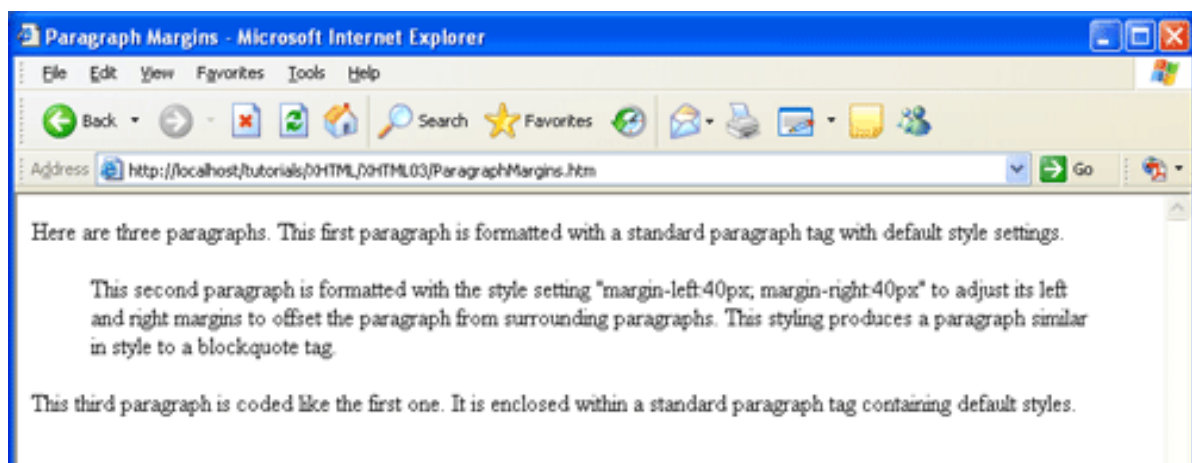


Figure 13: Setting paragraph margins

An in-line style sheet is used in this example since margin settings are applicable only to the single paragraph. There is no need to define an embedded style sheet or to create a linked style sheet document since this styling is not shared across all paragraphs.

Vertical Margins

It was mentioned previously that multiple `
` tags can be used to increase the amount of vertical spacing on a page. Each `
` tag beyond the one used to end a line of text produces an additional blank line down the page.

This same effect can be achieved with margin-top and margin-bottom style settings. By specifying these margin widths at the top and bottom of a text block the given number of pixels of blank space are inserted before and after the text, producing an effect similar to inserting blank lines. The following code, for example, inserts additional spacing before and after an indented paragraph by increasing its top and bottom margins with an in-line style sheet. The effect is the same as shown previously for a blockquoted paragraph offset with additional `
` tags. Browser output is shown in Figure 14.

`<p>Here is a tale about Mary and a pesky little lamb which followed her
anywhere and everywhere she went.</p>`

```
<p style="margin-left:40px; margin-right:40px;  
    margin-top:50px; margin-bottom:50px">  
Mary had a little lamb,<br/>  
Its fleece was white as snow;<br/>  
And everywhere that Mary went,<br/>  
The lamb was sure to go.<br/>  
</p>
```

`<p>Mary had an awkward social life. It's awfully difficult to date with sheep
trailing around after you all the time.</p>`

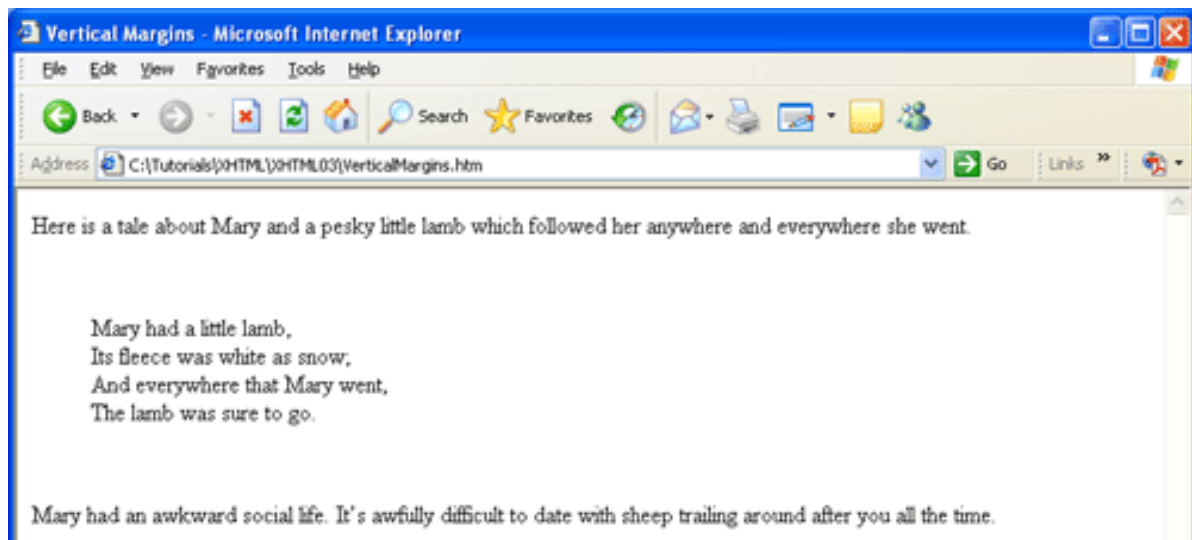


Figure 14: Using margin settings to increase vertical spacing

You can use either multiple `
` tags or extended margin settings to produce vertical spacing. Greater pixel precision, however, is gained by applying margin-top and margin-bottom styles. As you encounter additional tags throughout these tutorials keep in mind that they can be assigned margin properties to adjust the amount of space between their contents

and surrounding elements on the page. Margin settings are one of the main techniques used to reduce crowding of content on a Web page. There will be recurring need to apply margin properties to all manner of XHTML tags.

Alignment Styles

Elements on a Web page are normally blocked at the left margin of the page or at the left margins established by their margin or margin-left style properties. It is often the case, though, that you wish to change this default alignment and to align a page element at the right margin, to center it horizontally on the page, or to "float" the element to the left or right to permit text flow around the sides of the element.

The text-align Style

The text-align style property permits left, center, right, or justified alignment of page elements. Its property settings

Property	:	Values
text-align		left center right justify

The text-align style property is replacement for the deprecated align attribute previously used for <p>, <h1>, and <hr/> tags. From this point forward all horizontal alignment should be done with the text-align property.

Paragraph Alignment

A <p> tag surrounds a block of text which has its individual lines blocked at the left margin of the page or at the left margin established by the paragraph's margin or margin-left style setting. A text-align style can be applied to the paragraph to change this default left-alignment display.

The following code creates four paragraphs, each with an in-line style sheet to apply different text alignments. Browser display of these paragraphs is shown in Figure 15.

```
<p style="text-align:left">
```

This paragraph is formatted using the style="text-align:left" style setting. Each line of text is blocked at the left margin of the page and wraps at the right margin of the page. This is the default alignment style for a paragraph.</p>

<p style="text-align:center">

This paragraph is formatted using the style="text-align:center" style setting. Each line of wrapped text appears centered between the left and right page margins.</p>

<p style="text-align:right">

This paragraph is formatted using the style="text-align:right" style setting. Each wrapped line of text is aligned at the right margin of the page.</p>

<p style="text-align:justify">

This paragraph is formatted using the style="text-align:justify" style setting. All wrapped lines of text except the last line are expanded to reach both left and right margins. The final line, since it is not long enough to wrap, is not justified between the margins of the page.</p>

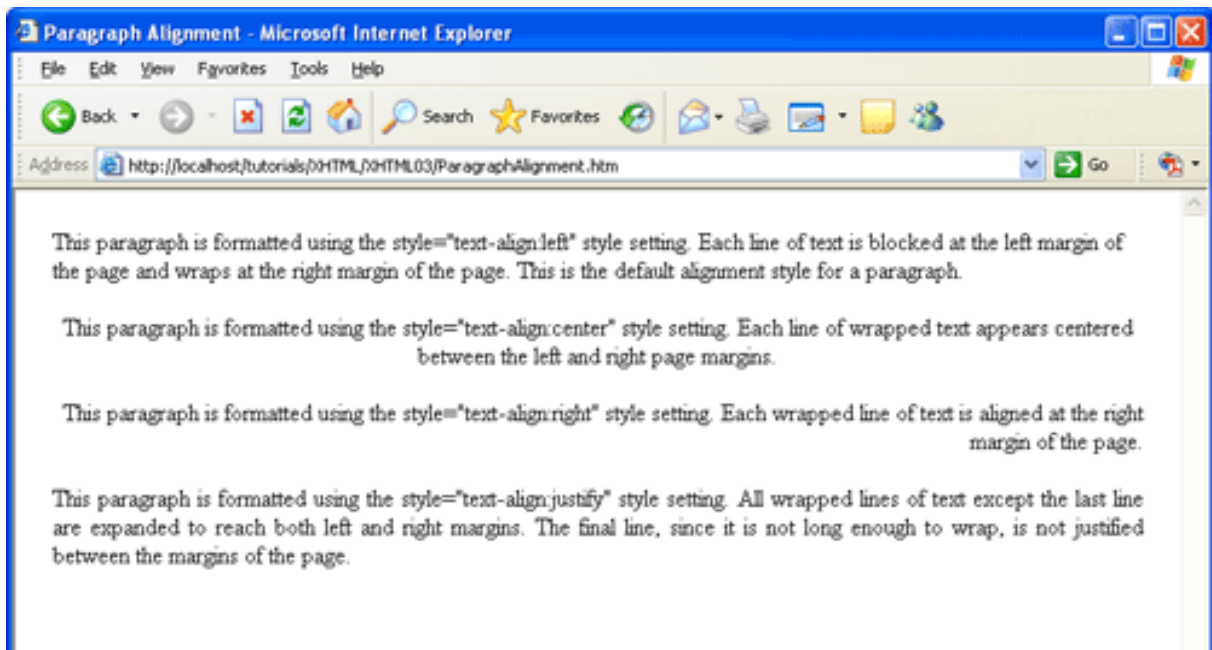


Figure 15: Various paragraph alignments displayed in browser window

In this example all four paragraphs have different styles. Thus, <p> tags are styled with in-line style sheets rather than with an embedded or linked style sheet which would produce common styling for all paragraphs. Keep in mind, though, that text alignment which should be shared by *all* paragraphs on a page should appear one time only in an embedded or linked style sheet. If it is decided, for example, that all paragraphs on a page will have justified lines then an embedded style declaration for the p selector is appropriate.

```
<style type="text/css">
  p {text-align:justify}
</style>
```

If it is further decided that all paragraphs on *all* pages of a Web site will have justified paragraphs, then this embedded style sheet can be promoted to a linked style sheet for sharing among all pages.

Heading Alignment

Text alignment styles can be applied to `<h>` tags to place headings at the left, in the center, or to the right of a page. In the following example the text-align property is applied to an `<h2>` heading with an in-line style sheet to center it horizontally on the page. This style property replaces the older `align="center"` attribute previously used to align headings.

```
<h2 style="text-align:center">Centered Heading</h2>
```

It is normally the case that various headings used on a Web page will have common styles in order to enforce a common look for the page. Thus, it is common practice to include heading alignment in an embedded style sheet for sharing among all like headings. The following code gives alignment styling for three heading sizes that appear on a page. In addition, top margins are increased to leave additional vertical blank space between the headings and preceding text.

```
<style type="text/css">
  h1 {text-align:center; margin-top:20px}
  h2 {text-align:left; margin-top:20px}
  h3 {text-align:left; margin-top:20px}
</style>
```

Although text alignment is not required for the above `<h2>` and `<h3>` tags (since left is the default alignment) these settings are included in the style sheet to make them explicit and to document the settings. Also, since all heading alignments appear in this embedded style sheet, any changes to heading alignments can be made at this single location for propagation across all headings on the page.

Image Alignment

Recall that image alignment takes place by enclosing an `` tag inside a container tag which aligns the enclosed picture. This alignment can be effected by enclosing the `` tag inside a `<p>` tag to which the text-align style is applied. A paragraph's enclosed picture is placed on a line by itself and is aligned at the left margin of the paragraph, at the right margin, or centered on the page. For example, coding to center an image on the page is given in the following in-line style sheet applied to a `<p>` tag.

```
<p style="text-align:center"></p>
```

Note that even though the property name is "text"-align, any type of content enclosed within the paragraph, text or otherwise, is centered on the line.

Floating Images

An alternative to positioning a picture on a line by itself is to place it at the left or right margin and permit word wrap around it. The picture is said to "float" to the left or right of its accompanying text. Floating an image makes use of the float style property.

Property:	Value
float	left right none

The picture below appears at the right margin of the page (floated right). The accompanying text paragraphs wrap around the image. All styling is contained in an embedded style sheet.



Figure 16: A floated image

```
<head>
  <title>Floated Image</title>
  <style type="text/css">
    body {margin:20px}
    p {text-align:justify}
    h2 {text-align:center}
    img {float:right; margin-left:25px}
  </style>
</head>
```

<body>

<h2>Floating Images</h2>

<p>

An alternative to positioning a picture on a line by itself is to place the image at the left or right margin to permit word wrap around it. The accompanying picture is floated to the right of the text on this page.</p>

<p>Remember that it is important to code the tag immediately before any text or other page elements that are to be wrapped around the image. The image is floated at the exact location on the page at which the tag is coded. The current tag is placed at the beginning of the first paragraph; therefore, it is floated to the right of that paragraph.</p>

</body>

Notice that the float property is associated with the img selector in the above style sheet. It is a property of the tag -- the tag to be floated -- unlike the case where an image is placed on a line by itself and is aligned by the text-align property of its enclosing <p> tag.

Images that are floated to the left or right of the page must be coded *immediately before* any text that wraps around the image. In the above example the tag appears at the beginning of the paragraph that wraps around the image. No line breaks appear between the image and its surrounding content. The tag is assigned the float:right style property to place it at the right margin of its enclosing paragraph. It also has a margin-left: 25px style setting in order to leave white space between the picture and its surrounding text on the left.

Images can be floated to the left or right of the page; they cannot be placed in the middle of the page with word wrap around both sides. When floating images it is usually a good idea to include additional white space between the image and its surrounding text by extending its margins. Like most style properties the float property can be applied to many other tags. In later tutorials you will learn how to float text containers to the left or right of a page.

In the above example, all tags are styled with an embedded style sheet. Although not shown here, any other <p>, <h2>, and tags appearing on this page would take on these same styling characteristics.

The text-indent Style

Previous examples of left-aligned paragraphs show them in “blocked style” wherein all lines of text are blocked at the left margin. It is common practice, however, to indent the first line of a paragraph as a visual clue that this is the beginning of a new paragraph. First-line indentation is not really necessary when paragraphs are separated by blank lines as is the case with Web pages; still, you may wish to apply this formatting to soften the look of your pages.

The text-indent style property is used to indent the first line of a paragraph. The amount of indentation is given in number of pixels or as a percentage of the width of the page.

Property:	Value
text-indent	<i>n</i> px <i>n</i> %

The following code indents the first line of two paragraphs, the first by 25 pixel and the second by 5% of the width of the page. Browser output is shown in Figure 17.

```
<p style="text-indent: 25px">This paragraph is formatted using style="text-indent: 25px". The first line of the paragraph is indented 25 pixels from the left margin. The remaining lines of the paragraph are blocked at the left margin and wrap at the right margin.</p>  
<p style="text-indent:5%">This paragraph is formatted using style="text-indent:5%". The first line of the paragraph is indented 5% of the page width from the left margin. The remaining lines of the paragraph are blocked at the left margin and wrap at the right margin.</p>
```

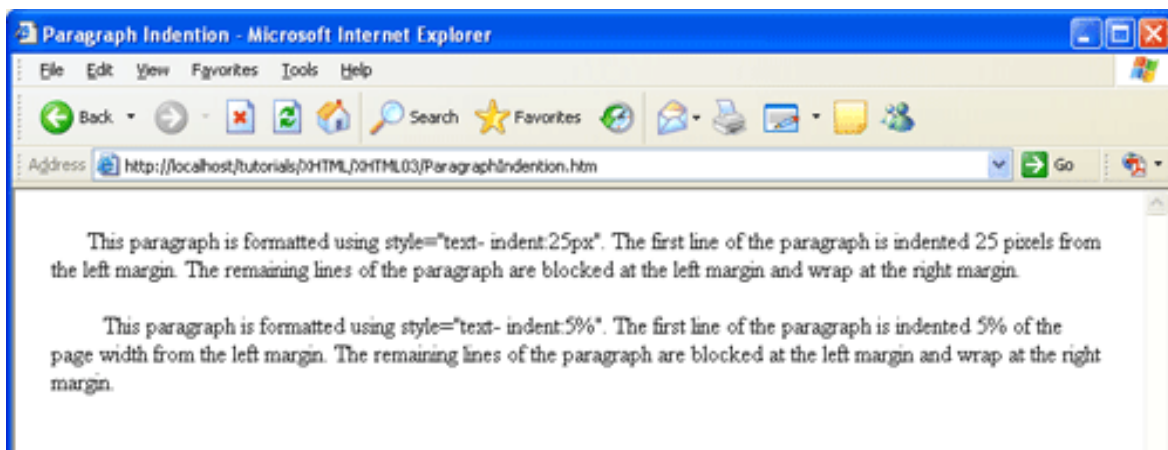


Figure 17: Text indentions applied to paragraphs

In this example both pixel and percentage values produce approximately the same amount of first-line indentation. However, when using a percentage measure for paragraph indentation, remember that this value is based on the width of the page. Different amounts of indentation will occur depending on the width of the browser window. In order to maintain a fixed amount of indentation regardless of window size uses a pixel value rather than a percentage.

Width and Height Styles

There are two general width and height style properties that can be used to size any page element. The width property gives the width of an element in pixels or as a percentage of the page width. The height property sets the element's height in pixels or as a percentage of page height.

Property	:	Values
width		<i>npx</i>
height		<i>n%</i>

Rule Widths and Heights

A Web page can be visually separated by displaying horizontal rules between them. This is done with the `<hr/>` tag to draw a line across the page.

The default rule is 2 pixels in height and spans the width of the browser window. You can change these dimensions by coding width and height styles for the `<hr/>` tag. This sizing is shown by the following in-line style sheet and browser display.

```
<hr style="width: 75%; height: 7px; text-align:center"/>
```

When specifying widths of page elements it is often preferable to code them as percentages. In this way the widths always appear in correct proportion to other elements on the Web page, even if the user adjusts the size of the browser window. If a horizontal rule is sized to less than the width of the browser window then it also can be aligned with `text-align:left`, `text-align:center`, or `text-align:right` to position it horizontally on the line. In the previous example the rule is 75% of the width of the page and is centered.

Normally all horizontal rules on a page will be of the same style. Therefore, it makes sense to code this styling in an embedded or linked style sheet so that it can be coded one time for application to all rules on the page or all rules at the site. The following code is an example of how horizontal rules can be standardized for a page with an embedded style sheet.


```
<style type="text/css">
  hr {height:1px; width:50%; text-align:center}
</style>
```

With this styling, all `<hr/>` tags on the page produce rules that are 1 pixel in height, 50% of the width of the browser window (irrespective of the window size), and centered horizontally on the page.

Image Widths and Heights

It is often the case that graphic images are not the exact widths or heights needed to fit comfortably on your Web page. This is a recurring problem if you do not possess, or do not know how to use, imaging software to resize the image prior to placing it on the page. However, by applying width and height style properties to these images they can be resized on-the-fly when displayed in the browser.

The actual size of the image below is 162 x 108 pixels. When displayed on the page its width is set to 100 pixels with an in-line style sheet.

```

```



Figure 18: A dynamically resized images

When resizing images, set *either* the width or height property; usually not both. The browser automatically adjusts the other dimension so as to maintain proper proportion. You can, of course, set both properties independently if you know their correct resized dimensions or if you wish to create special visual effects with disproportionate widths and heights.

Even though an image can be automatically resized when displayed in the browser it still retains its original file size. An extremely large image still can take up considerable disk space and can cause delays in downloading prior to its display. Ideally, you should edit a large image to reduce its actual size to the dimensions of its page display rather than using width and height styling to reduce only its display size.

An image should not normally be displayed at larger dimensions than its original size. When increasing the size of an image you risk losing resolution and producing a fuzzy “pixelated” image.

Recall from a previous tutorial that the `` tag has deprecated width and height attributes for resizing images. These attributes should be replaced by their comparable width and height style properties for concordance with XHTML standards. Although their names are the same, the way in which attributes and styles are applied is different.

The width and height style settings can be applied to any XHTML tag, not just to `<hr/>` and `` tags as in the previous examples. In subsequent tutorials there will be recurring occasion to resize other page elements to fit them with exact dimensions on the page.

Basic Color Styles

Colors can be applied to various page elements with the color and background-color properties. Color values can be given by a color name, a hexadecimal value representing the combination of red, green, and blue hues to be applied, or by an RGB (red, green, blue) decimal value. For example, color specifications can take the following forms:

```
color:red
color:#FF0000
color:rgb(255,0,0)
```

There are 16 basic color names recognized as standard Windows colors and shown below. Colors can be applied to page elements with the style settings `color:name` or `background-color:name` coded in the style sheet associated with the element

Property	:	Values
color		aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive,
background-color		purple, red, silver, teal, white, and yellow

Text and Background Colors

Text colors are applied by assigning a color value in the style sheet associated with the container tag enclosing the text. For instance, a heading can be displayed in blue by assigning this color name in an in-line style sheet for the `<h1>` tag.

```
<h1 style="color:blue">A Blue Heading</h1>
```

If an entire paragraph is to have a particular text color, then that color name can be applied with a style sheet associated with its `<p>` tag.

```
<p style="color:red">...red paragraph text...</p>
```

In certain cases you may wish to display all text on a page in a particular color. This is easily done by assigning the color property to the <body> tag. The following code displays all text on a page in green by using an embedded style sheet associated with the body selector.

```
<style type="text/css">
  body {color:green}
</style>
```

Any text container can also be given a background color by applying the background-color property. Just make sure that the color chosen is complementary to the text color and does not make it difficult to read the text. In the following style sheet, yellow is used as the page background color behind its green text.

```
<style type="text/css">
  body {color:green; background-color:yellow}
</style>
```

Rule Colors

The same color styling can be applied to horizontal rules. In the following example, rules are displayed with various style properties to produce different colors, sizes, and page positions. At the same time, the foreground (text) color of the page is blue and the background color is yellow.

```
<body style="color:blue; background-color:yellow">
<p>A blue horizontal rule 10 pixels in height, 300 pixels in width, and
aligned at the right margin:</p>
  <hr style="color:blue; height:10px; width:300px; text-align:right"/>
<p>A red horizontal rule 2 pixels in height, 75% of the page width, and
centered:</p>
  <hr style="color:red; height:2px; width:75%; text-align:center"/>
<p>A green horizontal rule 10 pixels in height; 25% of the page width, and
aligned at the left margin:</p>

  <hr style="color:green; height:10px; width:25%; text-align:left"/>

</body>
```

In this example, all rule styling is applied with in-line style sheets. The individual rules have different settings for which an embedded style sheet with a single hr selector is not appropriate. Body styling also uses an in-line style sheet, although in this case an embedded

style sheet with a body selector could have been used to apply foreground and background colors.

Colours can be applied to many different page elements besides text and rules. Also, many different colour combinations can be used to produce a full spectrum of colours. Extended discussion of colour values and styling is saved for a later tutorial. Use of the basic colour names can get you started in introducing colour variety to your pages.

1-2.3 Style Selectors and classes

Use of embedded style sheets has been suggested as the normal way to apply page-wide styles to tags. Where particular tags need to take on special styles different from these page-wide styles, in-line style sheets are used to modify the styling.

Simple Selectors

By way of review, most embedded styling can be done with individual tag selectors and their associated style declarations. These simple selectors declare default tag styling for an entire page. The general format for a simple selector is

selector {property:value [; property:value] ... }

Any number of selectors can be combined with any number of style properties within an embedded style sheet. For example, the following simple selectors set page-wide formatting for page margins, headings, and paragraphs. Anywhere one of these tags is encountered on the page, the browser applies the associated style.

```
<style type="text/css">
  body    {margin:20px; color:black}
  h1      {color:blue; text-align:center}
  h2      {color:blue; text-align:left}
  p       {text-align:justify; text-indent:25px}
</style>
```

There are likely to be occasions, though, where these page-wide styles need to be modified for particular tags. For example, one or more of the paragraphs in the document may require different alignment or indentation, or a particular heading may use a different style to emphasize it differently from other headings. Although you can use in-line style sheets to override these embedded style declarations, there are ways to designate exceptional styling within the embedded style sheet itself.

ID Selectors

One way to apply special styles to individual page elements is to provide the element with a unique identifier. Then, within the embedded style sheet, special styling can be applied only to the element with that identifier.

Assume, for instance, that common paragraph indentation and alignment are adopted for an entire Web page with the embedded style sheet declaration shown above. That is, all paragraphs have 25-pixel indentions and are text justified with first-line indents of 25 pixels. Now assume that one particular paragraph on the page requires different styling. This single paragraph needs to be offset from surrounding paragraphs with 25-pixel left and right margins and it should not have first-line indentation.

The first step in styling this particular paragraph is to assign it an id value. An id is a unique identifier for the tag. It can be any name you choose using combinations of alphabetic and numeric characters. In the following code, id="Special" is assigned to the paragraph requiring special formatting.

```
<p id="Special"> This paragraph requires special styling different from regular paragraphs on the page. It should be offset 25 pixels from both margins and not have first-line indentation.</p>
```

With an id assigned to this <p> tag it can be designated for special styling by using an ID selector in the embedded style sheet. The general format for an ID selector is

```
selector#id {property:value [; property:value] ...}
```

Styling is applied to the *selector*, but *only* to the selector with the designated (#) *id* value. Thus, an ID selector can be added to the above style sheet to format the special paragraph differently from regular paragraphs:

```
<style type="text/css">
  body    {margin:20px; color:black}
  h1      {color:blue; text-align:center}
  h2      {color:blue; text-align:left}
  p       {text-align:justify; text-indent:25px}
  p#Special {text-indent:0px; margin-left:25px; margin-right:25px}
</style>
```

The syntax p#Special refers to the <p> tag with the id value (#) of "Special". The standard 25-pixel first-line indentation given by the p simple selector is reset to 0 pixels for this particular paragraph. Also, the paragraph is given left and right margin settings different from normal paragraphs. Its text remains justified, however, as inherited from standard paragraph

styling given through the `p` selector. In other words, all paragraphs inherit the styling given by the `p` simple selector unless these styles are modified or supplemented by `p#id` selectors.

This type of tag identification and styling can be used for any tag. Just assign a unique id to the tag, and differentiate it from other tags of the same type by adding this `#id` value to the tag selector.

Styling Spans and Divisions

You should *not* style `` or `<div>` tags with simple selectors in an embedded style sheet. These tags often appear multiple times on a page to isolate and style different portions of text and different collections of page elements, all of which usually require different stylings. Thus, you cannot associate one particular style with these tags without reducing their flexibility in styling other spanned text or divisions that require different stylings. In other words, do NOT code the following simple selectors in embedded style sheets.

However, by using ID selectors with these tags you *can* apply different styles to different `` and `<div>` tags without committing them to one particular style.

In the following example, two paragraphs are given the same style by enclosing them inside a `<div>` to which special styling is applied. The style effects only this single division through its unique ID selector. Styling of other divisions is not affected since they do not have this id value. At the same time, two `` tags take on their unique stylings through their unique id values, leaving unaffected any other stylings of spanned text.

```
<style type="text/css">
  div#Justify {text-align:justify}
  span#Red   {color:red}
  span#Blue  {color:blue}
</style>
```

```
<div id="Justify">
<p>This paragraph is styled by inheriting the styling of its container
division which is formatted with the <span id="Red">"Justify"</span> style.</p>
```

```
<p>This paragraph is also styled by inheriting the styling of its container
division. In addition, <span id="Blue">"Blue"</span> styling is applied to
one of its words.</p>
</div>
```

1-2.4 Style Classes

By defining a class selector a general-purpose style can be declared for broad application to *any* tags needing to share the style. A class selector is defined in an embedded style sheet as shown by its general format

```
.class {property:value [ ;property:value] ...}
```

Rather than naming a tag or using an id in the selector, a *class* name is supplied, *preceded by a period*. This *.class* selector can be a name of your choosing but cannot include spaces or special characters. Any combination of style properties and values can be associated with this style class.

The following embedded style sheet includes a style class named *.Offset*. Once this class has been defined, it is assigned to any tag with the `class="class"` attribute. Below, the style class is applied to a paragraph by assigning it to the `<p>` tag. This paragraph takes on the styling of the *.Offset* class, in this case overriding normal styling given by the `p` simple selector.

```
<style type="text/css">
  p    {margin:20px}
  .Offset {margin-left: 30px; margin-right: 30px}
</style>
```

```
<p class="Offset">This paragraph requires special styling different from normal
paragraphs on the page. It has left and right margins of 30 pixels./p
```

(Note that the class assigned in the tag does *not* include the period that is necessary when declaring the *.class* selector in the style sheet.)

A style class is a general-purpose style that can be applied to any type of tag simply by assigning the class to the tag; plus, unlike the ID selector, the class selector can be assigned to any number of tags. Thus, any paragraphs, divisions, or other block-level tag types can reflect the above styling by assigning them to the *.Offset* class. Of course, the tag to which the class is applied must be receptive to the particular properties and values declared for the class.

Style classes are particularly relevant to `` and `<div>` tags for styling text strings and blocks of code without committing these tags to one particular style. The tags become carriers for many different styles simply by assigning different style classes to them. The following code is an example of applying various style classes through various tags to produce the page display shown below.

```
<head>
  <title>Class Styles</title>
```

```

<style type="text/css">
.Offset {margin-left:25px; margin-right:25px; text-align:justify}
.Red    {color:red}
.Blue   {color:blue}
.Rule   {height:2px; width:75%; text-align:center; color:green}
</style>
</head>
<body>
<hr class="Rule"/>
<div class="Offset">
  <p>Here is a paragraph that takes on the formatting given for its enclosing
  division. Within this paragraph the word <span class="Red">RED</span> has its
  own style class applied.</p>
  <p>This paragraph also is styled by its enclosing division. Within this
  paragraph the word <span class="Blue">BLUE</span> has its own style class
  applied.</p>
</div>
<hr class="Rule"/>
</body>

```

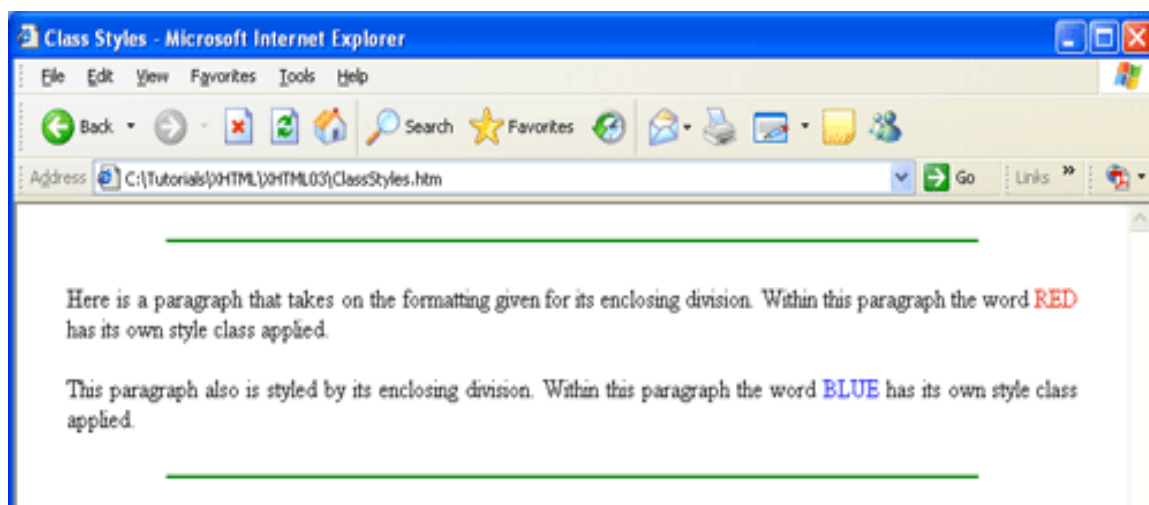


Figure 19: Application of class styles

Both of the above paragraphs are styled by enclosing them inside a division to which the .Offset style class is assigned. Thus, the paragraphs inherit division styling, which in this case offsets both paragraphs from the page margins and justifies their text. This technique has the same effect as declaring an ID selector for division styling (div#Offset) and applying the style through this ID selector (<div id="Offset">). However, the .Offset class is more flexible since it can be associated with other tags besides divisions. Any tags on the page can take on the .Offset style by assigning them to this class.

Within each paragraph are individual words that take on color styling. These words are enclosed inside `` tags through which colors are applied by assigning the tags to either the `.Blue` or `.Orange` style class. Incidentally, if any other sections of the page -- a paragraph, a heading, whatever -- need to be displayed in one of these colors then their enclosing tags can be assigned these same style classes. Any normal stylings of these tags are supplemented with assigned colors.

Both horizontal rules are 75% of the width of the page, 2 pixels in height, green, and centered. The choice is made to define a style class to represent this styling and to assign all `<hr/>` tags to this class. Of course, these styles could be defined for the simple `hr` selector without using a class. However, the choice is made to create classes for all styling and to assign tags to these classes as needs dictate. It would not be unusual to find an embedded style sheet composed exclusively of style classes that are selectively applied to all varieties of tags requiring those styles.

SESSION 2-2: SPANS AND DIVISIONS

2-2.1 Introduction

Up to this point, styles have been applied to individual page elements by associating style sheets with their tags. This technique works fine in most cases; however, there are styling situations not covered by this method.

For example, within a paragraph you might wish to apply a style to a particular letter, word, phrase, or other text string located *inside* the paragraph. Applying a style sheet to the `<p>` tag does not give you the ability to select which “substring” of the paragraph to style. All characters in the paragraph receive the same styling.

Another example is a consecutive *group* of paragraphs to which you want to apply the same style. You can, of course, duplicate and apply the same in-line style sheet individually to the separate `<p>` tags to give them the same appearance. However, it would be more convenient to be able to apply the same style to the paragraphs as a group, without having to style them individually.

For these and other styling situations involving strings of text less than a paragraph in length and blocks of text longer than a paragraph, XHTML provides “marker” tags to identify and isolate particular sections or subsections of a page to which styling can be applied.

2-2.2 The tag

A tag is an in-line tag placed around text for the purpose of identifying a string of characters to which this tag's style sheet is applied. The tag can enclose a single letter, a word, a phrase, a sentence, or any other substring of text for the purpose of identifying it for application of styling. As an in-line tag, the tag surrounds a string of text enclosed *inside* a block-level container.

In the following paragraph the words "RED" and "BLUE" are isolated with tags as words to which these tags apply their color properties. When this paragraph is displayed in the browser the two words are rendered in their associated colors.

```
<p>This paragraph contains the word <span style="color:red">RED</span> that is  
surrounded by a <span> tag to apply this color property to the word. In this sentence the word  
<span style="color:blue">BLUE</span> is given that  
color.</p>
```

A tag is nothing more than a marker to isolate text to which its style sheet can be applied. It has no built-in formatting characteristics of its own. It does not force a line break nor does it produce a blank space. Therefore, it can be used in-line, within the normal flow of text simply to style its enclosed content. Of course, the style that is applied must be appropriate to the enclosed text string. It would not be appropriate, for instance, to apply the text-indent property to indent the "first line" of a single word enclosed by a tag.

2-2.3 The <div>Tag

A <div> (division) tag is a similar type of marker to the tag. Its purpose is to enclose and designate a *collection* of page elements for application of styling to the enclosed set. The <div> tag is a block-level tag because it encloses other tags and, importantly, it forces a line break on the page because it creates a line break before and after its enclosed content.

Use of the <div> tag is illustrated by the following two paragraphs. Both paragraphs are styled the same. However, instead of coding these styles with in-line style sheets in both <p> tags, the paragraphs are surrounded by a <div> tag which applies these styles. The paragraphs *inherit* the styles of the enclosing <div> tag.

```
<div style="text-indent:25px; margin-left:30px; margin-right:30px;  
text-align:justify">
```

```
<p>This paragraph has first-line indentation of 25 pixels. It has both left  
and right margins of 30 pixel and its alignment is justified between the two  
margins.</p>
```

```
<p>This paragraph also has first-line indentation of 25 pixels. It has both
```

left and right margins of 30 pixel and its alignment is justified between the margins. Both paragraphs are styled with an enclosing division tag to apply these styles to both paragraphs.</p>
</div>

The <div> tag does not have any visible formatting characteristics of its own other than the fact that it creates a line break before and after its enclosed content. These line breaks are not evident in the above example since <p> tags produce their own line breaks, which are collapsed together with that produced by the <div> tag.

Since the <div> tag, like the tag, provides great flexibility in enclosing and styling page content, there are numerous occasions throughout these tutorials where spans and divisions are used to apply formatting to a wide assortment page elements.



Self Assessment

1. Cascading Style Sheets (CSS) can be defined for entire website by simply writing the CSS definitions
 - a. As Plain Text document
 - b. In the Head section of each page
 - c. In the Body section of each page
 - d. None
 - e. All of above
2. In CSS, which of the following options do you have for changing text color?
 - a. Common name and Hexcolor only
 - b. Common name, RBG value, Hexcolor
 - c. RBG value, Hexcolor
 - d. Hexcolor only
 - e. Common name only
3. Which of the following is NOT a valid CSS selector?
 - a. ID selectors
 - b. TAG selectors
 - c. Title selectors
 - d. HEAD selectors
 - e. None of the above
4. In CSS, a definition of fonts, colors etc is called
 - a. Font-family
 - b. ID tag
 - c. Style
 - d. Font colors
 - e. None of the above



Unit Summary

In this unit we have learnt the application of three methods of creating style sheets: Inline style sheets, Internal or Embedded Style Sheets, Linked or External Style Sheets

The in-line style sheet appears inside the tag to which its style declarations apply; an embedded or internal style sheet is a separate style section of a Web page which applies its styles to all designated tags on a page; a linked or external style sheet is an external document containing style settings that apply to all pages that link to it.



Unit Assignment

Create an external style sheet (with four CSS rules) for a web page.

TABLES, FRAMES AND FORMS

Introduction

Tabular arrangements of data into rows and columns can help the viewer sort through masses of data to understand their underlying structure and content. So, tables are useful devices for presentation of information in a meaningful form. At the same time, tables can be used to structure the layout of a Web page regardless of its content. You can produce a variety of page designs simply by designing different table structures into which your page information will fit.

Frames are separate "panes" inside the browser window within which different pages can be displayed at the same time.

An XHTML form provides a way for users to interact with a Web page. A form is, basically, a data capture device.



Learning Objectives

After reading this unit you should be able to:

1. Create tables to restructure web pages.
2. Use frames to divide a web page into regions.
3. Work with forms as a data capture device.

Unit content

Session 1-3: Tables

- 1-3.1 Creating tables
- 1-3.2 thead, tfoot and tbody tags

Session 2-3: Frames

- 2-3.1 Sets of frames
- 2-3.2 The frame set document
- 2-3.3 Nested frames

Session 3-3: Forms

- 3-3.1 Creating forms
- 3-3.2 Publishing files

SESSION 1-3: TABLES

1-3.1 Creating Tables

The `<pre>` tag is one method for arranging data into rows and columns for tabular presentation. A block of text surrounded by the `<pre>` tag is displayed in a monospace font in precisely the way it is typed in the text editor. The tag is often used to create simple tables where alignment of columns is produced with embedded blank spaces. The following editor code, for example, shows a table produced by using the keyboard Enter key to create separate rows and the Space Bar to align columns of data. This table layout appears on a Web page in exactly the format it is typed inside the `<pre>` tags.

```
<pre>
                Table 1
            Sales by Region
-----
Region/Year  2000   2001   2002   2003
-----
East          35.2   37.4   39.8   40.0
West          28.0   25.6   27.4   29.8
South         102.3   86.1   98.6   100.2
North          10.5    8.6    9.8    10.4
-----
</pre>
```

This method provides limited capabilities for arranging and styling a table. It is useful only for producing fairly simple tables without the need for styling it in conformance with overall page design. A better method is to construct tables using special XHTML tags designed for that purpose.

A table produced with XHTML table tags, like any other tabular presentation, contains rows and columns, the intersections of which are the cells of the table into which information is placed. A simple 3 x 3 table is

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

The rows, columns, and cell intersections of a table are defined with three basic tags. A `<table>` tag surrounds the entire table description; `<tr>` (table row) tags defined the rows of

the table; and <td> (table data) tags define the cells, or columns, that appear along each row. The <td> tag encloses the content that appears in a cell.

```
<table>
  <tr>
    <td>cell content</td>
    <td>cell content</td>
    ...
  </tr>
  ...
</table>
```

There are as many <tr> tags as there are rows in the table; there are as many <td> tags as there are cells (columns) in each row. When building tables you need to be cautious about properly pairing opening and closing tags. One minor oversight and the table arrangement can be totally disrupted. Also, take care to organize your coding so that the table structure is clearly evident. The following code, for example, is used to create the table shown above

```
<table>
<tr>
  <td>Cell 1.1</td>
  <td>Cell 1.2</td>
  <td>Cell 1.3</td>
</tr>
<tr>
  <td>Cell 2.1</td>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
</tr>
<tr>
  <td>Cell 3.1</td>
  <td>Cell 3.2</td>
  <td>Cell 3.3</td>
</tr>
</table>
```

Note that <table>, <tr>, and <td> tags are container tags, all requiring both an opening and closing tag. The <table> tag encloses all coding for the table; <tr> tags enclose each row of the table; <td> tags enclose each cell of a table row.

However, since table cells (<td>...</td>) can enclose information of varying length and complexity, it is best to code them on separate lines as in Listing 8-2 for more accurate visualization and typing of cell contents.

Table and Cell Borders

By default, table tags produce no borders around the table or around its cells. Data are aligned within rows and columns; however, no borders are displayed. In certain cases you may not wish to display table borders; in most cases you will. Therefore, style sheets must be applied to the table and to its cells to display borders around both.

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

A style sheet is applied to the table *and* to its cells to display borders around both. A border surrounds the entire table by styling the <table> tag; borders surround individual cells by styling the <td> tag. The code below produces the bordered table. The entire table is surrounded by a 1-pixel outset border and each of the cells is surrounded by a 1-pixel inset border. This styling is typical for a standard XHTML table.

```
<style type="text/css">
  table {border:outset 1px}
  td   {border:inset 1px}
</style>
```

```
<table>
<tr>
  <td>Cell 1.1</td>
  <td>Cell 1.2</td>
  <td>Cell 1.3</td>
</tr>
<tr>
  <td>Cell 2.1</td>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
</tr>
<tr>
  <td>Cell 3.1</td>
  <td>Cell 3.2</td>
  <td>Cell 3.3</td>
</tr>
</table>
```


Table Size

By default, the size of a table depends on the size of the data appearing in its cells. A column is as wide as the widest entry in the column; all rows are as wide as the combined width of data within the widest cells. In effect, table cells collapse around the data they contain. Although it is not apparent in the above examples, data are aligned horizontally left and vertically centered within the cells. Style settings to manage table and cell sizes and data alignment are described later in this tutorial.

Nested Tables

Tables can be nested; that is, a table can appear inside a cell of another table. This arrangement does not involve any particular coding complication--table specifications are simply inserted as contents of a cell. The cell expands in size to permit full display of the nested table as shown in the following table and XHTML code.

Cell 1	Cell 2				
Cell 3	<table><tr><td>Cell A</td><td>Cell B</td></tr><tr><td>Cell C</td><td>Cell D</td></tr></table>	Cell A	Cell B	Cell C	Cell D
Cell A	Cell B				
Cell C	Cell D				

```
<style type="text/css">
  table {border:outset 1px}
  td   {border:inset 1px}
</style>
```

```
<table>
<tr>
  <td>Cell 1</td>
  <td>Cell 2</td>
</tr>
<tr>
  <td>Cell 3</td>
  <td>
```

```
    <table>
    <tr>
      <td>Cell A</td>
      <td>Cell B</td>
    </tr>
    <tr>
      <td>Cell C</td>
      <td>Cell D</td>
    </tr>
  </td>
```

```
</tr>
</table>
```

```
</td>
</tr>
</table>
```

In this example you can also see the default alignment of data within cells. Notice in “Cell 2” that the entry is horizontally aligned at the left of the cell; in “Cell 3” the entry is vertically aligned in the middle of the cell.

Table Borders

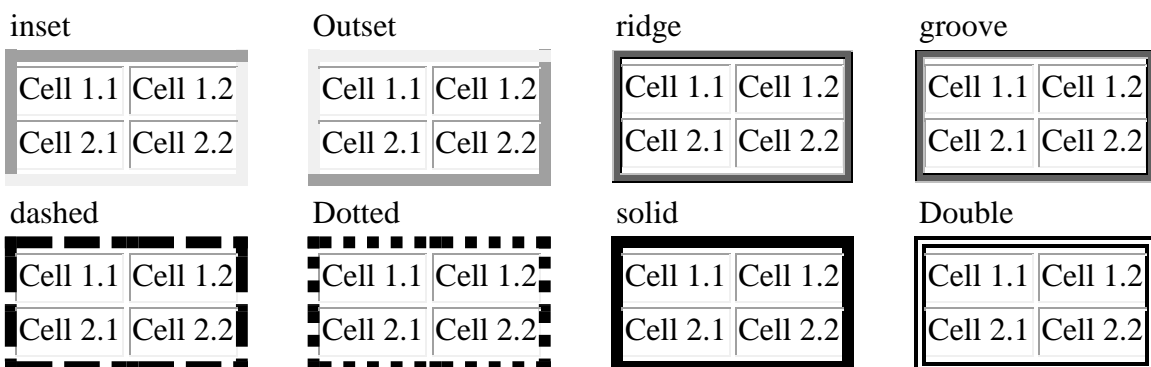
When a table is defined by a `<table>` tag with no styles specified, a borderless table as wide and as tall as its enclosed data is displayed. From this simple beginning you can structure and style a table to take on virtually any display characteristics you can imagine.

Table Border Styles

Any of the border styles that were introduced earlier can be applied to the outer border of a table.

```
<style type="text/css">
  table {border:style 5px}
  td   {border:inset 1px}
</style>
```

The border *style* is identified for each of the tables. The style sheet uses the general format `{border:style size}` for all border stylings.



All of the above tables have an outer border of 5 pixels to make them visually obvious. Different effects can be achieved with different border sizes.

Table Captioning

The <caption> Tag

You can caption a table by coding a <caption> tag immediately following the <table> tag. Enter a table title between the opening and closing <caption> tags and the title displays centered over the table. The caption is displayed in the page default font style and size; formatting is applied with style sheets as is done in the following code example

Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

```
<style type="text/css">
  table      {border:outset 3px}
  table td   {border:inset 1px}
  table caption {font:bold 10pt arial}
</style>
```

```
<table>
<caption>This is My Table</caption>
<tr>
  <td>Cell 1.1</td>
  <td>Cell 1.2</td>
  <td>Cell 1.3</td>
</tr>
<tr>
  <td>Cell 2.1</td>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
</tr>
<tr>
  <td>Cell 3.1</td>
  <td>Cell 3.2</td>
  <td>Cell 3.3</td>
</tr>
</table>
```

The <th> Tag

Column headings can be supplied with <th> tags in place of <td> tags enclosing cell contents. The <th> tag automatically centers and bolds the enclosed text. Additional styles

can be applied to the tags for additional formatting. Notice in the following example that the th selector is added to the style sheet for borders surrounding these cells.

This is My Table

Column 1	Column 2	Column 3
Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

```
<style type="text/css">
  table      {border:outset 3px; border-collapse:collapse}
  table td, th {border:inset 1px}
  table caption {font:bold 10pt arial}
</style>
```

```
<table>
<caption>This is My Table</caption>
<tr>
  <th>Column 1</th>
  <th>Column 2</th>
  <th>Column 3</th>
</tr>
<tr>
  <td>Cell 1.1</td>
  <td>Cell 1.2</td>
  <td>Cell 1.3</td>
</tr>
<tr>
  <td>Cell 2.1</td>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
</tr>
<tr>
  <td>Cell 3.1</td>
  <td>Cell 3.2</td>
  <td>Cell 3.3</td>
</tr>
</table>
```

The <th> tag is unique only in that it provides default bold and centered styling for its contents. With style sheets, however, this tag becomes redundant since this same styling can be applied without using this special tag. The following table duplicates the previous table without <th> tags simply by identifying and styling the first row of the table.

This is My Table

Column 1	Column 2	Column 3
Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

```
<style type="text/css">
  table      {border:outset 3px; border-collapse:collapse}
  table tr#HEAD {font:bold; text-align:center}
  table td    {border:inset 1px}
  table caption {font:bold 10pt arial}
</style>
```

```
<table>
<caption>This is My Table</caption>
<tr id="HEAD">
  <td>Column 1</td>
  <td>Column 2</td>
  <td>Column 3</td>
</tr>
<tr>
  <td>Cell 1.1</td>
  <td>Cell 1.2</td>
  <td>Cell 1.3</td>
</tr>
<tr>
  <td>Cell 2.1</td>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
</tr>
<tr>
  <td>Cell 3.1</td>
  <td>Cell 3.2</td>
  <td>Cell 3.3</td>
</tr>
</table>
```

The contextual selector `tr#HEAD td` indicates styling for `td` tags that appear inside a `tr` tag with `id="HEAD"`. It is your choice whether to use the `th` tag to take advantage of its inherent styling or to manage all styling through style sheets. As a general rule the fewer tags to have to deal with the better.

Table Colors and Backgrounds

Tables can have outer border colors and cell border colors. They also can have background colors, patterns, and graphic images as backdrops for the full table or for selected rows and cells.

Table Border Colors

The appearance of border colors is different depending on the border style. Below are example border styles displayed in red.

```
<style type="text/css">
  table {border:style 5px red}
  td   {border:inset 1px}
</style>
```

Table Size and Alignment

The sizes of cells in a table are governed by the size of the data within the cells. The table is as wide as the total of the widths of the widest cells. The table is as tall as the combined heights of the rows. In some case, however, you may want to specify a width and/or height for a table irrespective of the data it contains.

Table Widths and Heights

The overall table width is supplied by the width property; its height is given by the height property. Sizes can be measured in pixels -- to set exact sizes; or they can be expressed as percentages of the width and height of the browser window -- to vary the table size to correspond with the window size.

.

```
<style type="text/css">
  table {border:outset 1; width:50%; height:100px}
  table td {border:inset 1}
</style>
```

The table width is set to 50% of the page width. If you resize your browser window you can see that the table width remains half as wide as the page. The height is specified as 100 pixels and remains at that height irrespective of the window size. Although it is common to specify table widths for visual design purposes, table heights are normally permitted to expand in size for as many rows as it contains.

Column Widths

You can control column widths by specifying a size for any *one* cell in each column. This single cell width becomes the width to which all other cells in the column conform. The following table is sized to 50% of the width of the window. Then, cells in the first row of the table, containing column headings in this case, are assigned width percentages. These percentages are of the *width of the table*, not of the page. Therefore, the sum of column widths should add up to 100%, the full width of the table.

Column 1	Column 2	Column 3
Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

```
<style type="text/css">
  table      {border:outset 1; width:50% }
  table td   {border:inset 1}
  table tr#HEAD {font-weight:bold;
                  text-align:center;
                  background-color:#F0F0F0}
  table td#CELL1 {width:25% }
  table td#CELL2 {width:50% }
  table td#CELL3 {width:25% }
</style>
```

```
<table>
<tr id="HEAD">
  <td id="CELL1">Column 1</td>
  <td id="CELL2">Column 2</td>
  <td id="CELL3">Column 3</td>
</tr>
<tr>
  <td>Cell 1.1</td>
  <td>Cell 1.2</td>
  <td>Cell 1.3</td>
</tr>
<tr>
  <td>Cell 2.1</td>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
</tr>
<tr>
```

```

<td>Cell 3.1</td>
<td>Cell 3.2</td>
<td>Cell 3.3</td>
</tr>
</table>

```

It is not necessary to provide width percentages for all cells across a row. Any remaining cells on a row are sized equally by the browser so that the widths of all cells along the row total 100% of the width of the table. If cell widths are not sized sufficiently to display their data, the browser increases the widths to the minimum needed.

Rather than using percentages you can specify table widths in pixels. However, you should normally use percentages to give the browser sufficient latitude in sizing the table to fit within the available window width.

You can set rows heights in the same way you set column widths: specify the height for *one* cell along the row and the other cells expand to conform to that height. Unless you have a reason for expanding the height of a row you should let the browser determine the necessary height to display the data in the table.

Table Alignment

A Table		
Column 1	Column 2	Column 3
Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

Tables can be aligned to the left or right of the page by applying the float property to the table selector. When the table is floated, word wrapping takes place around the table, much like word wrapping takes place around floated images.

Code for a floated table appears immediately before the text to be wrapped around the table. Margins can be added to sides of the table to leave white space between the table and the wrapped text. We may apply the following partial style sheet applies to a floated table

```

<style type="text/css">
  table {border:outset 1; float:right; margin-left:20px; margin-bottom:10px}
  ...
</style>

```


A table can be aligned to the left or right, or centered on a line by itself. In this case, however, alignment styling is not associated with the table itself. The table needs to be enclosed inside a <p> or <div> tag (block-level tag) to which the text-align property is assigned. The table in below is coded inside a <p> tag which contain styling to center its contents.

A Centered Table		
Column 1	Column 2	Column 3
Cell 1.1	Cell 1.2	Cell 1.3
Cell 2.1	Cell 2.2	Cell 2.3
Cell 3.1	Cell 3.2	Cell 3.3

```
<style type="text/css">
p#CENTER    {text-align:center}
table      {border:outset 1}
table td   {border:inset 1}
table caption {font-weight:bold}
table tr#HEAD {font-weight:bold;
               text-align:center;
               background-color:#F0F0F0}
</style>

<p id="CENTER">
  <table>
  <caption>A Centered Table</caption>
  ...
  </table>
</p>
```

Column and Row Spanning

Table cells can be combined to make one larger cell from two or more contiguous cells. A cell can span two or more columns or two or more rows. One use of spanning is to display a heading across several columns as shown by the heading cells in the following table.

Heading 1		Heading 2	
Cell 2.1	Cell 2.2	Cell 2.3	Cell 2.4
Cell 3.1	Cell 3.2	Cell 3.3	Cell 3.4

The colspan Attribute

In the above example the first row of the table is used for column headings. The cell specifications use the `colspan="n"` attribute to extend each of the `<td>` heading cells across 2 cells.

```
<style type="text/css">
  table      {border:outset 1px}
  table td    {border:inset 1px; padding:3px}
  table tr#HEAD {font-weight:bold;
                  text-align:center;
                  background-color:#F0F0F0}
</style>

<table>
<tr id="HEAD">
  <td colspan="2">Heading 1</td>
  <td colspan="2">Heading 2</td>
</tr>
<tr>
  <td>Cell 2.1</td>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
  <td>Cell 2.4</td>
</tr>
<tr>
  <td>Cell 3.1</td>
  <td>Cell 3.2</td>
  <td>Cell 3.3</td>
  <td>Cell 3.4</td>
</tr>
</table>
```

Note that there is a need for only two `<td>` heading tags in the first row, although there are four `<td>` cell tags for each data row. Since each `<td>` heading tag spans two cells (`colspan="2"`), a total of four cells, the full width of the table, is accounted for.

The rowspan Attribute

In the same way that you span columns, you can span rows across two or more contiguous cells by using the `rowspan="n"` attribute as shown below.

Heading	Cell 1.2	Cell 1.3	Cell 1.4
	Cell 2.2	Cell 2.3	Cell 2.4

```
<style type="text/css">
  table      {border:outset 1px}
  table td   {border:inset 1px; padding:3px}
  table td#HEAD {font-weight:bold;
                text-align:center;
                background-color:#F0F0F0}
</style>

<table>
<tr>
  <td id="HEAD" rowspan="2">Heading</td>
  <td>Cell 1.2</td>
  <td>Cell 1.3</td>
  <td>Cell 1.4</td>
</tr>
<tr>
  <td>Cell 2.2</td>
  <td>Cell 2.3</td>
  <td>Cell 2.4</td>
</tr>
</table>
```

Note that the first cell in the first row spans two rows. Therefore, there is one less cell to specify in the second row. The first row has *four* `<td>` tags; the second row needs only *three* `<td>` tags since the first cell in that row is encompassed by the previously spanned row.

Spanning Rows and Columns

You can span cells across rows *and* columns to produce interesting and useful table structures. The coding isn't that complex if you just follow closely across each row, determining whether you need to code a `<td>` tag for each cell position or whether you can leave out the `<td>` tag because the cell has already been spanned. In the following code the `<td>` tags that do *not* have to be coded are displayed in brackets []. These cells are accounted for by other cells that have spanned across their table positions.

Cell 1.1		Cell 1.3	Cell 1.4
Cell 2.1	Cell 2.2	Cell 2.3	
Cell 3.1		Cell 3.3	

```
<style type="text/css">
  table {border:outset 1px}
  table td {border:inset 1px; padding:3px}
</style>
```

```
<table>
<tr>
  <td colspan="2">Cell 1.1</td>
  [<td>Cell 1.2</td>]
  <td>Cell 1.3</td>
  <td rowspan="2">Cell 1.4</td>
</tr>
<tr>
  <td>Cell 2.1</td>
  <td rowspan="2">Cell 2.2</td>
  <td>Cell 2.3</td>
  [<td>Cell 2.4</td>]
</tr>
<tr>
  <td>Cell 3.1</td>
  [<td>Cell 3.2</td>]
  <td colspan="2">Cell 3.3</td>
  [<td>Cell 3.4</td>]
</tr>
</table>
```

A Spanning Example

The output below is an example of using row and column spanning to produce a complex table structure.

Quantity and Value of Installed Software				
Software	Department A		Department B	
	Copies	\$ Value	Copies	\$ Value
Word Processors	10	700.00	15	1,050.00
Spreadsheets	12	780.00	18	1,170.00
Databases	7	595.00	9	765.00
Total Value	\$ 2,075.00		\$ 2,985.00	

Coding the Table Structure

The following code begins the table description with a caption followed by the coding for the first row of the table. At this point setting up the style sheet is not a consideration. Focus needs to be on the structure of the table and required column and row spanning.

```
<table>
<caption>Quantity and Value of Installed Software</caption>
<tr>
  <td rowspan="2">Software</td>
  <td colspan="2">Department A</td>
  <td colspan="2">Department B</td>
</tr>
```

Beginning at the top-left of the table, notice that the first cell spans two rows. Thus, this top-left cell is coded with `rowspan="2"` to combine it with the cell beneath to produce a single, combined cell. Also note that this cell contains no content, only the pair of `<td></td>` tags. By leaving the cell empty, cell borders are not displayed.

Moving from left to right across this first row, notice that the next cell, containing Department A, is not a single cell; it spans two horizontal cells. Therefore, the `<td>` tag contains `colspan="2"` to combine this cell with the one in the next column.

Moving to the right you come to the cell containing the heading Department B. Again, this is not a single cell, but a spanned cell that crosses two columns. This `<td>` tag, like the previous one, contains the `colspan="2"` attribute.

You have now reached the end of the first row of the table. It is important to realize that, although you have coded only three cell descriptions, you have accounted for a total of five cells, the total number of columns in the full table. Always keep in mind this fact that you must describe as many rows and as many columns as there are in the full dimensions of the table.

Continuing with the coding for the second row of the table, this row contains the Copies and \$ Value subheadings for their respective columns.



Software	Copies	\$ Value	Copies	\$ Value
----------	--------	----------	--------	----------

```
<tr>
  <td>Copies</td>
  <td>$ Value</td>
  <td>Copies</td>
  <td>$ Value</td>
</tr>
```

Again, begin at the left of the table and move to the right across the row. The first thing to notice is that the first cell in this row does *not* need to be coded. It has already been accounted for by the row spanning that took place for the cell above. So, move to the right to the next cell along the row. The next four cells, in fact, are regular non-spanned cells each of which are described by a <td> tag. You come to the end of the second row having coded four cells, but still accounting for a total of five columns. Coding for the third through fifth row of the table is added below.

Word Processors	10	700.00	15	1,050.00
Spreadsheets	12	780.00	18	1,170.00
Databases	7	595.00	9	765.00

```
<tr>
  <td>Word Processors</td>
  <td>10</td>
  <td>700.00</td>
  <td>15</td>
  <td>1,050.00</td>
</tr>
<tr>
  <td>Word Processors</td>
```

```

<td>12</td>
<td>780.00</td>
<td>18</td>
<td>1,170.00</td>
</tr>
<tr>
<td>Word Processors</td>
<td>7</td>
<td>595.00</td>
<td>9</td>
<td>765.00</td>
</tr>

```

Here there are five separate cells containing a row heading and the main information content of the table. There are no spanned cells so there are five <td> tags across the row. This same coding format can be followed for the next two rows of the table.

The final row of the table is a summary line presenting totals of the dollar amounts in the body of the table. This row is coded below.

Total Value	\$ 2,075.00	\$ 2,985.00
-------------	-------------	-------------

```

<tr>
<td>Total Value</td>
<td colspan="2">$ 2,075.00</td>
<td colspan="2">$ 2,985.00</td>
</tr>
</table>

```

Again, trace across the row beginning with the first cell. This is a standard cell containing the text Total Value. Next you come to a cell that is combined with the following cell and requiring the colspan="2" attribute. Finally, you encounter another cell that is combined with its following cell. You have coded three cell specifications, but again have accounted for a total of five cells across the row.

Viewing the Table Structure

When the above coding is completed the overall structure of the table and the cell contents are finished. This unstyled table is shown in Figure 8-43. Borders have been added to help visualize the cells. The issue at this point is whether you have correctly produced the table structure, ignoring for the moment any styling of the cells.

Quantity and Value of Installed Software				
Software	Department A		Department B	
	Copies	\$ Value	Copies	\$ Value
Word Processors	10	700.00	15	1,050.00
Spreadsheets	12	780.00	18	1,170.00
Databases	7	595.00	9	765.00
Total Value	\$ 2,075.00		\$ 2,985.00	

Styling the Table

Now it is a matter of adding a style sheet to decorate the table. In this example, shown in Listing 8-40, style classes are used to assign header, row, and footer styling. Class .HEAD is applied to the column headings, setting bold and centered text with a background color. Class .ROW formats each data row with right-aligned text, and class .LABEL overrides this row styling for the first cell in the row by aligning text labels to the left. Class .TOTAL formats the footer row with bold and right-aligned text with a background color.

```
<style type="text/css">
  table      {border:outset 1px}
  table td   {border:inset 1px; padding:5px}
  table caption {font:bold 12pt}
  .HEAD      {font-weight:bold; text-align:center; vertical-align:middle;
              background-color:#F0F0F0}
  .ROW       {text-align:right}
  .LABEL     {text-align:left}
  .TOTAL     {text-align:right; font-weight:bold;
              background-color:#F0F0F0}
</style>
```

```
<table>
<caption>Quantity and Value of Installed Software</caption>
<tr class="HEAD">
  <td rowspan="2">Software</td>
  <td colspan="2">Department A</td>
  <td colspan="2">Department B</td>
</tr>
<tr class="HEAD">
  <td>Copies</td>
  <td>$ Value</td>
  <td>Copies</td>
```



```

    <td>$ Value</td>
</tr>
<tr class="ROW">
    <td class="LABEL">Word Processors</td>
    <td>10</td>
    <td>700.00</td>
    <td>15</td>
    <td>1,050.00</td>
</tr>
<tr class="ROW">
    <td class="LABEL">Spreadsheets</td>
    <td>12</td>
    <td>780.00</td>
    <td>18</td>
    <td>1,170.00</td>
</tr>
<tr class="ROW">
    <td class="LABEL">Databases</td>
    <td>7</td>
    <td>595.00</td>
    <td>9</td>
    <td>765.00</td>
</tr>
<tr class="TOTAL">
    <td>Total Value</td>
    <td colspan="2">$ 2,075.00</td>
    <td colspan="2">$ 2,985.00</td>
</tr>
</table>

```

What appeared to be a complex coding exercise turns out to be fairly easily managed by separating structure from styling. First, work systematically across the rows of the table, coding all stand-alone cells and coding cells that require spanning, ignoring cells that are encompassed by a previous span. Then take a look at the raw table, making sure that its structure is correct. Finally, apply a style sheet to format those rows or cells that require styling.

1-3.2 thead, tfoot and tbody tags

The <thead> Tag

The <thead> section contains information that appears one time at the beginning of the table. It is coded very much like a normal set of headings. In this example there are two rows comprising the heading. The first row contains name and address information that is placed in a single cell that spans the width of the table. The second row is a set of column headings.

```
<thead>
<tr>
  <td colspan="5">
    Your Name<br/>
    Your Address<br/>
    City, ST 55555<br/>
  </td>
</tr>
<tr>
  <td>No.</td>
  <td>Description</td>
  <td>Quantity</td>
  <td>Price</td>
  <td>Amount</td>
</tr>
</thead>
```

The <tfoot> Tag

The table footer appears one time at the bottom of a table. It is identified with a <tfoot> tag. The present table contains three rows in the footer. All rows have a spanned label area and a single cell for data.

```
<tfoot>
<tr>
  <td colspan="4">Shipping</td>
  <td>50.00</td>
</tr>
<tr>
  <td colspan="4">Sales Tax</td>
  <td>155.77</td>
</tr>
<tr>
  <td colspan="4">Total</td>
  <td>$ 2,431.16</td>
```

```
</tr>
</tfoot>
```

The <tbody> Tag

The table body displays the main content of the table. It appears in the <tbody> section and contains as many rows as there are data items to report. In the present table, data appear in five separate cells across the row.

```
<tbody>
<tr>
  <td>11111</td>
  <td>Microsoft Windows XP Professional</td>
  <td>2</td>
  <td>169.99</td>
  <td>339.98</td>
</tr>
<tr>
  <td>22222</td>
  <td>Microsoft Office XP Professional</td>
  <td>2</td>
  <td>449.99</td>
  <td>999.98</td>
</tr>
<tr>
  <td>33333</td>
  <td>Adobe Photoshop 7.0</td>
  <td>1</td>
  <td>579.95</td>
  <td>579.95</td>
</tr>
<tr>
  <td>44444</td>
  <td>HP PhotoSmart 7550 Printer</td>
  <td>1</td>
  <td>299.99</td>
  <td>299.99</td>
</tr>
<tr>
  <td>55555</td>
  <td>USB Device Cable</td>
  <td>1</td>
  <td>5.49</td>
```

```
<td>5.49</td>
</tr>
</tbody>
```

SESSION 2-3: FRAMES

2-3.1 Set of frames

A set of frames normally work like the generalized example shown below. Clicking the links in the left frame enables one to view the linked documents in the right frame.

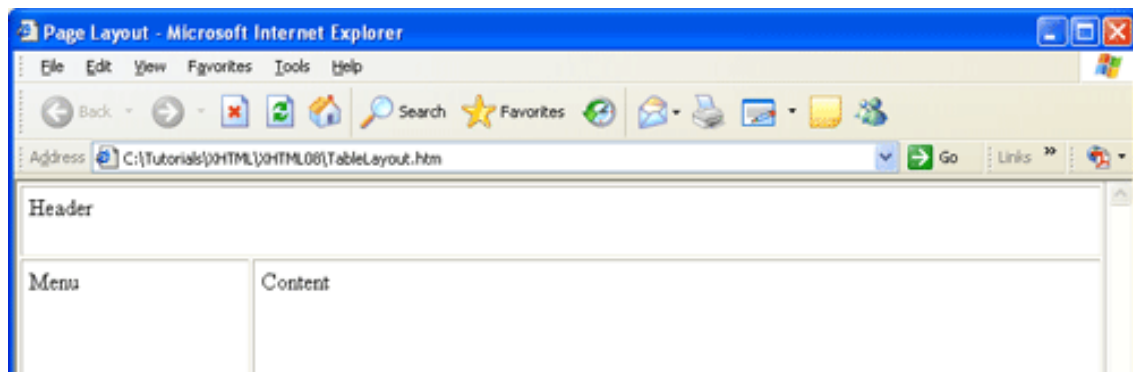


Figure 20: The example browser window is divided into three separate frames

The top frame displays a title; the left frame displays a menu of links; the right frame displays the different pages accessed by these links. All of the frames, in fact, display different Web pages, collected together for display in a single browser window.

An advantage of using frames in this manner is that it is not necessary to code the title and menu on every content page that displays a different "Wonder of the World." They are always viewable and accessible in their frames and do not have to be downloaded with every page. Plus, changing the information content on the screen is only a matter of changing the document appearing in the right frame. By creating these frames you can provide a consistent look to your pages, an ever-present title in the top frame, an ever-present menu in the left frame, and a common format for the content pages in the right frame.

2-3.2 The Frameset Document

A frameset document describes the overall structure of a window subdivided into frames. This is an XHTML document in which `<frameset>` tags *replace* the `<body>` section. A `<frameset>` tag describes the number, locations, and sizes of component frames. Enclosed

within a frameset tag are two or more <frame/> tags which identify and characterize the frames and specify the original documents that populate the frames.

The <frameset> Tag

The <frameset> container tag divides the browser window into separate frames. The general format for the <frameset> tag is shown below.

```
<frameset
  cols="n1[%],n2[%]"
  rows="n1[%],n2[%]"
  frameborder="1|0"
  bordercolor="color"
  framespacing="n"
>
...
</frameset>
```

The cols and rows attributes specify the layout of frames within the browser window. Cols are used to divide the window *vertically* into two or more frames; rows are used to divide the window *horizontally* into two or more frames. A <frameset> tag can specify *either* cols or rows, but not both for the same frameset.

The cols attribute specifies the number and *widths* of frames as percentages of the width of the browser window or as a certain number of pixels in width. The rows attribute specifies the number and *height* of frames as percentages of the height of the browser window or as a certain number of pixels in height. Normally, it is best to use percentages to express frame sizes so they remain proportional when the user resizes the browser window.

The following frameset document establishes a window with two vertical frames, the first of which is 20% and the second of which is 80% of the width of the browser window. The resulting frameset is displayed in Figure 21 as two side-by-side frames divided by a border.

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

<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head>
  <title>Frameset Document</title>
</head>
```

```
<frameset cols="20%,80%">
```

```
...
```

```
</frameset>
```

```
</html>
```

Notice in the above code that the page does *not* include a `<body>` tag. The `<body>` tag is *replaced* by the `<frameset>` tag.

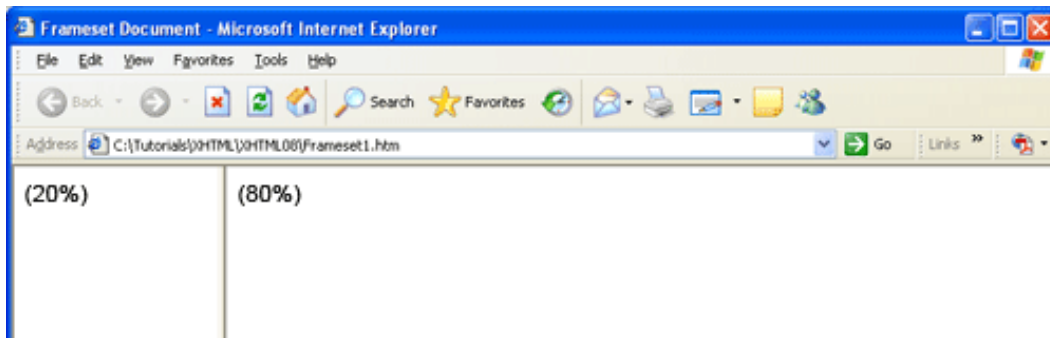


Figure 21: Browser display of a frameset with two column frames

When using the rows attribute the same rules apply except the frameset is created as horizontal rows with frames separated by borders.

```
<frameset rows="20%,80%">
```

```
...
```

```
</frameset>
```

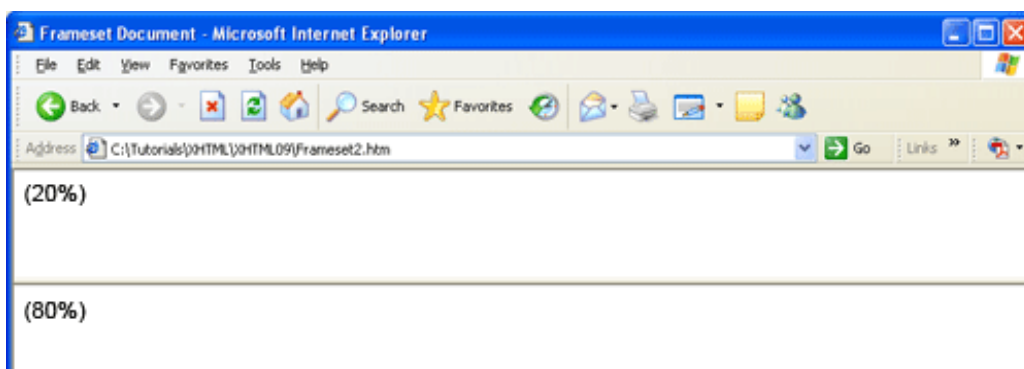


Figure 22: Browser display of a frameset with two row frames

You can use an asterisk (*) rather than a measurement to permit the browser to determine frame sizes. This notation indicates "any remaining space." For example, the previous frameset could have been coded as: `<frameset rows="20%,*">` to permit the browser to calculate the second row as occupying the final 80% of the browser window.

By default, framesets have borders between the frames. You can include the `frameborder="1|0"` attribute in the `<frameset>` tag to show (1) or hide (0) borders.

The width of borders between frames can be set with the `framespacing="n"` attribute, specifying in pixels the width of borders.

If you have visible borders between your frames, you can assign them a color with the `bordercolor="color"` attribute. Colors are specified as color names or hexadecimal or RGB values.

The `<frame>` Tag

While the `<frameset>` tag subdivides the window into frames, those frames are populated with Web pages through `<frame/>` tags. There is one `<frame/>` tag for each of the frames specified in the frameset.

The `<frame/>` tag contains attributes to indicate the original contents of the frame along with the frame's appearance and behavior. The `<frame/>` tag is not a container tag, so it is not paired with a closing tag. Its general format is shown below.

```
<frame  
  src="url"  
  name="framename"  
  frameborder="1|n"  
  bordercolor="color"  
  scrolling="auto|yes|no"  
  noresize
```

Note that the `<frame/>` tag uses the same `frameborder` and `bordercolor` attributes as does the `<frameset>` tag. These attributes are set for the frameset as a whole in the `<frameset>` tag; they can be overridden for particular frames in the `<frame/>` tag. Two frames that border each other, however, cannot have conflicting attributes.

The `src` Attribute

A frame can be initially loaded with a Web page by specifying its URL in the `src="url"` attribute. Coding this attribute ensures that the frame is not blank when the window is first displayed. The document that is preloaded in the frame can be local to the site, or the URL can point to an external document.

The following code shows a document named `Menu.htm` initially loaded into the left frame of a frameset, along with a document named `Home.htm` loaded into the right frame. The documents are assumed to be in the same directory as the frameset page.

```
<frameset cols="20%,80%">
  <frame src="Menu.htm"/>
  <frame src="Home.htm"/>
</frameset>
```

The name Attribute

Even though a frame is preloaded with a document through the src attribute, it can display other pages by targeting the frame in links to those documents. This linking technique is described later. Still, if you intend to load other documents into a frame, it needs to have a name through which it can be referenced in associated hyperlinks. Frames that will only contain the original document specified in the src attribute do not need to be named.

Frame names are assigned with the name attribute. In the following example, the names "Frame1" and "Frame2" are assigned to the two frames of a frameset. Again, you'll see later how to populate these frames with other linked documents.

```
<frameset cols="20%,80%">
  <frame name="Frame1" src="Menu.htm"/>
  <frame name="Frame2" src="Home.htm"/>
</frameset>
```

The noresize Attribute

When frame borders are visible they are also movable. When the mouse cursor is moved on top of a border, its icon changes and the user can drag the border to resize the frame. Normally, you will want your frame setup to remain fixed so that your pages appear as intended. So, to prevent users from changing frame sizes, code the noresize attribute in the <frame/> tag as shown below.

```
<frameset cols="20%,80%">
  <frame name="Frame1" src="Menu.htm" noresize/>
  <frame name="Frame2" src="Title.htm"/>
</frameset>
```

Setting noresize for the left frame has the effect of preventing resizing of the right frame since they share the same border.

The scrolling Attribute

The scrolling="auto|yes|no" attribute creates a frame with or without scroll bars. By default, frames permit scroll bars to appear when the document is too large to fit within the frame. The default value is scrolling="auto". You can change this setting to scrolling="yes" to

always display horizontal and vertical scroll bars even if the document fits within the frame. You can also set `scrolling="no"` to suppress the display of scroll bars even if the document is too large to fit within the frame. As a general rule, you should always permit document scrolling unless you have an overriding reason for not doing so. One case of scroll suppression may be a top frame containing a banner or title without additional content to scroll.

The marginwidth Attribute

As is the case with full-size browser windows, information is displayed across the entire frame, running from the left border to the right border with only a tiny amount of white space separating the text from the borders. You can help improve document readability by introducing margins on either side of the text through use of the `marginwidth="n"` attribute. The attribute value is given as the number of pixels of space to leave between the document and the left and right borders of the frame.

The marginheight Attribute

In the same manner, margins can be introduced at the top and bottom of a document with the `marginheight="n"` attribute. The pixel value indicates the amount of space to leave between the document and the top and bottom of the frame.

An alternative to using `marginwidth` and `marginheight` is to use style sheets to set margins on the documents that occupy the frame.

2-3.3 Nested Frames

The above examples describe a single frameset containing two columns or two rows of frames. You can, by nesting framesets within framesets, define more complex arrangements of frames. A typical example is shown below. The top frame is available for displaying a banner logo identifying the site, the left frame for a menu of links, and the right frame for the documents linked from the menu frame.

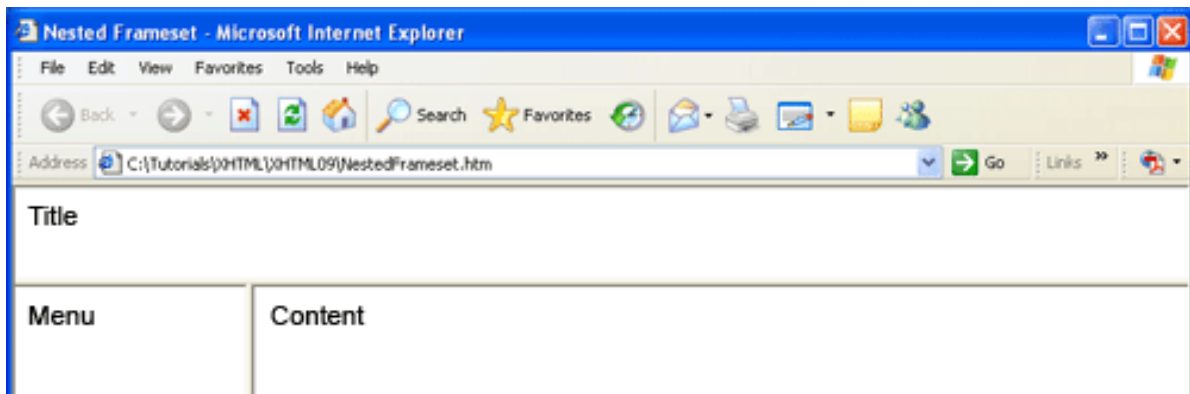


Figure 23: A nested frameset

This arrangement of frames is produced by coding one frameset inside another frameset. The "outer" frameset is composed of two rows. The top row contains the banner frame and the bottom row contains the "inner" frameset of two columnar frames, the left frame displaying a menu and the right frame containing a document. The coding to produce this frameset is shown below.

```
<frameset rows="15%,85%">
  <frame name="Frame1" src="Banner.htm"/>

  <frameset cols="20%,80%">
    <frame name="Frame2" src="Menu.htm"/>
    <frame name="Frame3" src="Document.htm"/>
  </frameset>
</frameset>
```

Of course, you can produce more complex arrangements of frames, but you need to be cautious about doing so. You should create only as many frames as you need to improve the functionality of your site, not to overload visitors with information or complicate navigation between pages.

Targeting Frames

Once you have created a frameset and loaded the frames with initial documents, you need to be able to open other documents inside the frames. Consider the case of a frameset with a menu of links in the left frame and the linked documents appearing in the right frame. By clicking a link in the menu frame you open the appropriate document in the document frame.

The above application is put together with the frameset document shown below. This is the Wonders.htm document initially opened in the browser to reveal the frameset.

```

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

<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head>
  <title>Seven Wonders of the World</title>
</head>

<frameset rows="20%,80%" frameborder="0" framespacing="0">

  <frame src="Title.htm" scrolling="no"/>

  <frameset cols="20%,80%" frameborder="0" framespacing="0">
    <frame src="Menu.htm"/>
    <frame src="Home.htm" name="Content"/>
  </frameset>

</frameset>

</html>

```

This is a nested frameset. The outer frameset is comprised of two rows. The upper row is a frame loaded with a title document (Title.htm); the lower row is a nested frameset. This second frameset contains two columns. The left frame is loaded with the menu document (Menu.htm); the right frame is initially loaded with the opening page of the site (Home.htm). The right frame is also named (name="Content") because it is the target frame for displaying documents linked from the menu frame.

Notice that the framesets do not display borders since frameborder="0" and framespacing="0" are coded in the <frameset> tags. (framespacing needs to be set to 0 to eliminate a tiny amount of spacing between frames even when borders are not displayed.) Also, the top frame is set to scrolling="no" to suppress the default scroll bar.

The target Attribute

Linked documents are loaded into frames by coding the target="*framename*" attribute of the <a> anchor tag. The target attribute specifies the name of the frame (given in the <frame name="*framename*"> tag) in which to open the document. Thus, the Menu.htm page that is

loaded into the left frame above is coded as follows to target the linked pages to the named "Content" frame.

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

<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head>
  <title>Menu Page</title>
  <style type="text/css">
    body {background-color:#F0F0F0}
    h2 {text-align:center}
  </style>
</head>
<body>

<h2>Menu</h2>

<div>
<a href="Artemis.htm" target="Content">Artemis</a><br/>
<a href="Colossus.htm" target="Content">Colossus</a><br/>
<a href="Gardens.htm" target="Content">Gardens</a><br/>
<a href="Halicarnassus.htm" target="Content">Halicarnassus</a><br/>
<a href="Lighthouse.htm" target="Content">Lighthouse</a><br/>
<a href="Pyramid.htm" target="Content">Pyramid</a><br/>
<a href="Zeus.htm" target="Content">Zeus</a><br/>
</div>

</body>
</html>
```

Although the documents appearing in the right frame are changed with each menu selection, the menu document itself always remains in the left frame. It does not have to be reloaded every time a new document page is displayed.

Document Coding

Pages that are loaded into frames utilize standard XHTML coding. No special coding is required to initially load pages with the frameset or when targeted to a frame with a link. The

following two example pages show complete coding for the above Title.htm document and one of the content documents.

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

<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head>
  <title>Title Page</title>
  <style type="text/css">
    body {background-color:#F0F0F0}
    h1 {text-align:center}
  </style>
</head>
<body>

<h1>Seven Wonders of the World</h1>

</body>
</html>

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

<!DOCTYPE html
  PUBLIC "-//W3C//DTD XHTML 1.1//EN"
  "http://www.w3.org/TR/xhtml1/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

<head>
  <title>Artemis.htm Page</title>
  <style type="text/css">
    body {background-color:black; color:white"}
  </style>
</head>
<body>

<div>
<br/>
```

The Temple of Artemis at Ephesus

</div>

</body>
</html>



Self Assessment 2-3

1. Write down the syntax for creating two column frames for a web page.

SESSION 3-3: FORMS

3-3.1 Creating Forms

An XHTML form provides a way for users to interact with a Web page. A form is, basically, a data capture device. It presents the user with one or more data input or selection controls, or fields, through which the user submits information to a Web page. On the basis of this submitted information, the page can respond to the user. This response can vary depending on the purpose of the form. The submitted data may be used, to direct visitors to a different page, much like what happens when clicking a link;

- to present visitors with personalized pages containing information and links pertinent to their interests or preferences;
- to trigger a complex search process to locate information, products, or services about which the user is interested;
- to perform personalized displays of options or calculations of prices of those products or services;
- to trigger a credit card check during purchase of products or services;
- to interface with the organization's accounting and billing systems to finalize purchase; to generate automated email responses as purchase confirmations; and
- the list can go on and on.

The point is that forms are the triggers for a whole host of activities that transform Web sites from simple electronic "page turners" into full-featured information processing systems.

Forms gather information from users by displaying special form controls that permit the user to enter data or make selections. The variety of standard controls that can be coded on a Web form is shown below.

Control Type	Browser Display
Textbox:	<input type="text" value="Entered inform"/>
Password:	<input type="password"/>
Textarea:	<div>Entered information</div>
Radio Button:	<input type="radio"/> Radio Button
Checkbox:	<input checked="" type="checkbox"/> Checkbox
Drop-down List:	<div>Item 1</div>
Submit Button:	<input type="submit" value="Submit Query"/>
Submit Graphic:	
Reset Button:	<input type="reset" value="Reset"/>

A form by itself cannot provide much in the way of processing power. Therefore, XHTML forms are backed by processing routines run remotely at the Web server to handle data submitted from these forms. Form information arriving at the Web server is processed by programs, or scripts, written in server languages such as Visual Basic, PHP, or Java. These are full-featured programming languages that often interact with databases and files on the server, thus offering a full range of information processing capabilities.

Although a large part of form processing takes place on the Web server, certain kinds of processing can take place locally at the browser. These processing scripts are written in the JavaScript language, the default scripting language included with modern browsers. This language has many of the features of regular programming languages. Its processing capabilities, though, are limited to what can be achieved on the local desktop PC.

Textbox Controls

Text input controls require the user to enter information by typing into an input area. Text boxes can be chosen for single-line or multiple-line scrolling input.

The `<input/>` Tag

The most commonly encountered type of form control is the textbox. This control presents a standard text entry box into which information is typed. A text input field is created using an `<input/>` tag in the following format.

```
<input type="text|password"
      id="id"
      size="n"
      maxlength="n"
      value="text"
      Deprecated:
      name="name"
/>
```

The `type` Attribute

The `<input/>` tag includes a `type` attribute to define this as a text entry field. Two different types of text input controls are available. If `type="text"`, or if no `type` attribute is specified, the field type is a standard textbox for entering information. If `type="password"`, a similar control is created but with typed characters echoed as bullet characters to disguise the entered information. The `<input/>` tag is an in-line requiring enclosure inside a block-level tag.

The two control types are shown below along with the coding to produce them. The controls appear in a borderless table in order to align them and their prompt labels. Enter any text into the fields to view their appearance.

Please enter the following information:

Name:

Password:

```
<form action="ThisPage.htm">
<p>Please enter the following information :</p>

<table>
<tr>
  <td>Name: </td>
  <td><input id="TheName" type="text"/></td>
</tr>
```



```

<tr>
  <td>Password: </td>
  <td><input id="ThePassword" type="password"/></td>
</tr>
</table>

</form>

```

The id Attribute

You will nearly always need to identify your text fields, as you will any other form control. Providing an id is important because it provides identification for the data value entered into the field by the user. Both browser and server scripts written to process this information does so through the control id. (The deprecated name attribute can be used in place of an id for scripts requiring this type of identification.)

The size Attribute

Unless you specify a size for a textbox, it is displayed at its default size, which is large enough for approximately 20 typed characters. In most cases you will want to specify a size that is suggestive of the number of characters expected to be entered. For example, the three textboxes below are sized at 15 (City), 2 (State), and 10 (Zip code) characters, respectively. Instead of using the size attribute you can specify a field size using the style sheet width property.

City:

State:

Zip:

```

<form action="ThisPage.htm">
<table>
<tr>
  <td>City: </td>
  <td><input type="text" id="City" size="15"/></td>
</tr>
<tr>
  <td>State: </td>
  <td><input type="text" id="State" size="2"/></td>
</tr>
<tr>
  <td>Zip: </td>

```

```

        <td><input type="text" id="Zip" size="10"/></td>
    </tr>
</table>
</form>

```

The maxlength Attribute

A textbox can hold up to 256 characters, irrespective of its display size. When typing reaches the right boundary of the field, the text scrolls to permit additional characters to be typed. As a general rule, you should not permit the user to enter more than the maximum number of expected characters. When capturing data for server processing, this maximum is often given by the field sizes in the database that stores the values.

You can set a maximum number of allowable characters by coding the maxlength attribute. When this value is coded, the user is not be able to type more than the specified number of characters into the field.

```

<form action="ThisPage.htm">
<table>
<tr>
    <td>City: </td>
    <td><input type="text" id="City" size="15" maxlength="15"/></td>
</tr>
<tr>
    <td>State: </td>
    <td><input type="text" id="State" size="2" maxlength="2"/></td>
</tr>
<tr>
    <td>Zip: </td>
    <td><input type="text" id="Zip" size="10" maxlength="10"/></td>
</tr>
</table>
</form>

```

The value Attribute

A textbox can include the value attribute to pre-enter text in the field. You can provide a default value for the field, or give suggestions about the content expected.

Name:

Name: <input type="text" id="FullName" size="30" value="Enter your full name here"/>

Radio Buttons

Radio buttons are collections of circular buttons that can be clicked "on" and "off." A filled-in, darkened center in the button indicates that the button is chosen; a open center indicates that the button is not chosen. Radio buttons normally provide users with *mutually exclusive* choices. That is, only one of a group of buttons can be chosen. When a different button is clicked, the previous button is turned off. A set of radio buttons is shown below.

The `<input type="radio">` Tag

Radio buttons are defined with one or more `<input type="radio"/>` tags. A separate tag is required for each button in a set. The general format for this tag is shown below.

```
<input type="radio"
  id="id"
  name="name"
  value="text"
  checked="checked"
/>
```

The `<input type="radio"/>` tag only displays a button. In order to provide a label for the button, text can appear either before or after the button code to place it to the left or right of the button.

An id can be assigned to a button if there is a need to identify it for individual script processing, normally by a browser script. Otherwise, an id is not needed.

Radio buttons are usually grouped to present a set of related choices for the user. In this case, only one of the options can be chosen -- they are mutually exclusive choices. When a button is clicked for selection, and that button is highlighted, any previously chosen button becomes de-selected and un-highlighted. A group of radio buttons is made to act in this way by assigning the *same* name to all buttons in the group, as shown in the following code for the buttons displayed above.

```
<input type="radio" name="Choice"/>Choice 1<br/>
<input type="radio" name="Choice"/>Choice 2<br/>
<input type="radio" name="Choice"/>Choice 3<br/>
```

There are three buttons in this group, each appearing on a separate line. A text label appears to the right of the buttons to identify each choice. All buttons have the same name to enforce only one choice from the group.

In a manner similar to text boxes, password boxes, and text areas, radio buttons have data values associated with the choices. These values are provided explicitly through value

attributes coded for the buttons. When a button is chosen, its particular value is associated with the name of the button that is, with the name assigned to all of the buttons in the group. Consider the following example.

What is your favorite color?

- ☐ Red
- ☐ Green
- ☐ Blue

What is your favorite color?


```
<input type="radio" name="Color" value="Red"/>Red<br/>
<input type="radio" name="Color" value="Green"/>Green<br/>
<input type="radio" name="Color" value="Blue"/>Blue<br/>
```

All of the buttons share the name "Color" in order to set up a group of mutually exclusive choices. The particular color chosen from the group is given by the value associated with the choice. So, if the user clicks the first button, the color "Red" is chosen. More accurately, the value "Red" becomes associated with the name "Color" as indicative of the choice.

It is normally always necessary to code value attributes for radio buttons. This is the way in which buttons take on values and how a browser script or server program determines which button is clicked. Also, it is not necessary to assign a value that matches the button label.

The checked Attribute

When a set of radio buttons is first displayed, all of the buttons appear unchecked; that is, none are selected and highlighted. You can, however, force one of the buttons in the group to appear pre-checked, or pre-selected. You do this by coding the checked="checked" attribute in the <input type="radio"/> tag of the button you want to appear "on" when the page is loaded.

- ☒ Red
- ☐ Green
- ☐ Blue

What is your favorite color?


```
<input type="radio" name="Color" value="R" checked="checked"/>Red<br/>
<input type="radio" name="Color" value="G"/>Green<br/>
<input type="radio" name="Color" value="B"/>Blue<br/>
```

Checkboxes

A checkbox, like a radio button, provides the user with choices. In the case of a series of checkboxes, however, the choices are not mutually exclusive; rather, the user can select *one or more* of the choices. For those boxes that are checked, a checkmark appears in the box. Unchosen boxes remain empty. A set of checkboxes is illustrated below.

.

The `<input type="checkbox"/>` Tag

The general format for coding a checkbox is similar to that for a radio button. An `<input type="checkbox"/>` tag is required for each checkbox.

```
<input type="checkbox"
  id="id"
  name="name"
  value="text"
  checked="checked"
/>
```

The `<input type="checkbox"/>` tag only displays a checkbox. In order to provide a label for the checkbox, text can appear either before or after the tag to place it to the left or right of the checkbox.

An id can be assigned to a checkbox if there is a need to identify it for script processing, normally by a browser script. Otherwise, an id is not needed.

The name Attribute

Checkboxes may need to be named with name attributes so that chosen values can be associated with the names for server script processing. As is the case with radio buttons, all checkboxes in a group can have the same name. Unlike radio buttons, however, a common name does *not* force the choices to be mutually exclusive. The name simply takes on more than one value if more than one choice is made. This presents some processing peculiarities, but it is valid to do this. A better solution is to assign different names to the checkboxes. Each name then becomes associated with a different value irrespective of the number of checks made by the user. If a set of checkboxes is involved only in browser processing, then id values, rather than names, can be coded for the checkboxes.

The value Attribute

Similar to a radio button, the value associated with a checkbox is given in the checkbox's value attribute. When a form is submitted for server processing, this value is associated with the name of the particular checkbox. For browser processing, the value is associated with the checkbox's id. Values do not have to match the labels. Normally, abbreviated codes are used for values, and more descriptive text is used for labels.

The checked Attribute

Checkboxes can be pre-checked by coding checked="checked" for as many different checkboxes as you want pre-checked.

The various settings for checkboxes are shown below. In this example, all checkboxes are assigned different names to differentiate them for server processing. No checkboxes are pre-checked. No id values are assigned since these checkboxes are processed by a server script rather than a browser script.

What are your favorite colors?


```
<input type="checkbox" name="Color1" value="R"/>Red<br/>
<input type="checkbox" name="Color2" value="G"/>Green<br/>
<input type="checkbox" name="Color3" value="B"/>Blue<br/>
<input type="checkbox" name="Color4" value="Y"/>Yellow<br/>
<input type="checkbox" name="Color5" value="M"/>Maroon<br/>
<input type="checkbox" name="Color6" value="P"/>Purple<br/>
<input type="checkbox" name="Color7" value="A"/>Aqua<<br/>
<input type="checkbox" name="Color8" value="T"/>Teal<br/>
```

A drop-down list, or selection list, presents a list of items from which one or more are chosen. By clicking the down-arrow at the right of the list, the full list is exposed. An item is chosen by clicking an entry.

The <select> and <option> Tags

The menu of choices is created with the <select> tag. Inside this tag are <option> tags, one for each item in the list. The general formats for the <select> and <option> tags are shown below.

```
<select
  id="id"
  name="name"
  size="n"
  multiple="multiple"
>
  <option
    value="text"
    selected="selected"
  >
    Label
  </option>
  ...
</select>
```

The id Attribute

An id can be assigned to a drop-down list if there is a need to identify it for browser script processing. Otherwise, an id is not needed.

The name Attribute

The drop-down list can be assigned a name with the name attribute coded in the <select> tag. The list name becomes associated with the value of the item selected for certain types of server processing.

The <option> Tag

Items are defined for appearance in a drop-down list with the <option> tag. There are as many tags as there are items in the list. The label for the option is entered between the opening and closing tags. This is the text string that is visible in the list and becomes the default value for the selection.

The value Attribute

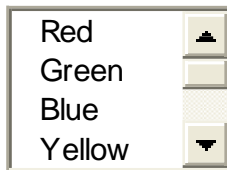
By default, the value associated with a drop-down option is given by the text label included in the <option> tag. However, a value attribute can be coded to supply a different value from the label. You might choose to do this when the labels are extended text strings but you need only to collect abbreviated codes for the values.

```
Choose your favorite color:<br/>
<select id="Color">
  <option value="R">Red</option>
  <option value="G">Green</option>
  <option value="B">Blue</option>
  <option value="Y">Yellow</option>
  <option value="M">Maroon</option>
  <option value="P">Purple</option>
  <option value="A">Aqua</option>
  <option value="T">Teal</option>
</select>
```

The size Attribute

The size attribute of the <select> tag indicates the number of items that are visible in the list. By default, the list displays only one item. If a size is specified, the list displays that number of items and, if necessary, a scroll bar to access them. The list shown below uses the attribute size="4" to display four items at a time.

Choose your favorite color:



The multiple Attribute

Normally, only a single item can be chosen from the list. However, one or more items can be chosen by coding the `multiple="multiple"` attribute of the `<select>` tag. Multiple items are chosen by using the Shift-Click or Ctrl-Click method of selection.

The `<select>` tag for the following list has `size="4"` and `multiple="multiple"` attributes to permit multiple choices. If multiple items are chosen, the name or id of this control is associated with that collection of values, and any scripts that process the control need to determine the multiple values selected.

Choose your favorite color:



The selected Attribute

Items in the list can be pre-selected by coding the `selected="selected"` attribute in the associated `<option>` tag. Multiple pre-selections can be made only when `multiple="multiple"` is coded for the `<select>` tag. The drop-down list has two items (Red and Blue) pre-selected.

Submit and Reset Buttons

All forms that are submitted for server processing must include at least one "submit" button to transmit form information to the Web page identified in the action attribute of the `<form>` tag. Either a standard button or a graphic image can be used for form submission.

The `<input type="submit"/>` Button

A form submission button can be created with an `<input type="submit"/>` tag and can appear anywhere on the form. The default appearance of the button with its "Submit Query" label is shown below.

A rectangular button with a light beige background and a thin black border. The text "Submit Query" is centered on the button in a black, sans-serif font.

The general format for the `<input type="submit"/>` tag to define a form submission button is given as

```
<input type="submit"
  id="id"
  name="name"
  value="text"
/>
```

The id Attribute

An id can be assigned to a submit button if there is a need to identify it for browser script processing. Otherwise, an id is not needed.

The name Attribute

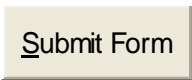
The submit button can be assigned a name with the name attribute. A name is required for certain types of server processing.

The value Attribute

The value attribute provides two kinds of identification for the button. On the one hand, this value is associated with the button name and indicates to a server script which button was clicked; on the other, this value is used as the label for the button. If a value attribute is not coded, the button is labeled "Submit Query"; however, you can assign any text string to provide helpful identification of the button as a submit button.

Typical coding and display for a submit button is shown below.

```
<input type="submit" name="SubmitButton" value="Submit Form"/>
```

A rectangular button with a light beige background and a thin black border. The text "Submit Form" is centered on the button in a black, sans-serif font.

The `<input type="reset"/>` Tag

One other button type is available for forms. A "reset" button can be created to permit users to clear all information from a form, either to permit entry of new information or to clear incorrect information. Its default appearance is shown below.

A rectangular button with a light beige background and a thin black border. The text "Reset" is centered on the button in a black, sans-serif font.

This button is created by coding an `<input type="reset"/>` tag. You can name or id the button and can replace the default "Reset" label by coding its value attribute. The action of the button is to automatically reset the form, clearing all text input areas and resetting radio buttons, checkboxes, and drop-down lists back to their default appearances.

The following example shows use of a reset button. Enter text into the textbox; then click the reset button. Information in the textbox is cleared.

Enter text:

Other Buttons

The two primary types of buttons appearing on forms are submit and reset buttons. These two buttons are associated with form submission to programs that process form information. Other general-purpose buttons can be defined. These are normally associated with browser scripts for local processing, of forms and otherwise, by JavaScript routines embedded on the Web page.

The `<input type="button"/>` Tag

An `<input type="button"/>` tag is used to create general-purpose buttons to call browser scripts located on the same Web page. Its general format is shown below.

```
<input type="button"
  id="id"
  name="name"
  value="text"
  [onclick="script"]
/>
```

An id or name may be necessary if the button is referenced in a script; otherwise these attributes need not be coded. A value attribute is required to supply a label for the button; otherwise, an unlabeled button appears.

This type of button usually has the purpose of responding to a mouse click and activating an embedded script or calling a JavaScript function to perform some processing. Therefore, an event handler such as onclick usually appears in the button to enclose a script or call a function.

The `<button>` Tag

The `<button>` tag provides versatility in producing buttons. It can be set up as a submit button, a reset button, or a general purpose button. As a container tag, it can enclose any text

```
<button
  type="submit|reset|button"
  id="id"
  name="name"
  [onclick="script"]
>
  ...text and XHTML tags
</button>
```

The label for the button appears between the opening and closing tags. This can be a simple text label, or XHTML tags can be coded to decorate the text or to include a graphic image on the button.

A Form Example

Membership Application Form

114

The form elements have been coded inside a table structure to help align the labels and fields. You should recognize all of the form controls in the following code listing.

```
<form name="MyForm" action="MembershipForm.htm" method="post">
```

```
<h2>Membership Application Form</h2>
```

```
<table>
<tr>
  <td>First Name: </td>
  <td><input type="text" name="FirstName" size="15" maxlength="15"/></td>
</tr>
<tr>
  <td>Last Name: </td>
  <td><input type="text" name="LastName" size="15" maxlength="15"/></td>
</tr>
<tr>
  <td>Email: </td>
  <td><input type="text" name="Email" size="30" maxlength="50"/></td>
</tr>
<tr>
  <td>Gender: </td>
  <td><input type="radio" name="Gender" value="F"/>Female
    <input type="radio" name="Gender" value="M"/>Male</td>
</tr>
<tr>
  <td>Major: </td>
  <td><select name="Major">
    <option>Web Development</option>
    <option>Digital Media</option>
    <option>Database Administration</option>
    <option>Networking</option>
    <option>Software Development</option>
    <option>Systems Analysis</option>
  </select></td>
</tr>
<tr>
  <td>Reason for<br/>Joining: </td>
  <td><textarea name="Reason" cols="30" rows="5"></textarea></td>
</tr>
<tr>
  <td></td>
  <td><input type="submit" name="SubmitButton" value="Submit Form"/></td>
</tr>
```

</table>

</form>

The mailto: Action

Normally, information from a form is transmitted to the server for processing by a script, as is the case with the above membership form. You can, however, transmit form information through an email message without using a script. This method uses mailto: in the action URL of the <form> tag, supplying an email address for receipt of the information. The syntax of the mailto: action is <form action="mailto:email@address">

When you click the submit button on a form that uses the mailto: action, your computer uses your email program to receive the content from the Web form. Depending on how the email system is configured, either you have a chance to edit the mail message, or it is sent automatically to the email address in the mailto: attribute.



Self Assessment

1. Which of the following is not valid HTML?
 - a. <frameset cols="16%,*">
 - b. <frameset rows="16%,84%">
 - c. <frameset rows="100,100">
 - d. <frameset cols="100,150">
 - e. <frameset tables="100,150">



Unit Summary

In this unit we have covered the use of frames, tables and forms in web design.



Unit Assignment 3

Study the KNUST web site and develop a web site for your former school including tables, forms and form controls etc.

CLIENT-SIDE, SERVER-SIDE SCRIPTING AND ASP.NET

Introduction

Scripting languages are programming languages with many of the same capabilities as compiled known programming languages, but they are written as source code and then, instead of being compiled at that point, they are interpreted (which is like real-time compiling) when they are used.

JavaScript is used in millions of Web pages to improve the design, validate forms, detect browsers, create cookies, and much more.

ASP.NET is Microsoft's application programming approach for the development of web site application software. This is a development approach that competes with JAVA programming.



Learning Objectives

After reading this unit you should be able to:

1. Use scripting languages for validation on the web pages.
2. Use ASP.Net to develop a web application.

Unit content

Session 1-4: Scripting languages

- 1-4.1 PHP and VB script
- 1-4.2 Client-sides scripting with JavaScript
- 1-4.3 Server-side scripting - CGI
- 1-4.4 Servlets and java server papers]

Session 2-4: Developing web applications using ASP.Net

- 2-4.1 ASP.Net
- 2-4.2 Differences between static and dynamic web pages
- 2-4.3 Start web application with ASP.Net
- 2-4.4 Using ASP. Net Validation controls
- 2-4.5 Multi-page web application

SESSION 1-4: SCRIPTING LANGUAGES

1-4.1 PHP and VB Script

Interpreted languages are programming languages with many of the same capabilities as compiled known programming languages, but they are written as source code and then, instead of being compiled at that point, they are interpreted (which is like real-time compiling) when they are used.

There are some differences of how interpreted languages are interpreted. For example, PHP and VBScript are both interpreted scripting languages (PHP is an open-source project, while VBScript is a lower-level version of Microsoft's Visual Basic programming language), meaning they are sent to an interpreter (called a scripting engine) when a web page is requested, so that before the HTML is gathered and sent back to the user, the PHP or VBScript mixed with the page is processed. All the user sees is the HTML code; the PHP or VBScript is processed and removed (except for the results).

Here is an example of PHP code:

```
<? if (isset($posted) {  
    echo "Your name is $first_name";  
} else {  
    echo "Don't know your name";  
}  
?>
```

Here is an example of similar functionality using VBScript

```
<% if (posted="True" Then  
    Response.Write "Your name is "& FirstName;  
else  
    Response.Write "Don't know your name";  
End If  
%>
```

JavaScript on the other hand, is sent to the user along with HTML making up a Web page and is interpreted by the user's browser. JavaScript can perform many programming language functions but is restricted from accessing the hard drive and reading or writing files, to protect users from malicious Web page designers.

1-4.2 Client side scripting with JavaScript

JavaScript is used in millions of Web pages to improve the design, validate forms, detect browsers, create cookies, and much more. It is the most popular scripting language on the internet, and works in all major browsers, such as Internet Explorer, Mozilla, Firefox, Netscape, and Opera.

What is JavaScript?

- JavaScript was designed to add interactivity to HTML pages.
- JavaScript is a scripting language. A scripting language is a lightweight programming language
- A JavaScript consists of lines of executable computer code
- A JavaScript is usually embedded directly into HTML pages
- JavaScript is an interpreted language (means that scripts execute without preliminary compilation)
- Everyone can use JavaScript without purchasing a license

Java and JavaScript are two completely different languages in both concept and design!

Java (developed by Sun Microsystems) is a powerful and much more complex programming language - in the same category as C and C++.

What can a JavaScript Do?

JavaScript gives HTML designers a programming tool – HTML authors are normally not programmers, but JavaScript is a scripting language with a very simple syntax!

- JavaScript can put dynamic text into an HTML page – A JavaScript statement like this: `document.write("<h1>" + name + "</h1>")` can write a variable text into an HTML page
- JavaScript can react to events – A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- JavaScript can read and write HTML elements – A JavaScript can read and change the content of an HTML element
- JavaScript can be used to validate data – A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- JavaScript can be used to detect the visitor's browser – A JavaScript can be used to detect the visitor's browser, and – depending on the browser – load another page specifically designed for that browser
- JavaScript can be used to create cookies – A JavaScript can be used to store and retrieve information on the visitor's computer

JavaScript's official name is "ECMAScript". The standard is developed and maintained by the ECMA organisation. The language was invented by Brendan Eich at Netscape (with Navigator 2.0), and has appeared in all Netscape and Microsoft browsers since 1996.

The HTML `<script>` tag is used to insert a JavaScript into an HTML page.

Write text with Javascript

The example demonstrates how to use JavaScript to write text

```
<html>
<body>
<script type="text/javascript">
document.write("Welcome to web concepts and development!");
</script>
</body>
</html>
```

The code above will produce this output on an HTML page:

Welcome to web concepts and development!

To insert a JavaScript into an HTML page, we use the `<script>` tag. Inside the `<script>` tag we use the "type=" attribute to define the scripting language.

So, the `<script type="text/javascript">` and `</script>` tells where the JavaScript starts and ends:

The word `document.write` is a standard JavaScript command for writing output to a page.

By entering the `document.write` command between the `<script>` and `</script>` tags, the browser will recognize it as a JavaScript command and execute the code line. In this case the browser will write "Welcome to web concepts and development!" to the page:



Note

If we had not entered the `<script>` tag, the browser would have treated the `document.write` ("Welcome to web concepts and development!") command as pure text, and just write the entire line on the page.

HTML Comments to Handle Simple Browsers

Browsers that do not support JavaScript will display JavaScript as page content. To prevent them from doing this, and as a part of the JavaScript standard, the HTML comment tag can be used to "hide" the JavaScript. Just add an HTML comment tag `<!--` before the first JavaScript statement, and a `-->` (end of comment) after the last JavaScript statement.

```
<html>
<body>
<script type="text/javascript">
<!--
document.write("Welcome to web concepts and development!");
//-->
</script>
</body>
</html>
```

The two forward slashes at the end of comment line (`//`) is the JavaScript comment symbol. This prevents JavaScript from executing the `-->` tag.

Where to Put the JavaScript

JavaScripts in the body section will be executed WHILE the page loads.

JavaScripts in the head section will be executed when CALLED.

Head section

Scripts that contain functions go in the head section of the document. Then we can be sure that the script is loaded before the function is called.

```
<html>
<head>
<script type="text/javascript">
function message()
{
alert("This alert box was called with the onload event");
}
</script>
</head>
<body onload="message()">
</body>
</html>
```

Body section

Execute a script that is placed in the body section.

```
<html>
<head>
</head>
<body>
<script type="text/javascript">
document.write("This message is written by JavaScript");
</script>
</body>
</html>
```

You can place an unlimited number of scripts in your document, so you can have scripts in both the body and the head

Using an External JavaScript

Sometimes you might want to run the same JavaScript on several pages, without having to write the same script on every page. To simplify this, you can write a JavaScript in an external file. Save the external JavaScript file with a .js file extension.



Note

The external script cannot contain the `<script>` tag!

To use the external script, point to the .js file in the "src" attribute of the `<script>` tag: `<html>`

```
<head>
<script src="xxx.js"></script>
</head>
<body>
</body>
</html>
```

JavaScript Statements

JavaScript is a sequence of statements to be executed by the browser. JavaScript is Case Sensitive. Unlike HTML, JavaScript is case sensitive – therefore watch your capitalization closely when you write JavaScript statements, create or call variables, objects and functions.

This JavaScript statement tells the browser to write "Hello KNUST" to the web page:

```
document.write("Hello KNUST");
```

It is normal to add a semicolon at the end of each executable statement. Most people think this is a good programming practice, and most often you will see this in JavaScript examples on the web.

The semicolon is optional (according to the JavaScript standard), and the browser is supposed to interpret the end of the line as the end of the statement. Because of this you will often see examples without the semicolon at the end.

This example will write a header and two paragraphs to a web page:

```
< script type="text/javascript">
document.write("<h1>This is a header</h1>");
document.write("<p>This is a paragraph</p>");
document.write("<p>This is another paragraph</p>");
</script>
```

JavaScript Blocks

JavaScript statements can be grouped together in blocks. Blocks start with a left curly bracket {, and ends with a right curly bracket, }. The purpose of a block is to make the sequence of statements execute together.

This example will write a header and two paragraphs to a web page:

```
<script type="text/javascript">
{
document.write("<h1>This is a header</h1>");
document.write("<p>This is a paragraph</p>");
document.write("<p>This is another paragraph</p>");
}
</script>
```

JavaScript Comments

JavaScript comments can be used to make the code more readable. Comments can be added to explain the JavaScript, or to make it more readable.

Single line comments start with //.

Multi line comments start with /* and end with */.

Comments may be used to Prevent Execution

JavaScript Variables

Variables are "containers" for storing information:

```
x=5; carname="Volvo";
```



NOTE

Because JavaScript is case-sensitive, variable names are case-sensitive.

Example

A variable's value can change during the execution of a script. You can refer to a variable by its name to display or change its value.

JavaScript Operators

To add a space between the two strings, insert a space into one of the strings:

```
txt1="What a very ";
```

```
txt2="nice day";
```

```
txt3=txt1+txt2;
```

```
x=5+"5";
```

```
document.write(x);
```

```
x="5"+5;
```

```
document.write(x);
```

JavaScript Comparison and Logical Operators

Comparison and Logical operators are used to test for true or false. Comparison operators are used in logical statements to determine equality or difference between variables or values.

Given that x = 5, the table below explains the comparison operators:

Operator	Description	Example
==	is equal to	x==8 is false
===	is exactly equal to (value and type)	x===5 is true x=== "5" is false
!=	is not equal	x!=8 is true
>	is greater than	x>8 is false
<	is less than	x<8 is true
>=	is greater than or equal to	x>=8 is false
<=	is less than or equal to	x<=8 is true

Comparison operators can be used in conditional statements to compare values and take action depending on the result:

```
if (age<18)
document.write("Too young");
```

Logical Operators

Logical operators are used to determine the logic between variables or values.

Given that x = 6 and y = 3, the table below explains the logical operators:

Operator	Description	Example
&&	and	(x < 10 && y > 1) is true
	or	(x==5 y==5) is false
!	not	!(x==y) is true

Conditional Operator

JavaScript also contains a conditional operator that assigns a value to a variable based on some condition.

Syntax

```
variablename=(condition)?value1:value2
```

Example

```
Greeting = (visitor=="PRES")?"Dear President ":"Dear ";
```

If the variable visitor has the value of "PRES", then the variable greeting will be assigned the value "Dear President" else it will be assigned "Dear".

Let us type and run the following examples



ACTIVITY Example 1

```
<html>
<body>
<script type="text/javascript">
var d = new Date();
var time = d.getHours();
if (time < 12)
{
document.write("<b>Good morning</b>");
```

```

}
</script>
<p>
This example demonstrates the If statement.
</p>
<p>
If the time on your browser is less than 12,
you will get a "Good morning" greeting.
</p>
</body>
</html>

```

Example 2

```

<html>
<body>

<script type="text/javascript">
var d = new Date();
var time = d.getHours();
if (time<10)
{
document.write("<b>Good morning</b>");
}
else if (time>=10 && time<16)
{
document.write("<b>Good day</b>");
}
else
{
document.write("<b>Welcome to web concepts and development!</b>");
}
</script>

<p>
This example demonstrates the if..else if...else statement.
</p>

</body>
</html>

<html>
<body>
<script type="text/javascript">

```

```

var r=Math.random();
if (r>0.5)
{
document.write("<a href='http://www.w3schools.com'>Learn Web Development!</a>");
}
else
{
document.write("<a href='http://www.refsnesdata.no'>Visit Refs Data!</a>");
}
</script>
</body>
</html>

```

Example 3

```

<html>
<body>
<script type="text/javascript">
var d = new Date();
theDay=d.getDay();

switch (theDay)
{
case 5:
document.write("<b>Finally Friday</b>");
break;
case 6:
document.write("<b>Super Saturday</b>");
break;
case 0:
document.write("<b>Sleepy Sunday</b>");
break;
default:
document.write("<b>I'm really looking forward to this weekend!</b>");
}
</script>

```

<p>This JavaScript will generate a different greeting based on what day it is. Note that Sunday=0, Monday=1, Tuesday=2, etc.</p>

```

</body>
</html>

```


Popup Boxes

In JavaScript we can create three kinds of popup boxes:

- Alert box,
- Confirm box, and
- Prompt box.

An alert box is often used if you want to make sure information comes through to the user. When an alert box pops up, the user will have to click "OK" to proceed.

Syntax:

```
alert("sometext");
```

A confirm box is often used if you want the user to verify or accept something. When a confirm box pops up, the user will have to click either "OK" or "Cancel" to proceed.

If the user clicks "OK", the box returns true. If the user clicks "Cancel", the box returns false.

Syntax:

```
confirm("sometext");
```

A prompt box is often used if you want the user to input a value before entering a page. When a prompt box pops up, the user will have to click either "OK" or "Cancel" to proceed after entering an input value.

If the user clicks "OK" the box returns the input value. If the user clicks "Cancel" the box returns null.

Syntax:

```
prompt("sometext","defaultvalue");
```

Examples

```
<html>
<head>
<script type="text/javascript">
function disp_alert()
{
alert("I am an alert box!!");
}
</script>
</head>
<body>
```

```
<input type="button" onclick="disp_alert()" value="Display alert box" />
```

```
</body>
</html>
```

```
<html>
<head>
<script type="text/javascript">
function disp_alert()
{
alert("Hello again! This is how we" + '\n' + "add line breaks to an alert box!");
}
</script>
</head>
<body>
```

```
<input type="button" onclick="disp_alert()" value="Display alert box" />
```

```
</body>
</html>
```

```
<html>
<head>
<script type="text/javascript">
function disp_confirm()
{
var r=confirm("Press a button");
if (r==true)
{
document.write("You pressed OK!");
}
else
{
document.write("You pressed Cancel!");
}
}
</script>
</head>
<body>
```

```
<input type="button" onclick="disp_confirm()" value="Display a confirm box" />
```

```
</body>
</html>
```

```

<html>
<head>
<script type="text/javascript">
function disp_prompt()
{
var name=prompt("Please enter your name","Harry Potter");
if (name!=null && name!="")
{
document.write("Hello " + name + "! How are you today?");
}
}
</script>
</head>
<body>

<input type="button" onclick="disp_prompt()" value="Display a prompt box" />

</body>
</html>

```

JavaScript Functions

To keep the browser from executing a script when the page loads, you can put your script into a function. A function contains code that will be executed by an event or by a call to that function.

You may call a function from anywhere within the page (or even from other pages if the function is embedded in an external .js file).

Functions can be defined both in the <head> and in the <body> section of a document. However, to assure that the function is read/loaded by the browser before it is called, it could be wise to put it in the <head> section.

Example

```

<html>
<head>
<script type="text/javascript">
function displaymessage()
{
alert("Welcome to web concepts and development!");
}
</script>
</head>

```

```
<body>
<form>
<input type="button" value="Click me!"
onclick="displaymessage()" >
</form>
</body>
</html>
```

The syntax for creating a function is: `function functionname(var1,var2,...,varX)`

```
{
some code
}
```

var1, var2, etc are variables or values passed into the function. The { and the } defines the start and end of the function.



Note

A function with no parameters must include the parentheses () after the function name:

```
function functionname()
{
some code
}
```

The function below should return the product of two numbers (a and b):

```
function prod(a,b)
{
x=a*b;
return x;
}
```

When you call the function above, you must pass along two parameters:

```
product=prod(2,3);
```

The returned value from the prod() function is 6, and it will be stored in the variable called product.

JavaScript Events

Events are actions that can be detected by JavaScript.

By using JavaScript, we have the ability to create dynamic web pages. Events are actions that can be detected by JavaScript.

Every element on a web page has certain events which can trigger JavaScript functions. For example, we can use the `onClick` event of a button element to indicate that a function will run when a user clicks on the button. We define the events in the HTML tags.

Examples of events:

- A mouse click
- A web page or an image loading
- Mousing over a hot spot on the web page
- Selecting an input box in an HTML form
- Submitting an HTML form
- A keystroke



Note

Events are normally used in combination with functions, and the function will not be executed before the event occurs!

The `onload` and `onUnload`

The `onload` and `onUnload` events are triggered when the user enters or leaves the page.

The `onload` event is often used to check the visitor's browser type and browser version, and load the proper version of the web page based on the information.

Both the `onload` and `onUnload` events are also often used to deal with cookies that should be set when a user enters or leaves a page. For example, you could have a popup asking for the user's name upon his first arrival to your page. The name is then stored in a cookie. Next time the visitor arrives at your page, you could have another popup saying something like: "Welcome John Doe!".

`onFocus`, `onBlur` and `onChange`

The `onFocus`, `onBlur` and `onChange` events are often used in combination with validation of form fields.

Below is an example of how to use the `onChange` event. The `checkEmail()` function will be called whenever the user changes the content of the field:

```
<input type="text" size="30"  
id="email" onchange="checkEmail()">
```

onSubmit

The onSubmit event is used to validate ALL form fields before submitting it.

Below is an example of how to use the onSubmit event. The checkForm() function will be called when the user clicks the submit button in the form. If the field values are not accepted, the submit should be cancelled. The function checkForm() returns either true or false. If it returns true the form will be submitted, otherwise the submit will be cancelled:

```
<form method="post" action="xxx.htm"
onsubmit="return checkForm()">
```

onMouseOver and onMouseOut

onMouseOver and onMouseOut are often used to create "animated" buttons.

Below is an example of an onMouseOver event. An alert box appears when an onMouseOver event is detected:

```
<a href="http://www.w3schools.com"
onmouseover="alert('An onMouseOver event');return false">

</a>
```

Examples

The example below contains a script that is supposed to display the message "Welcome guest!" when you click on a button. However, there's a typo in the message() function. alert() is misspelled as adddalert(). A JavaScript error occurs:

```
<html>
<head>
<script type="text/javascript">
function message()
{
adddalert("Welcome guest!");
}
</script>
</head>

<body>
<input type="button" value="View message" onclick="message()" />
</body>
```

</html>

To take more appropriate action when an error occurs, you can add a try...catch statement.

The example below contains the "Welcome guest!" example rewritten to use the try...catch statement. Since alert() is misspelled, a JavaScript error occurs. However, this time, the catch block catches the error and executes a custom code to handle it. The code displays a custom error message informing the user what happened:

```
<html>
<head>
<script type="text/javascript">
var txt=""
function message()
{
try
{
  adddler("Welcome guest!");
}
catch(err)
{
  txt="There was an error on this page.\n\n";
  txt+="Error description: " + err.description + "\n\n";
  txt+="Click OK to continue.\n\n";
  alert(txt);
}
}
</script>
</head>

<body>
<input type="button" value="View message" onclick="message()" />
</body>

</html>
```

The next example uses a confirm box to display a custom message telling users they can click OK to continue viewing the page or click Cancel to go to the homepage. If the confirm method returns false, the user clicked Cancel, and the code redirects the user. If the confirm method returns true, the code does nothing:

```
<html>
<head>
<script type="text/javascript">
var txt=""
```

```

function message()
{
try
{
  adddlert("Welcome guest!");
}
catch(err)
{
  txt="There was an error on this page.\n\n";
  txt+="Click OK to continue viewing this page,\n";
  txt+="or Cancel to return to the home page.\n\n";
  if(!confirm(txt))
  {
    document.location.href="http://www.w3schools.com/";
  }
}
}
</script>
</head>
<body>
<input type="button" value="View message" onclick="message()" />
</body>
</html>

```

JavaScript Special Characters

In JavaScript you can add special characters to a text string by using the backslash sign.

Insert Special Characters

The backslash (\) is used to insert apostrophes, new lines, quotes, and other special characters into a text string.

Look at the following JavaScript code:

```
var txt="We are the so-called "Vikings" from the north.";
document.write(txt);
```

In JavaScript, a string is started and stopped with either single or double quotes. This means that the string above will be chopped to: We are the so-called

To solve this problem, you must place a backslash (\) before each double quote in "Viking". This turns each double quote into a string literal:

```
var txt="We are the so-called \"Vikings\" from the north.";
document.write(txt);
```


JavaScript will now output the proper text string: We are the so-called "Vikings" from the north.

The table below lists other special characters that can be added to a text string with the backslash sign:

Code	Outputs
\'	Single quote
\"	Double quote
\&	Ampersand
\\	Backslash
\n	New line
\r	Carriage return
\t	Tab
\b	Backspace
\f	Form feed

JavaScript Objects Introduction

JavaScript is an Object Oriented Programming (OOP) language.

Object Oriented Programming

JavaScript is an Object Oriented Programming (OOP) language. An OOP language allows you to define your own objects and make your own variable types. Note that an object is just a special kind of data. An object has properties and methods.

Properties

Properties are the values associated with an object. In the following example we are using the length property of the String object to return the number of characters in a string:

```
<script type="text/javascript">
var txt="Hello World!"
document.write(txt.length)
</script>
```

The output of the code above will be: 12

Methods

Methods are the actions that can be performed on objects. In the following example we are using the toUpperCase() method of the String object to display a text in uppercase letters:

```
<script type="text/javascript">
```

```
var str="Hello world!"
document.write(str.toUpperCase())
</script>
```

The output of the code above will be: HELLO WORLD!

```
<html>
<body>
<script type="text/javascript">
var txt="Hello World!"
document.write(txt.length)
</script>
</body>
</html>
```

JavaScript Date Object

The Date object is used to work with dates and times.

Examples

Return today's date and time

How to use the Date() method to get today's date.

- getTime()

Use getTime() to calculate the years since 1970.

- setFullYear()

use setFullYear() to set a specific date.

- toUTCString()

use toUTCString() to convert today's date (according to UTC) to a string.

- getDay()

Use getDay() and an array to write a weekday, and not just a number.

Complete Date Object Reference

The Date object is used to work with dates and times. We define a Date object with the new keyword. The following code line defines a Date object called myDate:

```
var myDate=new Date()
```



Note

The Date object will automatically hold the current date and time as its initial value!

Manipulate Dates

We can easily manipulate the date by using the methods available for the Date object.

In the example below we set a Date object to a specific date (14th January 2010):

```
var myDate=new Date()  
myDate.setFullYear(2010,0,14)
```

And in the following example we set a Date object to be 5 days into the future:

```
var myDate=new Date()  
myDate.setDate(myDate.getDate()+5)
```



Note

If adding five days to a date shifts the month or year, the changes are handled automatically by the Date object itself!

Comparing Dates

The Date object is also used to compare two dates. The following example compares today's date with the 14th January 2010:

```
var myDate=new Date()  
myDate.setFullYear(2010,0,14)  
var today = new Date()  
if (myDate>today)  
    alert("Today is before 14th January 2010")  
else  
    alert("Today is after 14th January 2010")
```

JavaScript Array Object

The Array object is used to store a set of values in a single variable name.

Create an array

Create an array, assign values to it, and write the values to the output.

```
<html>  
<body>  
<script type="text/javascript">  
var mycars = new Array()
```

```

mycars[0] = "M Benz"
mycars[1] = "Volvo"
mycars[2] = "BMW"

for (i=0;i<mycars.length;i++)
{
document.write(mycars[i] + "<br />")
}
</script>

</body>
</html>

```

For...In Statement

How to use a for...in statement to loop through the elements of an array.

```

<html>
<body>
<script type="text/javascript">
var x
var mycars = new Array()
mycars[0] = "M Benz"
mycars[1] = "Volvo"
mycars[2] = "BMW"

for (x in mycars)
{
document.write(mycars[x] + "<br />")
}
</script>
</body>
</html>

```

Join two arrays - concat()

How to use the concat() method to join two arrays.

```

<html>
<body>
<script type="text/javascript">
var arr = new Array(3)
arr[0] = "Jani"
arr[1] = "Tove"
arr[2] = "Hege"

```

```

var arr2 = new Array(3)
arr2[0] = "John"
arr2[1] = "Andy"
arr2[2] = "Wendy"
document.write(arr.concat(arr2))
</script>
</body>
</html>

```

Put array elements into a string - join()

How to use the join() method to put all the elements of an array into a string.

```

<html>
<body>

<script type="text/javascript">

var arr = new Array(3)
arr[0] = "Jani"
arr[1] = "Hege"
arr[2] = "Stale"

document.write(arr.join() + "<br />")
document.write(arr.join("."))

</script>

</body>
</html>

```

Literal array - sort()

How to use the sort() method to sort a literal array.

```

<html>
<body>
<script type="text/javascript">
var arr = new Array(6)
arr[0] = "Jani"
arr[1] = "Hege"
arr[2] = "Stale"
arr[3] = "Kai Jim"
arr[4] = "Borge"

```

```
arr[5] = "Tove"  
document.write(arr + "<br />")  
document.write(arr.sort())  
</script>  
</body>  
</html>
```

Numeric array - sort()

How to use the sort() method to sort a numeric array.

```
<html>  
<body>  
  
<script type="text/javascript">  
  
function sortNumber(a, b)  
{  
  return a - b  
}  
  
var arr = new Array(6)  
arr[0] = "10"  
arr[1] = "5"  
arr[2] = "40"  
arr[3] = "25"  
arr[4] = "1000"  
arr[5] = "1"  
  
document.write(arr + "<br />")  
document.write(arr.sort(sortNumber))  
  
</script>  
  
</body>  
</html>
```

The Array object is used to store a set of values in a single variable name.

We define an Array object with the new keyword. The following code line defines an Array object called myArray:

```
var myArray=new Array()
```

There are two ways of adding values to an array (you can add as many values as you need to define as many variables you require).

1:

```
var mycars=new Array()  
mycars[0]="M Benz"  
mycars[1]="Volvo"  
mycars[2]="BMW"
```

You could also pass an integer argument to control the array's size:

```
var mycars=new Array(3)  
mycars[0]="M Benz"  
mycars[1]="Volvo"  
mycars[2]="BMW"
```

2:

```
var mycars=new Array("Saab","Volvo","BMW")
```



Note

If you specify numbers or true/false values inside the array then the type of variables will be numeric or Boolean instead of string.

Accessing Arrays

You can refer to a particular element in an array by referring to the name of the array and the index number. The index number starts at 0.

The following code line: `document.write(mycars[0])` will result in the following output:

M Benz

Modify Values in Existing Arrays

To modify a value in an existing array, just add a new value to the array with a specified index number: `mycars[0]="Opel"`

Now, the following code line: `document.write(mycars[0])` will result in the following output:

Opel

JavaScript Math Object Reference

Math Object Methods

abs(x)	Returns the absolute value of a number
acos(x)	Returns the arccosine of a number
asin(x)	Returns the arcsine of a number
atan(x)	Returns the arctangent of x as a numeric value between -PI/2 and PI/2radians
atan2(y,x)	Returns the angle theta of an (x,y) point as a numeric value between -PI and PI radians
ceil(x)	Returns the value of a number rounded upwards to the nearest integer
cos(x)	Returns the cosine of a number
exp(x)	Returns the value of E^x
floor(x)	Returns the value of a number rounded downwards to the nearest integer
log(x)	Returns the natural logarithm (base E) of a number
max(x,y)	Returns the number with the highest value of x and y
min(x,y)	Returns the number with the lowest value of x and y
pow(x,y)	Returns the value of x to the power of y
random()	Returns a random number between 0 and 1
round(x)	Rounds a number to the nearest integer
sin(x)	Returns the sine of a number
sqrt(x)	Returns the square root of a number
tan(x)	Returns the tangent of an angle
toSource()	Represents the source code of an object
valueOf()	Returns the primitive value of a Math object

Math Object Properties

constructor	A reference to the function that created the object
E	Returns Euler's constant (approx. 2.718)
LN2	Returns the natural logarithm of 2 (approx. 0.693)
LN10	Returns the natural logarithm of 10 (approx. 2.302)
LOG2E	Returns the base-2 logarithm of E (approx. 1.414)
LOG10E	Returns the base-10 logarithm of E (approx. 0.434)
PI	Returns PI (approx. 3.14159)
prototype	Allows you to add properties and methods to the object
SQRT1_2	Returns the square root of 1/2 (approx. 0.707)
SQRT2	Returns the square root of 2 (approx. 1.414)

JavaScript and Form Objects

The form is inside the document body and within the form are four input tags that represent three text fields and one submit button. Note that not only do the three text input buttons contain Name attributes but so does the form tag. Naming the tags allows them to be referred to by name in a JavaScript statement.

```
<HTML>
<HEAD>
<TITLE>Simple Personal Information Form</TITLE>
</HEAD>
<BODY>
<H3>Personal Information Form</H3>
<FORM Action="http://www.ryerson.ca/cgi-bin/mirror/mirror.cgi"
      Name="person">
  <PRE>
    First Name: <INPUT Type="TEXT" Name="fName">
    Last Name: <INPUT Type="TEXT" Name="lName">
    Student Number: <INPUT Type="TEXT" Name="sNumber">
                  <INPUT Type="Submit">
  </PRE>
</FORM>
</BODY>
</HTML>
```

Here is the form as it would appear inside an HTML page:

Personal Information Form

Top of Form

First Name:

Last Name:

Student Number:

Referring to Form Objects

Whenever a form is placed within a document a JavaScript object is created with attributes or properties that represent the properties of the form – including its name and action attributes. The input tags within the form are also represented by JavaScript objects that contain properties that reflect (or represent) the properties or attributes of the input tags within the document. One of the most important of these is the value property that contains the text that has been entered into an input text field. As these objects reflect tags that are within other tags in the body of the document, they are made available in JavaScript as nested objects. For

example a form object will be available as a property of the document object and an input object will be available as the property of the form object. Here is an example:

```
myForm = document.person
```

Provided there is a form named person in the document, this statement will result in the myForm variable containing a reference to the form object that reflects the form tag. Once the myForm variable contains a reference to the form object, it can be used to access the properties of the form object. In this example:

```
myForm = document.person
```

alert(myForm.action)

The alert dialog would display the text: <http://www.ryerson.ca/cgi-bin/mirror/mirror.cgi> which is the action attribute of the form tag in the example above. The input objects are properties of the form object which is a property of the document object so: myInput = document.person.fNname will result in the variable myInput containing a reference to the input object that reflects the input tag with the Name="fNname" attribute in the example above.

```
myInput = document.person.fNname
```

alert(myInput.value)

After these statements, the myInput variable will contain a reference to the input object named fNname and can now be used to access the value property of the object. In this example the alert statement will display whatever text has been entered into the fNname field. It is also possible to write text into an input text field this way:

```
myInput = document.person.fNname  
myInput.value = "John"
```

Here is yet another way to do this:

```
document.person.fname.value = "John"
```

For these examples to work these code samples must be executed after the form has appeared in the document and the JavaScript interpreter has already created the form and input objects that reflect them. This is usually done in a function that is only called when data in a form must be validated before being sent to a server or when data in a form must be checked or manipulated for some other reason

JavaScript: Forms and Functions

JavaScript extends the functionality of HTML forms. It allows script writers to write code that can check user input and display additional information to the user. There are two parts to how JavaScript does this. First, JavaScript permits the declaration of named functions

much like in C/C++/Java. These functions, designed to perform a specific task such as checking user input for errors, are called when certain events occur. These events may be the user pressing a submit button in an HTML form or merely placing the cursor over a hypertext link. Second, special HTML tag attributes called event handlers can be placed in HTML tags. When an event occurs that the attribute is designed to handle, a short script in the handler is executed. Usually, event handlers contain one or two statements that just call a function or two. To make this work you must both define the function to correctly get information from the form and place the call to the function in an event handler in an HTML form.

Simple Form without JavaScript

The following is the code for a simple HTML form. It contains a single text field, a submit button, and a reset button. In this form no matter what text is entered into the text field, it is sent on to the server when the submit button is pressed. This is a waste of server-side resources.

```
<HTML>
<head>
<title>simple student number form</title>
</head>
<body>

<form action="http://www.ryerson.ca/cgi-bin/mirror/mirror.cgi">
<h3>a simple form</h3>
<p>enter your ryerson student number:
<input type="text" value="1234567"><br>
<input type="submit">
<input type="reset"></p>
</form>
</body>
</html>
```

A Simple Form with JavaScript

The following code has two additions (a function and an event handler) that make it possible to check what the user enters into the student number text field. In the `<form>` tag an event handler has been added. The handler calls a function (normally in the head of the document) when the user submits the form. This function checks what was entered into the field. If the user enters a 9 character string, containing only the numbers 0 through 9, the form is submitted. If they did not, the form is not submitted. Try it. Of course this doesn't work if JavaScript has been turned off. For completeness the entire HTML source is shown.

```

<HTML>
<HEAD>
<TITLE>Simple Student Number Form</TITLE>
<SCRIPT Language="JavaScript">
<!--

function checkStdNumber(stdNumber){
    var i
    var chr
    var errorMsg = "Please enter your student number."

    if (stdNumber.length != 7){
        alert(errorMsg)
        return false
    }

    for (i=0; i < stdNumber.length; i++){
        chr = stdNumber.charAt(i)
        if (chr > "7" || chr < "0"){
            alert(errorMsg)
            return false
        }
    }

    return true
}

//-->
</SCRIPT>
</HEAD>
<BODY>

<FORM ACTION="http://www.ryerson.ca/cgi-bin/mirror/mirror.cgi"
    NAME="simpleForm"
    onSubmit="return checkStdNumber(this.sNumber.value)">
<H3>A Simple Form</H3>
<P>Enter your Student Number:
<INPUT TYPE="TEXT" VALUE="1234567" NAME="sNumber"><BR>
<INPUT TYPE="SUBMIT">
<INPUT TYPE="RESET"></P>
</FORM>
</BODY>
</HTML>

```

Detailed Explanation

The following is a detailed explanation of how each part of the HTML and JavaScript in this page work together to check the student number text field for errors. The logic of how the function evaluates the string is not described here.

The `checkStdNumber()` function

The function named `checkStdNumber()` is passed a text string in the variable named `stdNumber`. If the string is nine characters long and contains only the text characters 0 through 9 the function returns the value `true` otherwise it returns the value `false`. There is nothing unusual in the function itself – it is like any other JavaScript function. For example – although it would not be useful – it could be called this way: `result = checkStdNumber("Hi")`, in which case the variable `result` would hold the value `false` because "Hi" will be copied into the `stdNumber` variable in the function's parameter list. Since it only has two characters the function will return `false`. The value the function returns will take the place of the function in the statement – the function is evaluated and returns a value - and in this case `false` will be returned and assigned to the variable named `result`.

Of course the script and HTML together do more than this. When the user presses the submit button the function is called with the value the user entered in the text field as a parameter. If the function returns `true` the form is submitted but if the function returns `false` the form is not submitted. When someone presses the submit button an "event" that the browser recognizes occurs. It is possible to have the browser invoke code designed to respond to the event. The mechanisms used to tie an event to code to be executed when it occurs are called event handlers. One way to create an event handler is to use a special HTML attributes designed just for this purpose. In this case an event handler attribute of the form tag named `onSubmit` can be used. `onSubmit` is defined as part of HTML 4 for this purpose. Here is the `onSubmit` attribute from the example again:

```
onSubmit="return checkStdNumber(this.sNumber.value)"
```

Here is a description of each part of the event handler. Each bolded part is discussed separately:

```
onSubmit="return checkStdNumber(this.sNumber.value)"
```

The format for an event handling attribute is the same as the format for any HTML tag attribute. The attribute name is followed by an equals sign and then a value in quotes. In this case what is inside the quotes are JavaScript statements to be executed when the event occurs. Several events are defined for certain form tags.

`onSubmit` is defined for the form tag itself. The JavaScript statements in quotes are executed when the form is submitted.

```
onSubmit="return checkStdNumber(this.sNumber.value)"
```

`checkStdNumber()` is a function call.

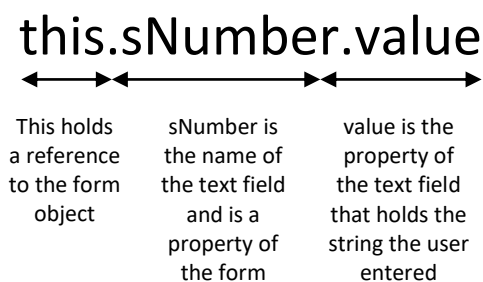
In order to pass the value returned by the `checkStdNumber()` function back to the event handler, the `return` keyword is added. If you omit this, the form will be submitted regardless of what the function returns.

```
onSubmit="return checkStdNumber(this.sNumber.value)"
```

Inside the event handler `this.sNumber.value` is passed into the `checkStdNumber()` function. The dot notation indicates that `sNumber` is a property of the object and that `value` is a property of the `sNumber` object. But what is `this`? Each form tag in an HTML document is represented by a JavaScript object that allows script writers to examine properties of the form object. This is a special keyword that is used to refer to the JavaScript object that represents the tag an attribute is within. Since this appears inside an attribute of the form tag it represents the form tag. Since the text field is an input tag named `sNumber` inside the form tag it can be accessed by `this.sNumber`, and since each input field has a property named `value` that holds the current string in the text field it can be retrieved with `this.sNumber.value`.

In other words,

- the text field has a property named `value`;
- the text field is a property of the form it is in and can be referenced by name;
- inside the form tag `onSubmit` attribute, `this` is a variable that holds a reference to the form object.



Therefore `this.sNumber.value` will retrieve the value in the text field.

Handling Events

Events occur when the user does something with the browser. This may include moving the mouse over a hypertext link, clicking a radio button in a form, or loading a page into a window. A number of these events can trigger JavaScript code, giving script writers the ability to respond to user actions when they occur. Events that occur when a user manipulates an HTML form are normally handled using HTML event handler attributes. These attributes are an extension to HTML that provide a method for the browser to invoke JavaScript code when the events they refer to occur.

If an event handler has been added into the appropriate HTML tag, every time the event it refers to occurs, the code in the handler will be invoked. The general format for this is:

```
<HTMLTAG eventHandler="statement1; statement2; statmentn">
```

Note that the semicolons separate the different JavaScript statements allowing more than one to be placed in the same line.

JavaScript Form Validation

JavaScript can be used to validate input data in HTML forms before sending off the content to a server.

Form data that typically are checked by a JavaScript could be:

- has the user left required fields empty?
- has the user entered a valid e-mail address?
- has the user entered a valid date?
- has the user entered text in a numeric field?

Required Fields

The function below checks if a required field has been left empty. If the required field is blank, an alert box alerts a message and the function returns false. If a value is entered, the function returns true (means that data is OK):

```
function validate_required(field,alerttxt)
{
with (field)
{
if (value==null||value=="")
{alert(alerttxt);return false}
else {return true}
}
}
```

The entire script, with the HTML form could look something like this:

```
<html>
<head>
<script type="text/javascript">
function validate_required(field,alerttxt)
{
with (field)
{
if (value==null||value=="")
{alert(alerttxt);return false}
else {return true}
}
}
```

```

}
function validate_form(thisform)
{
with (thisform)
{
if (validate_required(email,"Email must be filled out!")==false)
{email.focus();return false}
}
}
</script>
</head>
<body>
<form action="submitpage.htm"
onsubmit="return validate_form(this)"
method="post">
Email: <input type="text" name="email" size="30">
<input type="submit" value="Submit">
</form>
</body>
</html>

```

E-mail Validation

The function below checks if the content has the general syntax of an email. This means that the input data must contain at least a @ sign and a dot (.). Also, the @ must not be the first character of the email address, and the last dot must at least be one character after the @ sign:

```

function validate_email(field,alerttxt)
{
with (field)
{
apos=value.indexOf("@")
dotpos=value.lastIndexOf(".")
if (apos<1||dotpos-apos<2)
{alert(alerttxt);return false}
else {return true}
}
}

```

The entire script, with the HTML form could look something like this:

```

<html>
<head>

```



```

<script type="text/javascript">
function validate_email(field,alerttxt)
{
with (field)
{
apos=value.indexOf("@")
dotpos=value.lastIndexOf(".")
if (apos<1||dotpos-apos<2)
{alert(alerttxt);return false}
else {return true}
}
}
function validate_form(thisform)
{
with (thisform)
{
if (validate_email(email,"Not a valid e-mail address!")==false)
{email.focus();return false}
}
}
</script>
</head>
<body>
<form action="submitpage.htm"
onsubmit="return validate_form(this);"
method="post">
Email: <input type="text" name="email" size="30">
<input type="submit" value="Submit">
</form>
</body>
</html>

```

JavaScript Animation

With JavaScript we can create animated images.

It is possible to use JavaScript to create animated images. The trick is to let a JavaScript change between different images on different events.

In the following example we will add an image that should act as a link button on a web page. We will then add an onMouseOver event and an onMouseOut event that will run two JavaScript functions that will change between the images.

The HTML Code

The HTML code looks like this:

```
<a href="http://www.w3schools.com" target="_blank">

</a>
```

The onmouseover event tells the browser that once a mouse is rolled over the image, the browser should execute a function that will replace the image with another image.

The onmouseout event tells the browser that once a mouse is rolled away from the image, another JavaScript function should be executed. This function will insert the original image again.

The JavaScript Code

The changing between the images is done with the following JavaScript:

```
<script type="text/javascript">
function mouseOver()
{
document.b1.src = "b_blue.gif"
}
function mouseOut()
{
document.b1.src = "b_pink.gif"
}
</script>
```

The function mouseOver() causes the image to shift to "b_blue.gif".

The function mouseOut() causes the image to shift to "b_pink.gif".

```
<html>
<head>
<script type="text/javascript">
function mouseOver()
{
document.b1.src = "b_blue.gif"
}
function mouseOut()
```

```

{
document.b1.src="b_pink.gif"
}
</script>
</head>

<body>
<a href="http://www.w3schools.com" target="_blank">
</a>
</body>
</html>

```

1-4.3 Server side scripting- Common Gateway Interface (CGI)

Up until now, we have looked at forms only from the perspective of the browser but the real work with the forms occurs at the server. Remember that the form is a tool for collecting and/or delivering data from the user. In most cases, after the data are collected, the browser packages them up, and then ships them to the server for processing. The server runs a set of programs that takes data from the browser, processes them and passes them to a script that does something with the data.

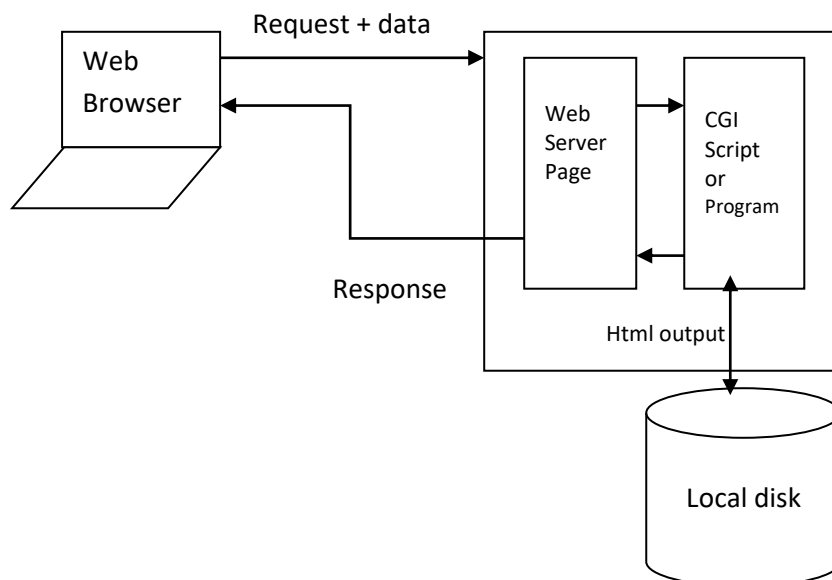


Figure 24: How CGI works

The set of programs that handle the data formatting and passing is called the Common Gateway Interface (CGI). The user-program that processes the user data and sends it back to

the user is called a script. A script is the name we give to programs written by the web weaver that do the actual data processing.

It may simply thank users for their input. Or it may run a sophisticated search program, retrieve some collection of data from a database and create an HTML document to present those data back to the users. Actually, the only limitation on what scripts can do is those imposed by imagination of the scriptors!

CGI provides the standard and format that the browser uses to send data to a server as well as format the server uses to hand the data off to a script. That program usually called the CHI script does whatever is necessary to process data and send something back to the user.

CGI scripts can be written in either the Perl or C programming languages or as a Shell script on your Unix machine. Although Perl and C are the most common high-level languages used for CGI scripting, theoretically any high-level language can be used for this purpose.

How CGI Works

CGI is a collection of programs the Web server uses to communicate with scripts on the server. The most common use of CGI is processing forms, but there are other sophisticated ways to use CGI to pass data. However, we will direct our attention to the use of CGI to process data from forms. Following are steps for using a CGI script on a server to process a form.

1. The browser request the form from the server
2. The user fills the form and activate the submit control
3. The browser sends the form to the server
4. The server recognizes the CGI call and passes the script name and the associated data to set of programs called CGI
5. The CGI application massages the data and creates a set of environmental variables then starts the request scripts
6. The CGI scripts run, usually generating a response to the user along with the other processing
7. The CGI software passes the response created by the script back to the server
8. The server passes the processed data and response back to the browser
9. The browser displays the processed data and response to the user

The only parts of this process that concern the Web weaver are the form the user fills out and the CGI script that processes data and sends them back along with a proper response to the user. The rest can be considered part of the “magic of http”. When a server gets a request for a CGI script, the server handles it differently than the request for just another Web page. Rather than posting the requested document to the browser, the server looks for the script, or program, specified in the request, or call, and tries to run that CGI script.

For example, the following line <http://www.bugsbeewee.com/cgi-bin/vote.cgi?goodbug=Ib> asks the server, <http://www.bugsbeewee.com> to invoke the script called vote.cgi that resides in the /cgi-bin directory.

Part of preparing the CGI script to run is passing the data from the client to the script. In our example, the data here is a single variable (goodbug=Ib). This format for sending data illustrates the method="GET" process. If the method were the "POST", the data would not be sent as part of the URL. Different platforms and different operating systems dictate different ways that the CGI script gets their data.

ADVANTAGES OF CGI

- Simplicity
- Language independence: CGI scripts can be written in nearly every language
- Process isolation: Web server or CGI applications run in separate process. Buggy applications cannot crash the web server.
- Open standard: Implemented on every web server
- Many interactive possibilities
- Connect Web with other databases
- Pages created on the fly
- Many available for free

DISADVANTAGES OF CGI

- Security: can access anything on the local machine
- Performance/scalability: For every request, a new person is created
- More severe processing
- More complex than HTML

Using a form with method = "GET"

- The data is URL encoded and appended to the URL (of action attribute) by the browser
- The data is passed in the QUERY_STRING variable
- The total length of a URL is limited to something between 256 characters and 4KB → limit for data size
- The data is collected from the form and is transmitted as a single string of name value pairs separated by &:
Name1=value1&name2=value2&.....
e.g. Name=James+Bond&address=London&profession=other&.....
- The output data is cached by the browser

Using a form with method = “post”

- The data is URL encoded but not appended to the URL
- The data is transmitted as single string of name-value pairs
- The size of the data is passed to the CGI program in the environment variable CONTENT_LENGTH
- The data itself is passed in standard input (STDIN)
- There is no size limitation for the data
- The output data is not cached by the browser.

1-4.4 Servlets and Java Server Pages

Servlets commonly are used when a small portion of the content sent to the client is static text and markup. In fact, some servlets do not produce content. Rather, they perform a task on behalf of a client, and then invoke other servlets or JSPs to provide a response. Note that in most cases, servlet and JSP technologies are interchangeable. The server that executes a servlet is referred to as the servlet container or servlet engine. Java Server Pages Technology, an extension of servlet technology, simplifies the process of creating pages by separating presentation from content. Normally, JSPs are used when most of the content sent to the client is static text and markup, and only a portion of the content is generated dynamically with Java code.

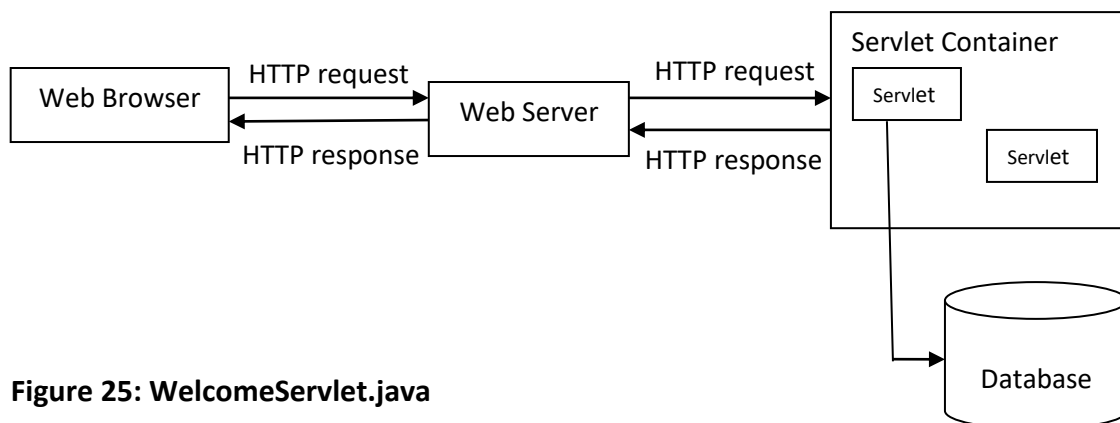


Figure 25: WelcomeServlet.java

```
// A simple servlet to process get request
import javax.servlet.ServletException;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.io.IOException;
import javax.io.PrintWriter;
```

```

public class WelcomeServlet extends HttpServlet
{
    // process "get" requests from clients
    protected void doGet( HttpServletRequest request
    HttpServletResponse response )
    throws ServletException, IOException
    {
        response.setContentType( "text/html" );
        PrintWriter out = response.getWriter();

        // send XHTML page to client

        // start XHTML document
        out.println( "<?xml version = '1.0'?>" );

        out printf( "%s%s%s", "<!DOCTYPE html PUBLIC",
        "\"-//W3C/DTD XHTML 1.0 Strict//EN\"",
        "\"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd>\n" );

        out.println( "<html xmlns = http://www.w3.org/1999/xhtml>" );

        // head section of document
        out.println( "<head>" );
        out.println( "<title>A Simple Servlet Example</title>" );
        out.println( "</head>" );

        //body section of document
        out.println( "<head>" );
        out.println( "<title>A Simple Servlet Example</title>" );
        out.println( "</head>" );

        //end XHTML document
        out.println( "</html>" );
        out.close(); // close stream to complete the page
    } // end method doGet
} // end class WelcomeServlet

```

JAVASERVER PAGES (JSP)

JavaServer pages simplify the delivery of dynamic Web content. They enable Web application programmers to create dynamic content by reusing predefined components and by interacting with components using server-side scripting.

There are four key components to JSPs – directives, actions, scripting elements and tag libraries. **Directives** are messages to the JSP container – the server components that executes JSPs – that enable the programmer to specify page settings, to include content from other resources and to specify custom tag libraries for use in a JSP.

Actions encapsulate functionality in predefined tags that programmers can embed in a JSP. Actions often are performed based on the information sent to the server as part of a particular client response. They also can create Java objects for use in JSP scriptlets.

Scripting elements enable programmers to insert JAVA code that interacts with components in a JSP (and possibly other Web application components) to perform request processing. Scriptlets, one kind of scripting element, contain code fragments that describe the action to be performed in response to a user request.

Tag libraries are part of the tag extension mechanism that enables programmers to create custom tags. Such tags enable Web page designers to manipulate JSP content without prior Java knowledge.

Example

```
<?xml version = "1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd>

<!-- Figure 27.4: welcome.jsp -->
<!-- JSP that processes a "get" request containing data. -->

<html xmlns = http://www.w3.org/1999/xhtml>

<!-- head section of document -->
<head>
<title>Processing "get" requests with data</title>
head>
<!-- body section of document -->
<body>
<% // begin scriptlet

String name = request.getParameter( "firstName" );

if ( name != null )
{
%> <!-- end scriptlet to insert fixed template data --%>
```



```

<h1>
Hello <%= name %>, <br />
Welcome to JavaServer Pages!
</h1>

<% // continue scriptlet

} // end if
else {
%> <%- - end scriptlet to insert fixed template data - -%>

<form action = "welcome.jsp" method = "get">
<p>Type your first name and press Submit</p>

<p><input type = "text" name = "firstName" />
<input type = submit" value = "Sumit" />
</p>
</form>
<% // continue scriptlet

} // end else

%> <%- - end scriptlet - -%>
</body>

</html> <!-- end XHTML document - ->

```

SESSION 2-4: Introduction to web application using Active Server Pages.Net (ASP.NET)

2-4.1 ASP.Net

ASP.NET is Microsoft's application programming approach for the development of web site application software. This is a development approach that competes with **JAVA programming**.

Software for ASP.NET Programming

Software requirements to develop ASP.NET applications are summarized in this table. This includes the Windows operating system, Visual Studio software, database management system, and web browser.

Client Computer	Server Computer
Windows 2000 or later (including Windows XP Pro or Vista)	Windows 2000 Server or later
Microsoft .NET Framework 2.0 (downloadable from Microsoft for free, also included in Visual Studio)	Microsoft .NET Framework 2.0
Web Browser, e.g., Internet Explorer (6.0 or later), Mozilla Firefox, Netscape Navigator	Internet Information Services 5.0 or later (6.0 recommended)
Visual Studio 2005/2008	Microsoft SQL Server (2005/2008) database management system. FrontPage Server Extensions (for remote development).

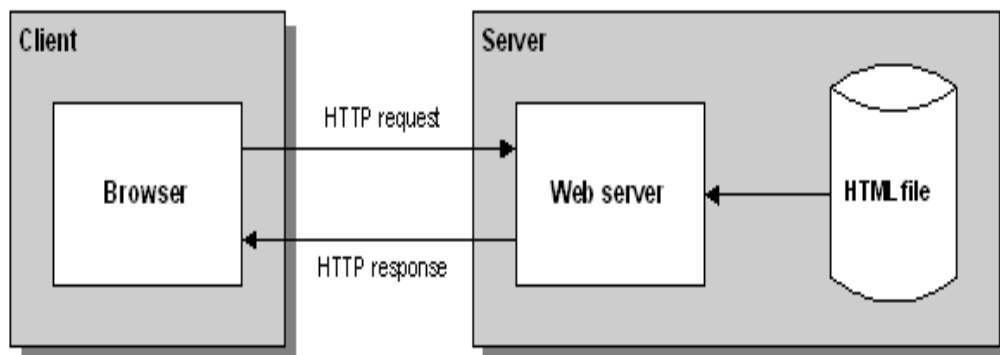
The current version of Visual Studio is the 2008 release; however, we may use either the 2005 or 2008 versions. We will use the Professional Edition and the Visual Basic programming language for web development coding.

Database support for web sites is provided through any major relational database management system such as **Oracle**, **Microsoft SQL Server**, **Microsoft Access**, and others.

2-4.2 Differences between Static and Dynamic Web Pages

Static – web pages that do not respond to user input. Examples include Hypertext Markup Language (HTML) documents.

This Figure depicts the interaction between a client and server for a static web page. The Hypertext Transfer Protocol (HTTP) request from the client is sent to the server. The server runs Web Server software and accesses a HTML file that stores the static document. The HTTP response is returned to the client where the Web Browser software formats the response and displays the document.



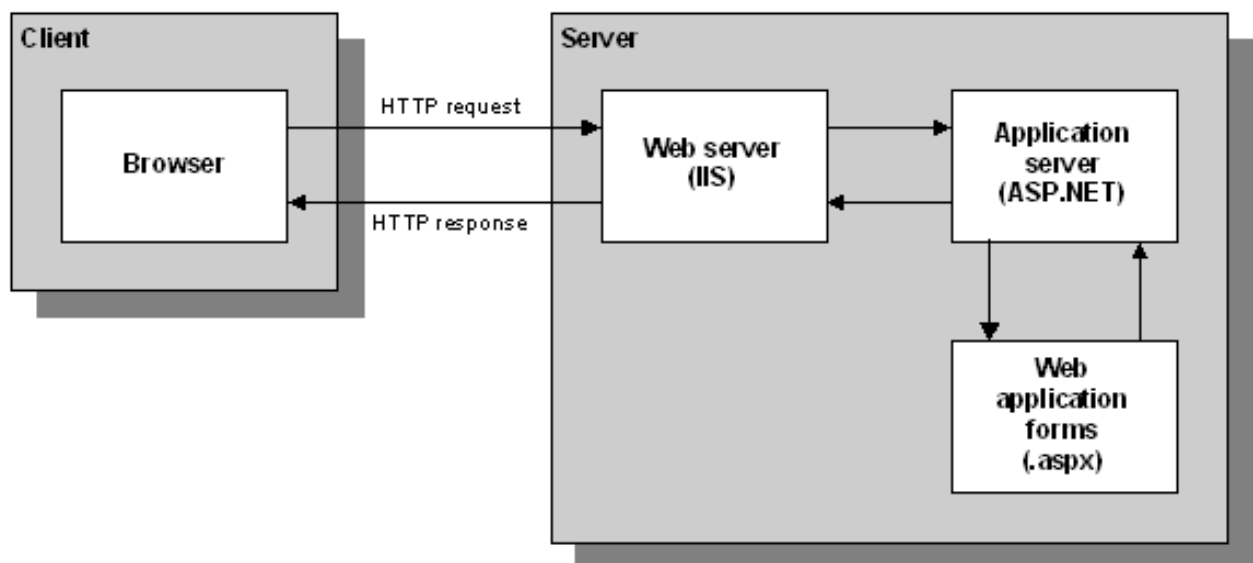
Dynamic – web applications that include one or more web pages where the page contents may change each time data displayed on the page changes. An example is an online shopping cart application.

Dynamic web pages contain server controls (Labels, TextBoxes, DropDownLists, and Buttons) that provide for application interaction by the application user.

This Figure depicts processing a dynamic page. The HTTP request from the client is sent to the server. Here the server computer is running the Internet Information Services (IIS) Web Server software. The Web Server software compares the requested file's "extension" against a list of application mappings to determine which application server needs to process the HTTP request. For .aspx extensions, the request goes to the ASP.NET application server.

The application server executes the requested web site form and generates a corresponding HTML document which is returned to the application server, which is in turn returned by to the Web Server, which in turn returns the document to the client's Browser.

The HTTP response is returned to the client where the Web Browser software formats the response and displays the document.



If the application user interacts with the web page by using the server controls on the page, a post back of the page to the server may occur and the process is repeated. The request-to-response process is termed a round trip.

How ASP.NET Applications Execute

The .NET Framework is the basis for the execution of ASP.NET applications.

Applications are developed using one of the .NET languages. The .NET Framework includes the Class Library and Common Language Runtime (CLR).

The Class Library consists of a hierarchy of classes organized into *namespaces* – the ASP.NET web program classes are stored in the System Web namespace.

The CLR manages execution of .NET programs – it coordinates memory management, code execution, security, and other aspects of the execution process.

The CLR includes the Common Type System – this specifies that all .NET applications use the same data types regardless of the programming language.

The Microsoft Intermediate Language (MSIL) is the language to which all .NET languages compile for the creation of an *assembly* – the assembly is executed by the CLR.

ASP.NET Development Environments

There are three different ways that ASP.NET applications are developed, coded, and tested.

Standalone Environment – the single computer serves as both the client and server. A Development Server (web) is provided with Visual Studio to run applications for local testing where IIS is not installed. This environment can also include use of SQL Server Express Edition for a database management system.

Typical Files in an ASP.NET Application

App_Code folder – used to store files that represent business classes. Business class files have a .vb extension – they are Visual Basic files.

Web form files – have an .aspx extension – this code represents the design layout of the form – includes standard HTML plus ASP tags to define the form's server controls.

Code-Behind files – have an .aspx.vb extension – this is Visual Basic code that is behind the web form.

Other files and folders, e.g., image files, database files.

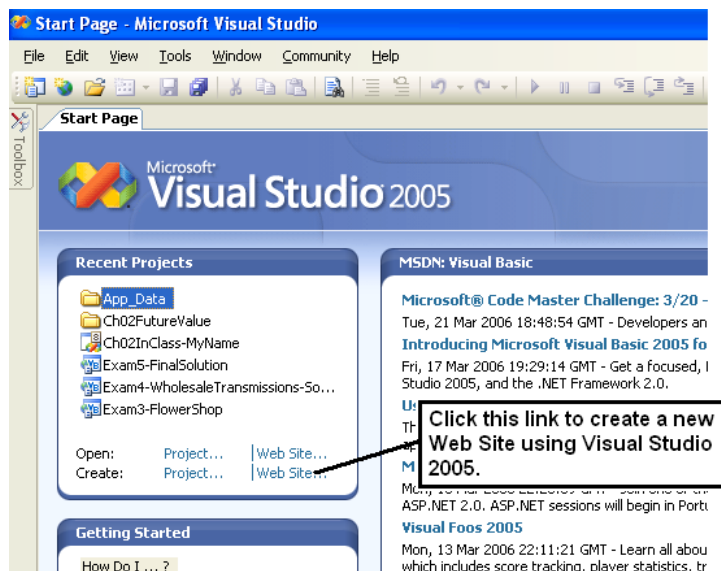
Thus, each web form has two files – the .aspx and the .aspx.vb files.

2-3.4 Working with ASP.NET Web Sites

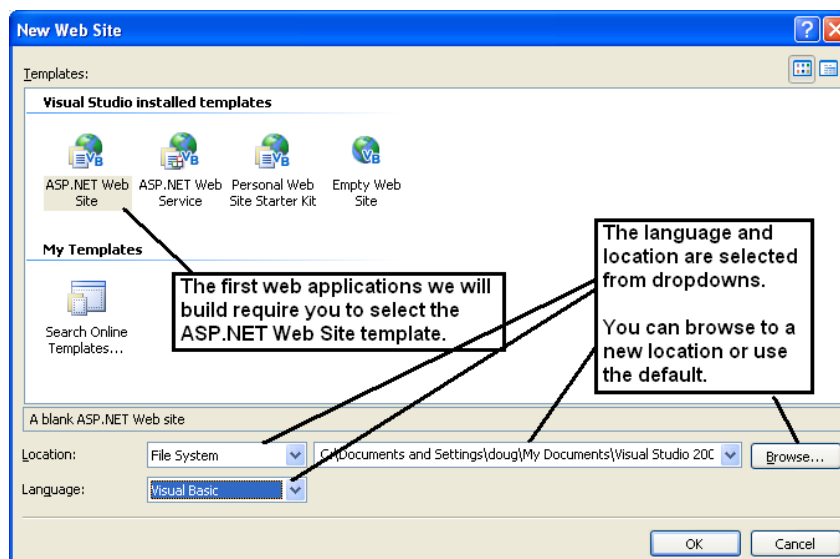
The Visual Studio Integrated Development Environment (IDE) is used to visually build a web page. A visual studio web application is called a web site. Sometimes the term web project is also used.

Start a web application using ASP.Net

The start page for Visual Studio is shown here. Note the link you click to create (or open an existing) web site.



The New Web Site dialog box displays as shown in the next Figure.



Installed Templates – there are several template options. Initially we will use the ASP.NET Web Site template – Visual Studio uses this template to generate the initial files and folders that comprise a web site.

Location Options – there are three location options for ASP.NET web sites:

File System – Created in a folder on your local computer (or network shared folder). Web site runs in the built-in development server or an IIS virtual directory folder.

HTTP – web site is created under control of an IIS web server on your local computer or a local area network server.

FTP – web site is created on a remote hosting server by uploading the site files to that server using FTP.

Web Site Directory – use the Browse button to select a folder if you do not want to use the default. The files for a File System Web Site are stored in different locations.

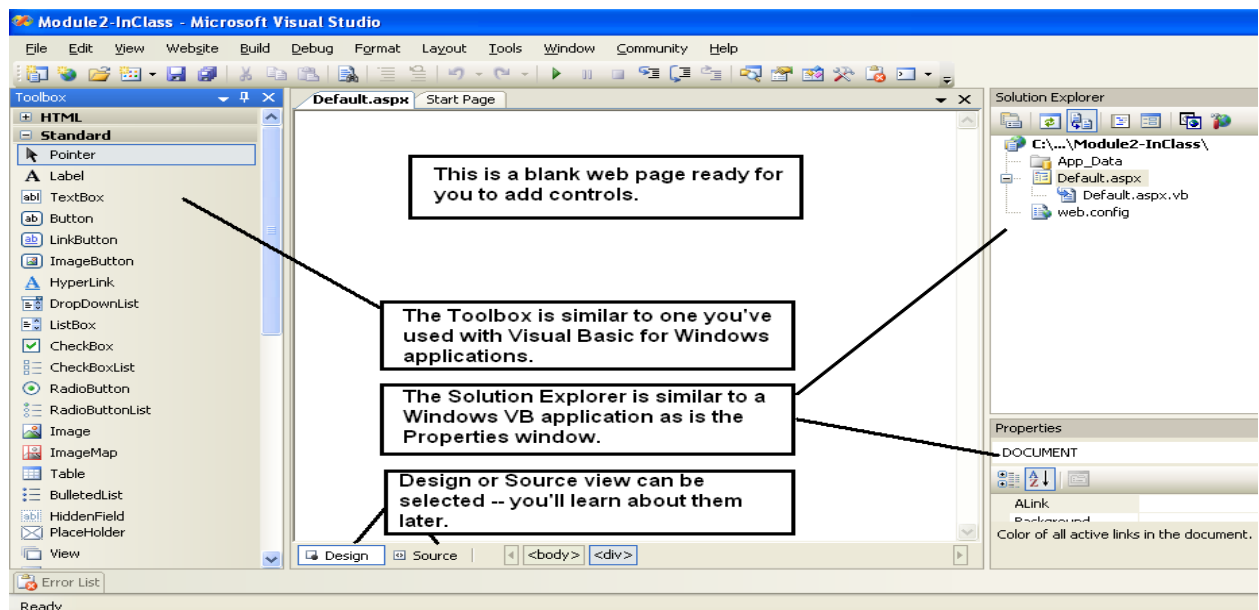
Web site name – you need to change the default to give the web site a meaningful name – the default is something like WebSite1, WebSite2 – not very meaningful.

Web.Config file – project information is stored in a web.config file (older versions of ASP.NET used a project file).

Solution file – the Solution file (.sln) is stored by default in the location: My Documents\Visual Studio2005\Projects – you can change this default by using Tools->Options, then expand the Project and Solutions node, select General Category, and enter the location you prefer.

The Web Site IDE

This Figure shows the initial IDE for a web site. Note the similarity to a Windows VB project IDE. The web page in the middle is where you place controls to build a web form.



Note the Solution Explorer entries:

- App_Data – stores application data
- Default.aspx – the first web form for the site – contains HTML and asp code to define the form.
- Default.aspx.vb – the VB code behind file – stores the VB code you write for events for the web form.

Web Form Designer – this is the blank space in the middle of the IDE where you design the form's layout. It displays in two different views:

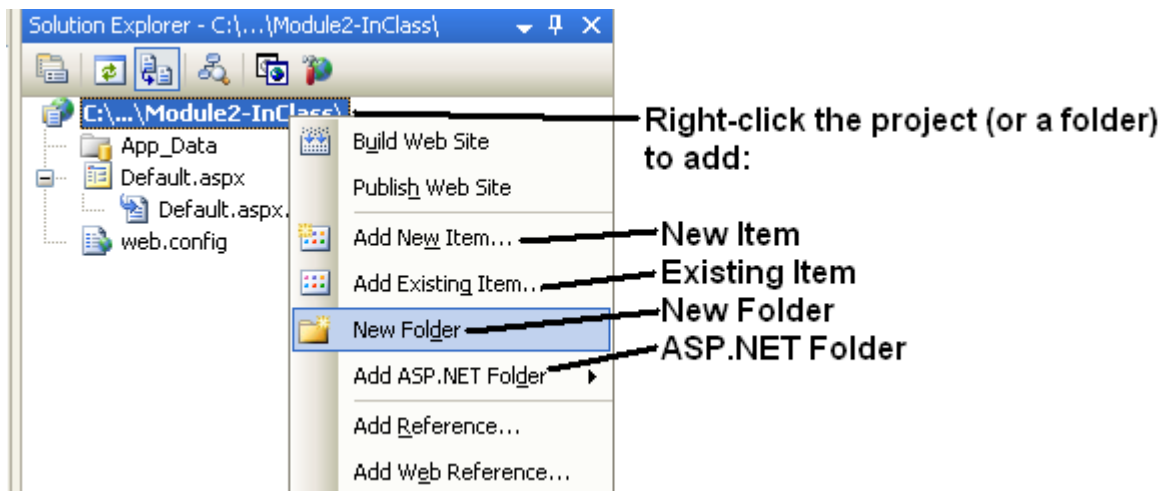
- Source View – displays the HTML code for the page.
- Design View – displays the form in form view.

IDE Layout – the IDE works much like the VB Windows IDE – you can:

- Hide and unhide windows – click the Auto Hide button of a window to hide the window. Close windows by clicking the close button in the upper right corner of each window.
- Use the View menu to restore windows.
- Use the Window menu – Reset Window Layout option to restore the IDE windows to the default configuration.

Add Folders and Existing Files to a Web Site

Web sites often require you to add additional folders to store files such as graphics and other web site objects that have already been created.

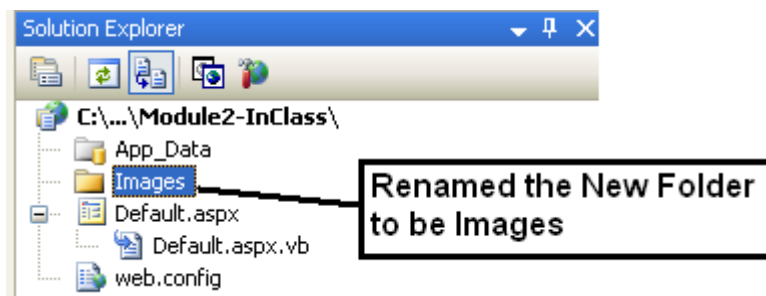


To add a folder:

In Solution Explorer, right-click the project (or folder) to which you want to add a folder – choose New Folder – name the folder.

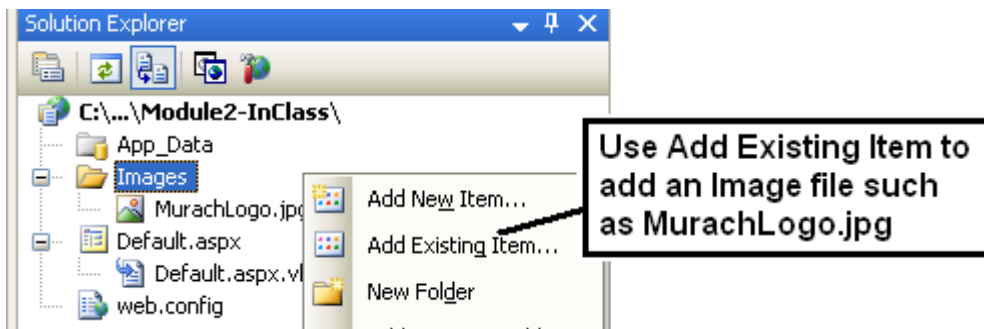
To add a special ASP.NET folder:

In Solution Explorer, right-click the project – choose Add ASP.NET folder.



To add an existing item, such as a predefined business class object, database file, or image:

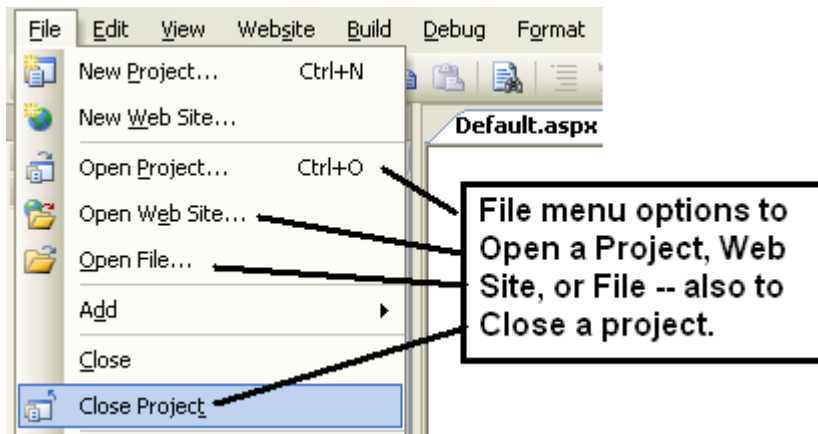
- In Solution Explorer, right-click the project or folder to which you wish to add an item.
- Select Add Existing Item and navigate/locate the item desired.



Open and Close an Existing Web Site

The ways to accomplish these common tasks are:

- Use the File menu, Open Project option.
- Use the File menu, Recent Projects option.
- Use the File menu, Open Web Site option.
- Use the File menu, Close Project option.



The first three options are available from the Microsoft Visual Studio Start page.

The Open Web Site dialog box is used to:

- Open a file-system web site by selecting File System (left side of dialog box) and then use the File System tree to locate the web site.
- For IIS managed sites, use the File System Tree or click Local IIS and select the web site from the list.

Using Design View

In this section we focus on building a web form by using Design View.

The form to be built is shown in this Figure. The form is used to calculate the future value of a monthly investment at a selected interest rate for a set time period.

401K Future Value Calculator

Monthly investment:	<input type="text" value="Unbound"/>	
Annual interest rate:	<input type="text"/>	
Number of years:	<input type="text"/>	
Future value:	<input type="text" value="Label"/>	
<input type="button" value="Calculate"/>	<input type="button" value="Clear"/>	

Interest rate is required. Interest rate must range from 1 to 20.
Number of years is required. Years must range from 1 to 45.

Web Server and HTML Server Controls

Initially you will need to learn to use web server controls – these are derived from ASP.NET classes in the .NET Framework Class Library and are available from the Toolbox window.

- DropDownList – used to select items from a drop down window.
- TextBox – used to enter values or to display values as output.
- Label – used to display values as output or for data entry prompts.
- Button – used to post back a form to a server or to execute code behind the form associated with an event. A post back is when the data on the form is sent to the server, and then the calculated values are returned by the server to post back the page within the web browser software.

HTML server controls are used to display an image on the form and to display text as a heading on the form. A HTML table can be used to format the layout display of controls on the form.

Flow Layout

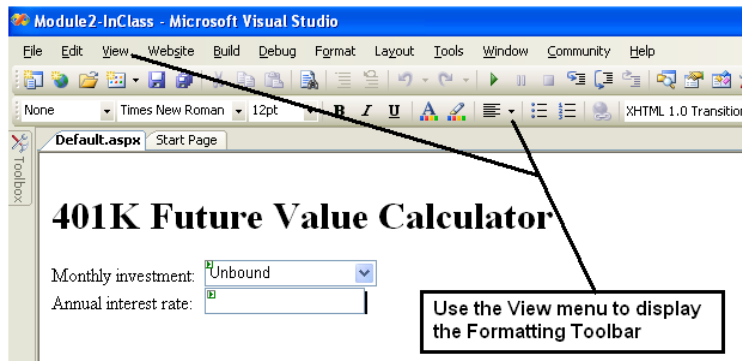
The default design layout for a web form is flow layout. This means the text and controls on the form are positioned from left to right and top to bottom and the actual position of controls on the form can change when it displays in a web browser – positioning of controls is affected by the browser window size and display monitor resolution.

You can create a web page by simply typing information onto the page by using spaces and line breaks – the layout will flow naturally; however, you will have almost no control over the positioning of controls and information on the page.

In this Figure information is typed onto the page and two controls are added from the Toolbox – a DropDownList and a TextBox.

The Formatting Toolbar is displayed by selecting the View menu, Toolbars option.

The text "401K Future Value Calculator" is formatted to display bold and 16 pt font from the Formatting Menu.



The source code (click the Source button at the bottom of the design window) generated for this page looks like this:

```
<% @ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="_Default" %>
```

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<html xmlns="http://www.w3.org/1999/xhtml" >
```

```
<head runat="server">
```

```
  <title>Untitled Page</title>
```

```
</head>
```

```
<body>
```

```
  <form id="form1" runat="server">
```

```
    <div>
```

```
      <br />
```

```
      <strong><span style="font-size: 24pt">401K Future Value
```

```
Calculator</span></strong><br />
```

```
      <br />
```

```
      Monthly investment:&nbsp;
```

```
      <asp:DropDownList ID="DropDownList1" runat="server" Width="144px">
```

```
      </asp:DropDownList><br />
```

```
      Annual interest rate: &nbsp; &nbsp;  <asp:TextBox ID="TextBox1" runat="server"
```

```
Width="128px"></asp:TextBox></div>
```

```
    </form>
```

```
</body>
```

```
</html>
```

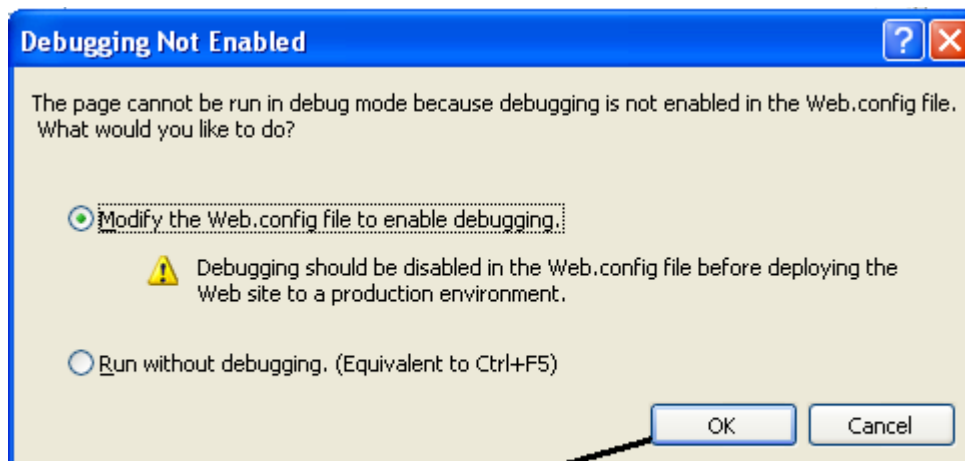
Different tags accomplish different tasks:

- `html` and `/html` mark the beginning and end of the HTML code.
- `asp` and `/asp` mark the beginning and end of the definition of a web server control such as the `DropDownList` control with `ID = DropDownList1`.

If you elect to modify the HTML code, the result will display in the Design view. Example: Change the Title tag from Untitled Page to assign a title such as CMIS460 - Future Value Calculator. At runtime you'll notice that the title reflects your change.

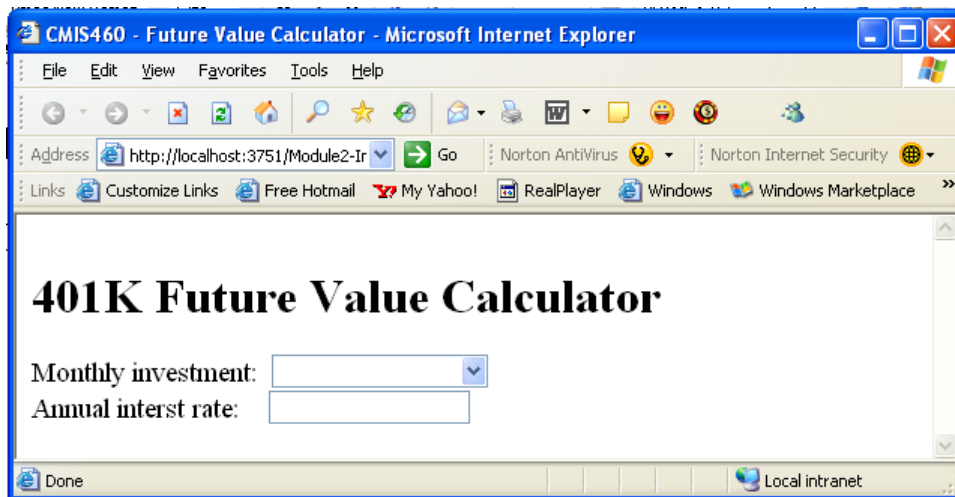
Test the project by clicking the Debug menu, Start Debugging option (or the green arrow used to run VB projects).

The first time you run this project, you'll receive a Debugging Not Enabled popup message. Select either option at this point and click OK.



Select either option and click OK.

The project runs inside a web browser. Here the browser option is Internet Explorer (the client computer's default browser). Note the title reflects the change made to the HTML source code.



Using Tables for Form Design

Tables can be used to aid in formatting a form in flow layout. To insert a table:

- Position the cursor on the web site form where you want the table to be inserted.
- Select Layout menu, Insert Table option → this displays the Insert Table dialog box.
- Set the number of rows and columns and other desired options.
- Format the table by:
 - Resizing rows – drag rows by the bottom border and columns by the right border.
 - Use the Layout menu or shortcut menu to add, delete, resize, and merge rows and columns.
 - Add text by typing into the cells – use the Formatting toolbar to format the text.
 - Align controls by adding them to the cells.

Add Server Controls and Set Control Properties

There are two ways to add controls from the toolbox:

- Drag/drop the control to the web site design surface.
- Position the cursor on the form where you want the control, then double-click the control in the Toolbox.

Setting properties is like a Windows VB project.

- Select the control, and then use the **Properties** window to set property values.
- Set **Height** and **Width** properties of controls like a TextBox by dragging the control handles.
- Select multiple controls (hold **Ctrl** or **Shift** and click the controls one at a time or use the **Lasso** method) at the same time to set common properties.
- Use the **Format menu** to set control properties such as size.
- Use the +/- expansion buttons to expand/collapse a property list such as Font.

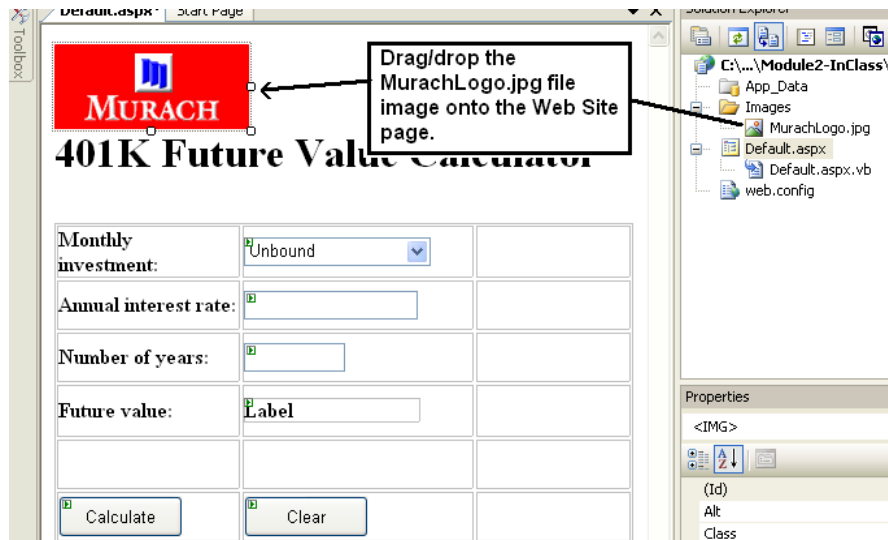
- Use the control's **smart tag menu** (click the **smart tag arrow** in the control's upper right corner) to perform common tasks and set common properties.
- Add a **HTML** image control by dragging it from the Solution Explorer onto the form.

This table describes common web server control properties.

Common Web Server Control Properties	
Property	Description
AutoPostBack	Causes page to post back to the server when the control value changes – default = False (no post back)
CausesValidation	Determines if validation is done by a validation control when you click on a button, link button, or image button – default = True (validate).
EnableViewState	Determines if the control maintains its view state (value) between HTTP requests – default = True .
Enabled	Control is enabled or not – default = True .
Height	Height of control in pixels.
ID	Name of control (use instead of the Name property of a Windows form).
Runat	If the control will be processed on the server by ASP.NET, e.g., runat the server.
TabIndex	Order of control tabbing.
Text	Text that displays on a control.
ToolTip	Text to display when the mouse hovers over the control.
Visible	If the control is displayed or hidden – default = True .
Width	Width of control in pixels.
Common DropDownList and ListBox Control Properties	
Items	Collection of ListItem objects in the control. Can insert at Design Time or by using code to add, insert, and remove items in a List or ListBox
SelectedItem	The ListItem object for the currently selected item.
SelectedIndex	The index of the currently selected item – default = -1 when no ListItem is selected – item index values are numbered 0, 1, 2, 3, etc.
SelectedValue	Value of the currently selected item.

The AutoPostBack property does NOT apply to button controls – they always post back to the server.

Images can be added by dragging/dropping them from the Solution Explorer window onto the design view surface as shown in this Figure. Here the form is modified to add a table with six rows and two columns. The Murach logo image has been added to the form.



The controls are named as follows:

- Monthly investment DropDownList box: ddlMonthlyInvestment
- Annual interest rate TextBox: txtInterestRate
- Number of years TextBox: txtYears
- Future value Label: lblFutureValue (set the Text property = blank).
- Calculate button: btnCalculate
- Clear button: btnClear

Using Source View

- Source view displays the code that represents the visual rendering of the form that you see in Design view.
- Source code includes HTML and ASP tags.
- ASP tags are converted to HTML before the form is sent by the server to a client browser.
- Control properties from Design view appear as ASP tag attributes in Source view.
- Source code files have aspx file name extensions.

In this Source view code extract, the code between the <body> tags is shown. Note the ASP tag for the DropDownList control with the ID attribute = ddlMonthlyInvestment and the Width attribute = 144px.

```

<body>
  <form id="form1" runat="server">
    <div>
      <br />
      <strong><span style="font-size: 24pt">401K Future Value Calculator<br />
        <br />
        <div style="text-align: left">
          <table>
            <tr>
              <td style="width: 160px">
                <span style="font-size: 12pt">
                  Monthly investment:</span></td>
              <td style="width: 191px">
                <asp:DropDownList ID="ddlMonthlyInvestment" runat="server" Width="144px">
                  </asp:DropDownList></td>
              <td style="width: 160px">
                [PART of the code here is deleted ]
                <asp:Button ID="btnCalculate" runat="server" Height="32px" Text="Calculate"
                Width="96px" /></td>
              <td style="width: 191px">
                <asp:Button ID="btnClear" runat="server" Height="32px" Text="Clear" Width="96px"
                /></td>
              <td style="width: 173px">
                </td>
            </tr>
          </table>
          [PART of the code here is deleted ]
        </div>
      </div>
    </form>
  </body>

```

Modifying Image Tags – the image tag is generated with only a Src (source) attribute when an image is drag/dropped on the web site design surface. It is necessary to manually add the Alt attribute to specify text to display if the image is not available or being displayed by the browser.

The modified code looks like this:

```

<br />

```


2-4.4 Validation Controls

There are five validation controls and one summary control. Validation controls are used to validate input data automatically and display error messages when the data fails validation tests.

The approach is to add a Validation control, then attach the Validation control to an input control such as a TextBox by setting the ControlToValidate property = ID of the control to be validated.

Next, set the ErrorMessage property to display a specific message when validation fails.

Validation is performed when a control loses focus (tabs away from the control or mouse clicks another control), and also when an application user clicks a button control where its CausesValidation property = True.

A single server control can have more than one associated validation control.

For uplevel browsers (Internet Explorer 5.5 and above), validation takes place on the client without a post back so no bad data is submitted. For all other browsers (called downlevel browsers), validation takes place on the server.

Validation controls run a client-side script – this makes them efficient because a network trip to the server is not necessary to validate data. Client-side validation requires the browser to support dynamic HTML (DHTML).

Validation is also always done by the server when a page is submitted during a post back. ASP.NET validates by testing the IsValid property of each control, and if all are valid, it sets the page's IsValid property = True. You can test this in an event handler for a Button's control's click event. If validation fails, the page does NOT post back to the server.

This table lists the various Validation controls.

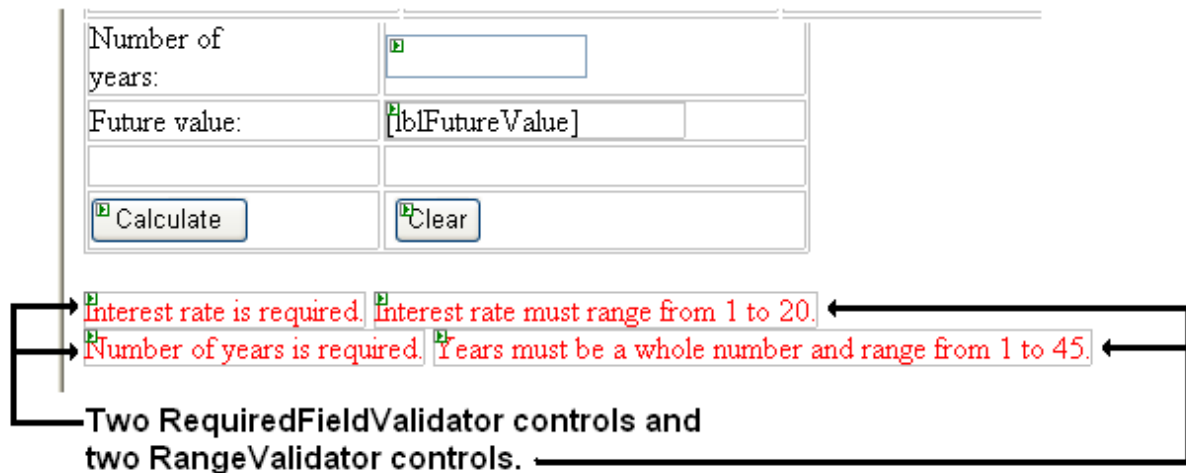
Control	Validation Purpose	Properties To Set
RequiredFieldValidator	Field cannot be left blank	ControlToValidate ErrorMessage Display = Dynamic
CompareValidator	Compares value in the field to a value in another control or to a constant value. You can set the Type property to a specific numeric type and this validator will verify the input can be converted to the correct data type.	ControlToValidate ControlToCompare Or ValueToCompare Type (to force type checking) ErrorMessage
RangeValidator	Ensures the data falls within a specified range.	ControlToValidate MinimumValue MaximumValue Type (type of data to check) ErrorMessage
RegularExpressionValidator	Validates against an expression such as a required number of digits, or formatted value such as a social security number. Use the Regular Expression Editor to select/edit expressions.	ControlToValidate ValidationExpression ErrorMessage
ValidationSummary	Summarizes all messages from other validation controls	DisplayMode (set to bullet list, list, or message box)

Display property – determines how an error message is displayed. Static allocates space for a message in page layout. Dynamic allocates space when an error occurs. None displays errors in a validation summary control.

Type – data type to use for range checking (Integer, Double, Decimal, String).

A blank value in a control such as a textbox will pass each validation check except the RequiredFieldValidator control. To check if a value is not blank AND passes a range check, use both a RangeValidator and RequiredFieldValidator.

This Figure shows two RequiredFieldValidator and two RangeValidator controls added to the web site and positioned just below the Calculate and Clear buttons.



The property settings for these four controls are:

ID = RequiredFieldValidator1

ControlToValidate = txtInterestRate

ErrorMessage = Interest rate is required.

Display = Dynamic

ID = RangeValidator1

ControlToValidate = txtInterestRate

ErrorMessage = Interest rate must range from 1 to 20.

Display = Dynamic.

MaximumValue = 20

MinimumValue = 1

Type = Double

ID = RequiredFieldValidator2

ControlToValidate = txtYears

ErrorMessage = Number of years is required.

Display = Dynamic

ID = RangeValidator2

ControlToValidate = txtYears

ErrorMessage = Years must be whole number and range from 1 to 45.

Display = Dynamic

MaximumValue = 45

MinimumValue = 1

Type = Integer

Add the controls and test the application. Try clicking Calculate without entering the interest rate or years.

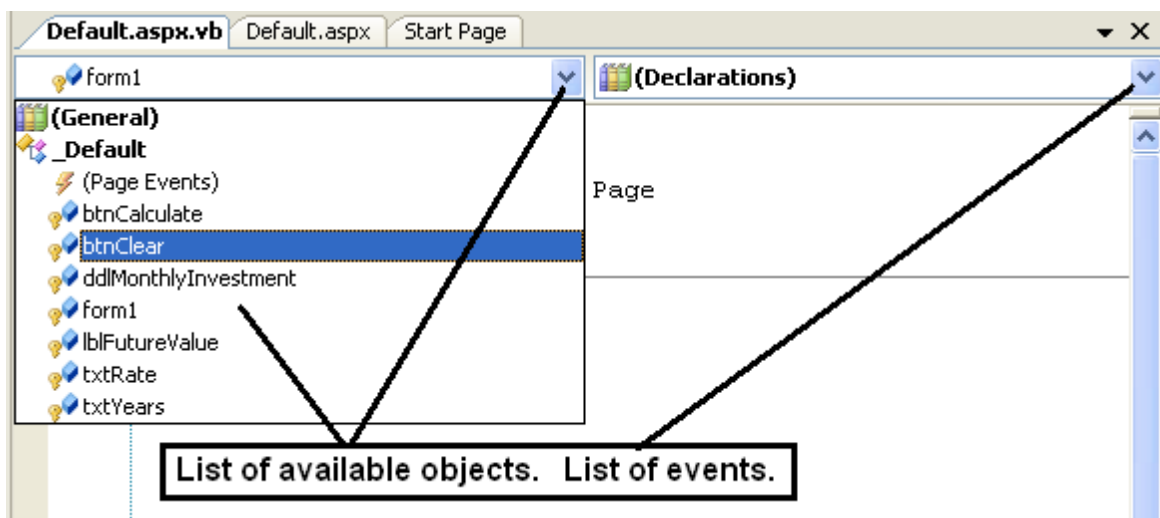
Coding a Form

The Web Forms Code Editor functions exactly like the Windows Forms Code Editor. The Code Editor opens by:

- Double-click the form – this displays the page's Load event.
- Double-click a web site control – this generates a Sub procedure for the control's default event.
- Select the View menu, Code option (or click the F7 shortcut).

The Code Editor window can be opened and the drop down lists at the top of the code editor can be used to generate other event Sub procedures.

- Left drop down list displays a list of available objects.
- Right drop down list displays events associated with the selected object.



This table lists common ASP.NET page and control events.

Common ASP.NET Page Events		
Event	Procedure Name	Fires When
Init	Page_Init	Fires when a page is requested from a server – event occurs before the view state of page controls is restored.
Load	Page_Load	Fires when a page is requested from a server after all controls are initialized and view state is restored.
PreRender	Page_PreRender	Fires when all control events for a page have been processed but prior to generation of the HTML that will be sent back to a browser.
Common ASP.NET Control events		
Click	--	Fires when a button, link button, or image button is clicked.
TextChanged	--	Fires when TextBox value changes.
CheckedChanged	--	Fires when RadioButton (in a group of RadioButtons) is selected or unselected, or when a CheckBox is selected or unselected.
SelectedIndexChanged	--	Fires when an item from a ListBox, DropDownList, CheckBoxList, or RadioButtonList is selected.

PreRender Event – raised after all control events for the page have processed and just before a page is rendered to HTML.

Init Event – raised just before the Load event – used by ASP.NET to restore the view state of an event and its controls – usually you will NOT program this event.

Load Event – raised after the Init event – use this to initialize object values.

Click Event – raised when a button is clicked, then the page posts back to the server, the Init and Load events execute, and then the event handler for the Click event of the control that was clicked.

To make an event handler execute immediately when an event occurs, set the AutoPostBack property of the control = True. The event handler will execute after the Init and Load event handlers for the page.

If `AutoPostBack = False`, the event is raised, but the event handler code does not execute until another action causes the page to post back to the server.

Page_Load Event

Here the `Page_Load` event is used to populate the `DropDownList` named `ddlMonthlyInvestment` with possible investment values. Add this code to your project.

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
```

```
    'Fill DropDownList with values if page is
    'requested for the first time.
    If Not IsPostBack Then
        Dim iIndex As Integer
        For iIndex = 50 To 500 Step 50
            ddlMonthlyInvestment.Items.Add(iIndex)
        Next iIndex
    End If
End Sub
```

The `DropDownList` should only be filled with values the first time the application user requests the page. This is controlled with the `IsPostBack` property of the page.

`IsPostBack = True` – page is being posted back by the user.

`IsPostBack = False` – page is being requested for the first time – this is equivalent to `Not IsPostBack`.

`Protected Sub Procedure` – note the `Protected` access modifier (instead of the `Private` modifier) – a Sub procedure that is protected may contain sensitive data or restricted code. The `Protected` modifier limits access to the element to the base class and all derived classes.

Calculate Button Click Event and FutureValue Function

Before calculating the future value, the `IsValid` property of the page is tested to determine if all of the data values entered by the application user for the page's controls are valid.

The `DropDownList` control's `SelectedValue` property is used to specify the investment amount. `TextBox` values are parsed as appropriate.

The future value of the investment stream is computed by calling a function named `FutureValue`.

```
Protected Sub btnCalculate_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles btnCalculate.Click
```

```
    Dim iMonths As Integer
```

```
    Dim dInterestRate, dMonthlyInvestment As Decimal
```

```
    Dim dFutureValue As Decimal
```

```
    'Check that the data values on the page are valid
```

```
    If Me.IsValid Then
```

```
        iMonths = Integer.Parse(txtYears.Text) * 12
```

```
        dInterestRate = Decimal.Parse(txtInterestRate.Text) / 12 / 100
```

```
        dMonthlyInvestment = Convert.ToDecimal(ddlMonthlyInvestment.SelectedValue)
```

```
        'Call the FutureValue function
```

```
        dFutureValue = FutureValue(iMonths, dInterestRate, dMonthlyInvestment)
```

```
        lblFutureValue.Text = dFutureValue.ToString("C2")
```

```
    End If
```

```
End Sub
```

```
Private Function FutureValue(ByVal Months As Integer, ByVal InterestRate As Decimal,
ByVal MonthlyInvestment As Decimal) As Decimal
```

```
    Dim iIndex As Integer
```

```
    Dim dFutureValue As Decimal
```

```
    'Loop for the number of month periods and
```

```
    'accumulate the future value of the investment
```

```
    For iIndex = 1 To Months
```

```
        dFutureValue = (dFutureValue + MonthlyInvestment) * (1 + InterestRate)
```

```
    Next iIndex
```

```
    Return dFutureValue
```

```
End Function
```

Add the sub procedure and function to your program.

Clear Button Click Event

The Clear button's Click event code is straight-forward as shown here.

```
Protected Sub btnClear_Click(ByVal sender As Object, ByVal e As System.EventArgs)
Handles btnClear.Click
```

```
    ddlMonthlyInvestment.Text = 50
```

```
    txtInterestRate.Text = ""
```

```
    txtYears.Text = ""
```

```
    lblFutureValue.Text = ""
```

```
End Sub
```

Add the sub procedure and function to your program.

Test a Web Site

Visual Studio compiles the application and launches the built-in ASP.NET Development Server – this displays the Web Site page in the default browser.

If errors are detected during compilation, the Error List window displays and lists errors.

To run an application:

- Click the Debug menu, Start Debugging option (or the F5 key, or the green arrow on the Standard toolbar).
- Click OK on the window that asks if you want to modify the Web.ConFigure file to enable debugging.

To stop running an application:

- Click the web browser Close window button.
- If an exception occurs, click the Debug menu, Stop Debugging option (or press Shift+F5).

To fix Build Errors during compilation:

- Double-click on the error listed in the Error List window – this displays the source code that contains the error.
- Run the application again after all errors are fixed.

To view HTML for a page displayed in a browser, select the browser's View menu, Source option. This Figure shows part of the HTML for the web site developed in these notes in Internet Explorer.

View State data is stored in hidden input field(s) within the HTML and is encrypted. Values entered by an application user are returned to the browser as part of the HTML. Validation controls generate script code to perform the validation if the client supports DHTML.


```

Default[1] - Notepad
File Edit Format View Help

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head><title>
    CMIS460 - Future Value Calculator
</title></head>
<body>
    <form name="form1" method="post" action="Default.aspx" onsubmit="ja
WebForm_OnSubmit();" id="form1">
    <div>
    <input type="hidden" name="__EVENTTARGET" id="__EVENTTARGET" value="" />
    <input type="hidden" name="__EVENTARGUMENT" id="__EVENTARGUMENT" value=
    <input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
    value="/wEPDwUKMTk5NTk4NDUzOQ9kFgICAw9kFgQCAQ8QZA8WcmYCAQICAgMCBAIFAgYC
MGcQBQMxMDAFazEwMGcQBQMxNTAFazE1MGcQBQMxMDAFazIwMGcQBQMxNTAFazI1MGcQBQM
FAZM1MGcQBQM0MDAFazQwMGcQBQM0NTAFazQ1MGcQBQM1MDAFazUwMGdkZAIHdw8Wah4EVG
RktRQ9dmo3V0smehJZE3n/d6/tf+4=" />
    </div>
    <script type="text/javascript">
    <!--
    var theForm = document.forms['form1'];
    // -->
    </script>

```

This is the hidden view state values for the Web Site in encrypted format.

```

Default[1] - Notepad
File Edit Format View Help

// -->
</script>
    <div>
    <br />
    <strong><span style="font-size: 24pt">401K Future Value Calculato
    <br />
    <div style="text-align: left">
    <table>
    <tr>
    <td style="width: 160px">
    <span style="font-size: 12pt">
    Monthly investment:</span></td>
    <td style="width: 191px">
    <select name="ddlMonthlyInvestment" id="ddlMonthlyInvestment" sty
    <option selected="selected" value="50">50</option>
    <option value="100">100</option>
    <option value="150">150</option>
    <option value="200">200</option>
    <option value="250">250</option>
    <option value="300">300</option>
    <option value="350">350</option>
    <option value="400">400</option>
    <option value="450">450</option>
    <option value="500">500</option>

```

The selected value for the DropDownList control along with the list of possible values.



ACTIVITY Study Tasks: Build the Future Value web site application and test it.

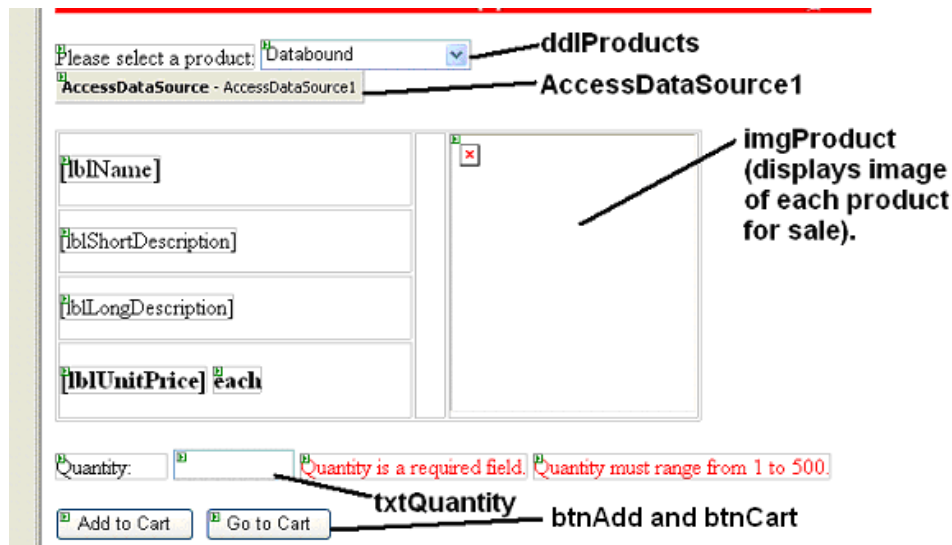
2-4.5 Multi-Page Application

Shopping Cart Application

In this module you will learn to develop part of the Shopping Cart Application. Your work will include developing two web pages, learning to retrieve data from a database, saving data in session state, and using two business classes.

Order Page

The Order Page is shown in design view in this Figure.



Application users order items by selecting a product from the ddlProduct DropDownList control, enter the quantity in the txtQuantity TextBox control, and click the btnAdd Add to Cart Button.

Add to Cart Button:

Selected product is added to the Shopping Cart – the Shopping Cart is a SortedList object that is saved in session state for retrieval/updating when a new product is ordered.

Quantity is updated if the product is already in the Shopping Cart
Cart web page is displayed.

Database:

The application connects to a Microsoft Access database through the AccessDataSource1 control. The Product table in the database has the following columns:

- ProductID
- Name
- ShortDescription
- LongDescription
- CategoryID
- ImageFile
- UnitPrice and
- OnHand.

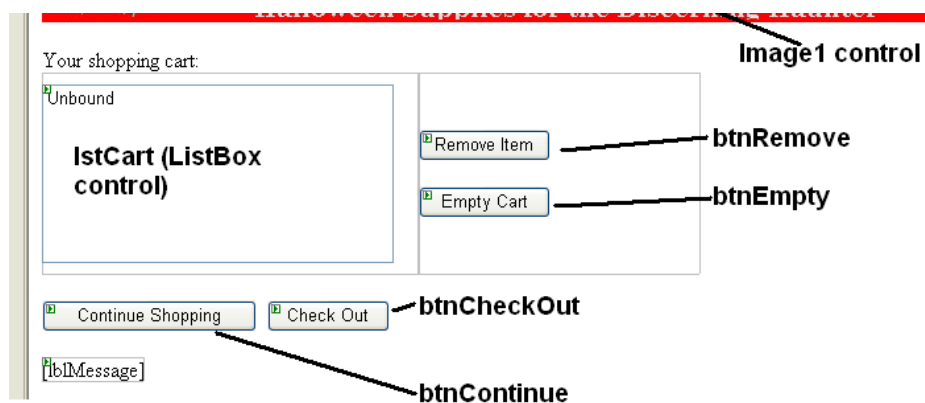
The CategoryID and OnHand columns are not used with this form.

DropDownList (ddlProduct): The AutoPostBack property = True. When a product is selected, data about the product is retrieved from the data source and data is displayed to the labels on the form that are bound to the data source. The product image is displayed in an Image control by setting the ImageUrl property to the value of the ImageFile column of the Product table.

Validation: Data validation is implemented with RequiredFieldValidator and RangeValidator controls.

Cart Page

The Cart Page is shown in design view in this Figure.



The Cart page displays when the application user clicks the Add to Cart button on the Order page.

A ListBox control (lstCart) displays all items in the shopping cart. The list is loaded with cart information saved in session state by the Order page.

Continue Shopping button: Returns to the Order page so additional items can be ordered.

Remove Item button: Application user selects an item in the lstCart ListBox control, and then clicks this button to remove the item.

Empty Cart button: Clicking this empties the lstCart ListBox control.

Check Out button: This simply displays a message stating that the check out function is not yet implemented.

Files and Folders for the Application

The application has several special folders used by ASP.NET to organize application objects. This table summarizes these folders and the files that are in each folder.

Folders Used in ASP.NET Applications		
App_Code	Non-page class files compiled to create a single assembly.	
App_Data	Database files (when the database is part of the project).	
App_Themes	Themes used.	
Bin	Compiled code.	
Files in this Application		
Folder	File	Description
App_Code	CartItem.vb	Class – an item in the shopping cart.
App_Code	Product.vb	Class representing a product.
App_Data	Halloween.mdb	Access database file for the Halloween database.
App_Data	Halloween.ldb	Locking file for the database.
Images	Banner.jpg	Image file for the page banner.
Images\Products	(multiple)	Contains image file for each product.
(root)	Cart.aspx	Cart page aspx file.
(root)	Cart.aspx.vb	Cart page code-behind VB file.
(root)	Order.aspx	Order page aspx file.
(root)	Order.aspx.vb	Order page code-behind VB file.
(root)	web.conFigure	Application conFigureuration file.

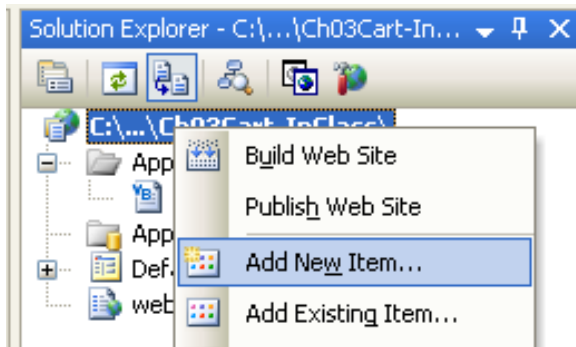
Multiple Web Form Skills and Techniques

This section explains how to perform tasks needed to build a multi-page web site. Later you will begin to build the project.

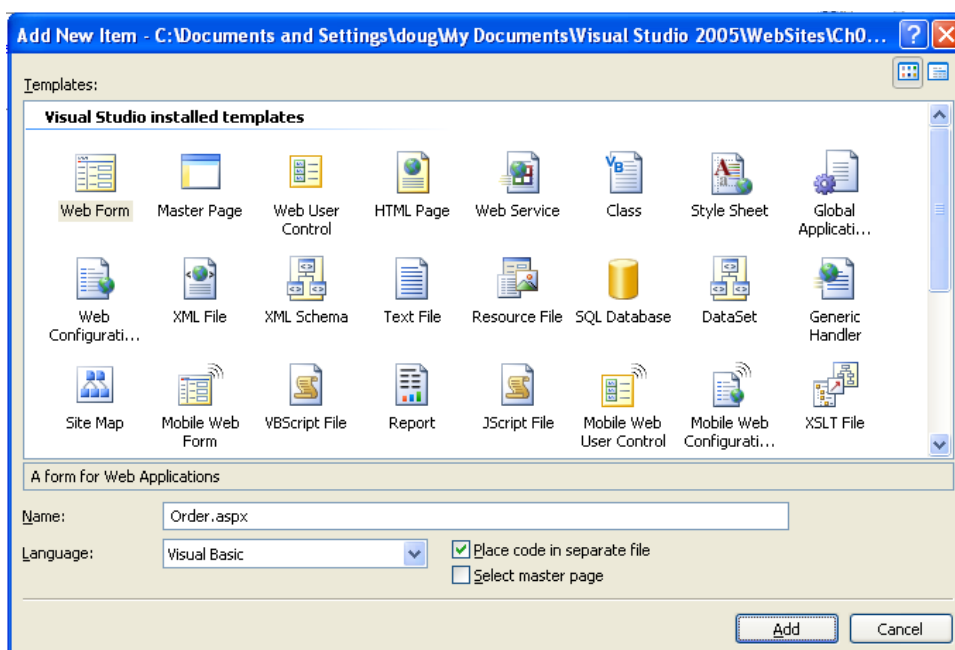
Add a Web Form

Web forms are added through the Add New Item or Add Existing Item dialog box. To add a new web form:

- Right-click the Project in Solution Explorer and choose Add New Item, OR,
- Click on the Project in Solution Explorer to select the Project,
- then choose the Website menu, Add New Item option.



- In the Add New Item dialog box select the Web Form template, name the form, check the Place Code in Separate checkbox, and click Add as shown in this Figure.



To add an existing web form to a project:

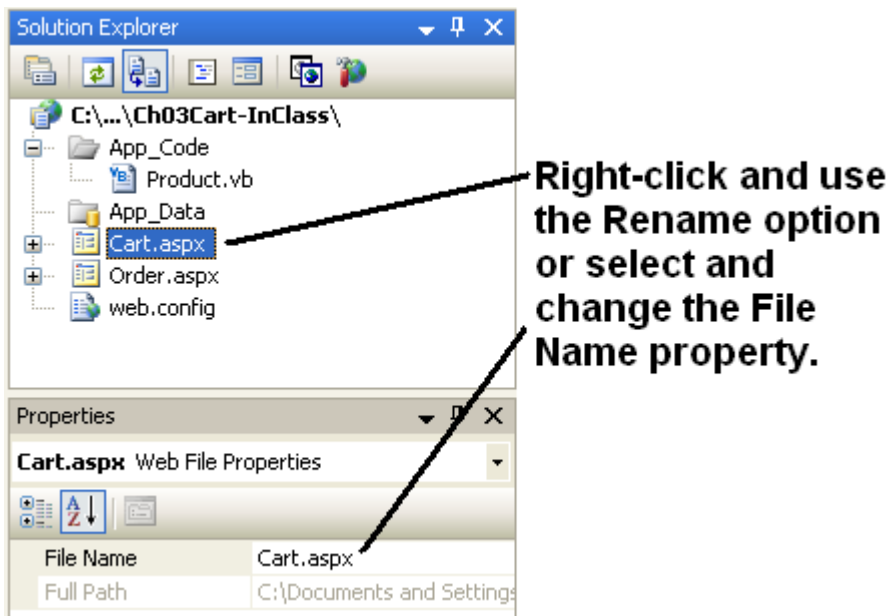
- Right-click the Project Object in Solution Explorer and choose Add Existing Item.
- Browse to the form to be added, select it and click the Add button.

Rename a Web Form

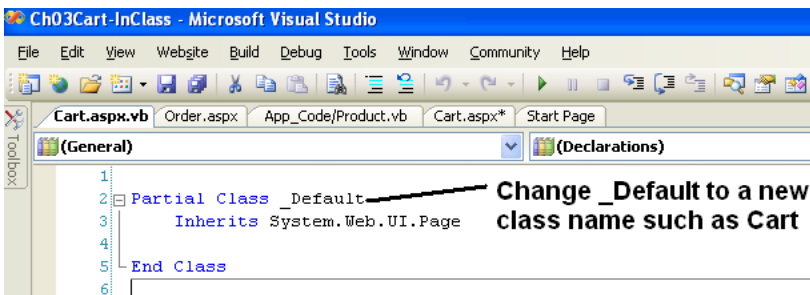
Examine the Solution Explorer – note the web form named Default.aspx. You should rename this form to something that is more descriptive such as Cart because the form will be developed to display the shopping cart for the application.

Renaming a web form requires three steps because Visual Studio does not automatically rename the class in the code-behind file:

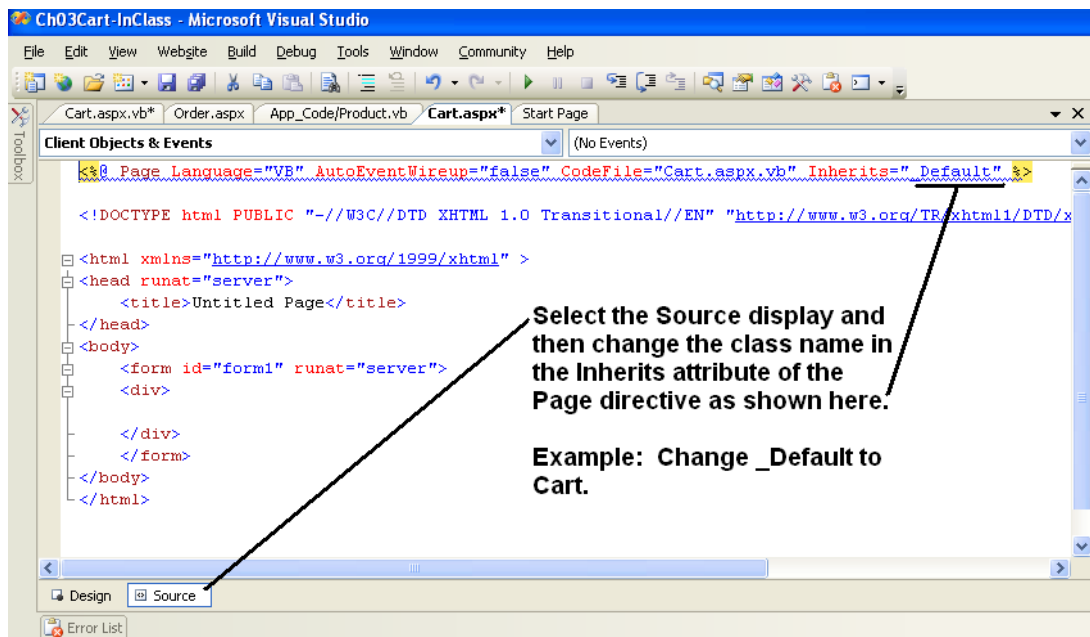
- Rename the form by right-clicking the form in Solution Explorer and using the Rename option, OR by selecting the form in Solution Explorer and changing the File Name property in the Properties window.



- Display the Code Editor window for the form. Change the name on the Class statement (change `_Default` to `Cart`).



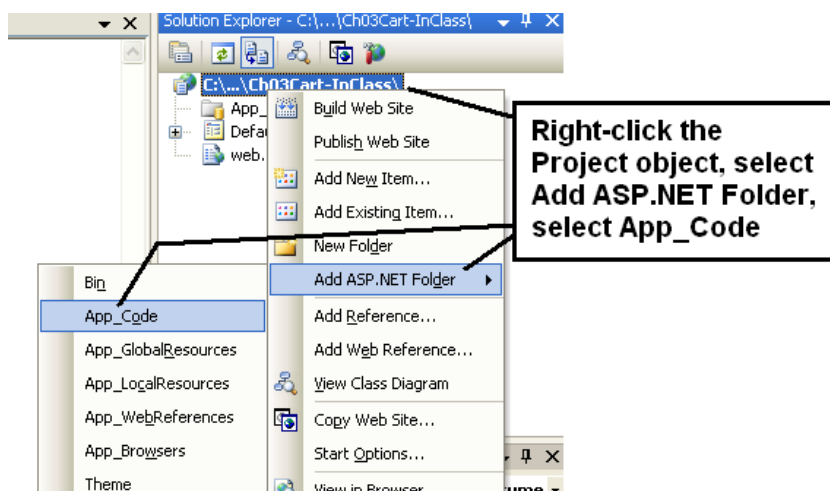
- Select the Source view for the .aspx form code and change the name in the Inherits attribute of the Page directive to the new class name as shown in this Figure (change `_Default` to `Cart`).



ACTIVITY Practice – Start Building the Shopping Cart Application

Start a new file-system web site named Ch03Cart.

Add the App_Code folder. Follow the directions shown in this Figure.



Add a folder named Images as follows:

- Right-click the Project object and select New Folder. Name the folder Images.
- Right-click the Images folder and select Add Existing Item – browse to and add the image file named banner.jpg to the folder (the image file is stored on the class server directory).

- Add a subfolder to the Images folder – name the subfolder Products.
- Right-click the subfolder and select Add Existing Item –
- Add all product jpg image files (there is one file for each product) to the subfolder at one time by holding down the Shift key and selecting all of the image files (all image files are available from the class web site).

Build the Order.aspx Web Form

- Begin by building the Order.aspx web form. In this part of the exercise you will start building the form – you will finish the form later in this module.
- Add a new web form to the project named Order.aspx – right-click the Project and select Add new Item – select a Web Form from the Add New Item dialog box. Check the Solution Explorer and confirm the new web form has been added.
- Add two blank lines to the new form, then position the cursor to the first line and add the banner.jpg image to the form by dragging/dropping the image to the form.
- Add a Label control – Text = Please select a product:
- Add a DropDownList control – ID = ddlProducts; AutoPostBack = True
- Add a HTML table to the form:
- Position the cursor down about two lines (carriage returns) from the ddlProducts control.
- Select Layout menu, Insert Table option to display the Insert Table dialog box. Set the number of rows = 4; number of columns = 3.
- Stretch and merge cells until the table looks approximately like that shown in this Figure.

The screenshot shows the Order.aspx web form in Visual Studio. The form contains the following elements:

- Label1:** "Please select a product:"
- ddlProducts:** A dropdown list with "Unbound" selected.
- Table:** A table with 4 rows and 3 columns. The first column contains labels: "Name", "ShortDescription", "LongDescription", and "UnitPrice". The second column contains labels: "Name", "ShortDescription", "LongDescription", and "Each". The third column is empty.
- imgProduct:** A large image placeholder to the right of the table.
- Label3:** "Quantity:"
- txtQuantity:** A text input field.
- btnAdd:** "Add to Cart" button.
- btnCart:** "Go to Cart" button.

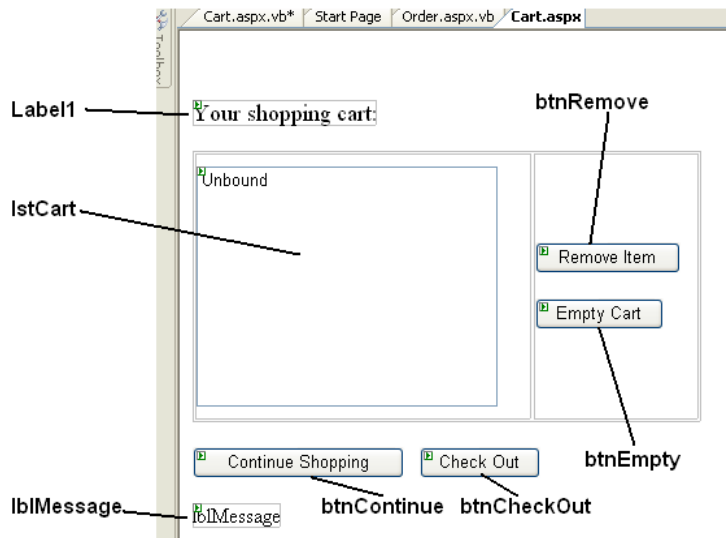
Red error messages are visible below the text input field: "Quantity is a required field" and "Quantity must rang".

- Add labels to display the product's name, short description, long description, and unit price. Set the ID property as follows:
- Name label: ID = lblName
- Short description label: ID = lblShortDescription
- Long description label: ID = lblLongDescription
- Unit price label: ID = lblUnitPrice
- Each label: Text = Each
- Add an Image control to the right-most cell in the table (the rows have been merged): ID = imgProduct
- Add a Label below the HTML table: Text = Quantity:
- Add a TextBox next to the Quantity label: ID = txtQuantity
- Add two Button controls:
- First Button: ID = btnAdd; Text = Add to Cart
- Second Button: ID = btnCart; Text = Go to Cart
- Add a RequiredFieldValidator control:
- ID = RequiredFieldValidator1
- ControlToValidate = txtQuantity
- Display = Dynamic
- ErrorMessage = Quantity is a required field.
- Add a RangeValidator control:
- ID = RangeValidator1
- ControlToValidate = txtQuantity
- Display = Dynamic
- ErrorMessage = Quantity must range from 1 to 500.
- MaximumValue = 500
- MinimumValue = 1
- Type = Integer.
- Switch to the Source view – change the page title tag from Untitled Page to be Shopping Cart – Order Page

Build the Cart.aspx Web Form

This will reinforce your learning regarding renaming a form. You will rename the Default web form to be the Cart.aspx form. This will be the second form for the multiform web site.

- Rename the Default web form to Cart.aspx in the Solution Explorer using the steps described earlier in these notes.
- Add two blank lines (carriage returns) to the form and then position the cursor to the top line and add the banner.jpg image to the form by dragging/dropping the image to the form.
- Add a Label control – Text = Your shopping cart: (use a larger font)
- Add a HTML table with one row and two columns – size the table as shown in this Figure.



- Add a ListBox control inside the first cell of the table – ID = lstCart
- Add two Button controls inside the second cell of the table as shown in the above Figure.
 - First Button: ID = btnRemove; Text = Remove Item
 - Second Button: ID = btnEmpty; Text = Empty Cart
- Add two Button controls below the table as shown in the Figure.
 - Third Button: ID = btnContinue; Text = Continue Shopping
 - Fourth Button: ID = btnCheckOut; Text = Check Out
- Add a Label – ID = lblMessage; Text = (empty)
- Switch to the Source view – change the page title tag from Untitled Page to be Shopping Cart – Cart Page

Set the Starting Web Form

When an application has multiple forms, it is necessary to specify the page that will be the starting form for the web site:

- Right-click the web form in Solution Explorer.
- Choose Set As Start Page from the shortcut menu.



ACTIVITY Practice:

- Set the Order.aspx form as the starting page.
- Redirect/Transfer to another Web Form

To display one web page from another the application user will usually click a button control such as the Add to Cart button on the Order page (to display the Cart page) or the Continue Shopping button on the Cart page (to display the Order page).

Transfer method

This terminates execution of the current page and loads and executes the page specified in the Transfer method.

Member of the `HttpServerUtility` class

A drawback is the browser does not know a new page has been returned so the URL for the original page still displays in the browser's address box which may cause application user confusion.

```
Server.Transfer("Cart.aspx")
```

Redirect method

This sends a special message (an HTTP redirect message) back to the browser which causes the browser to send a new HTTP request to the server to request the new page → the new page is processed and send to the browser.

Member of the `HttpResponse` class

This requires an extra round trip, but is more user-friendly.

Use the Transfer method if the extra round trip causes slow performance for heavily used web sites; otherwise use the Redirect method.

```
Response.Redirect("Cart.aspx")
```

Use Cross-Page Posting

This is a third way to transfer to a new web page.

Specify the page URL in the `PostBackUrl` property of a button control.

When clicked, an HTTP Post message containing the URL (from the `PostBackUrl` property) is sent to the server – this load and executes the page.

Example:

To use cross-page posting for the Go to Cart button of the Order page set its `PostBackUrl` property to `~/Cart.aspx`.

Use this mostly when no data needs to be retrieved from a previous page because retrieving data from a previous page can be done, but is more difficult to program.

Example of aspx code for a Button control that uses cross-page posting:

```
<asp:Button ID="btnCart" runat="server" CausesValidation="False"  
PostBackUrl="~/Cart.aspx" Text="Go to Cart" />
```

If data was entered on the previous page, use the `PreviousPage` property to reference that data.

Example of code that retrieves data from a previous web page – coded in the Page_Load event for the new page. The FindControl method retrieves data from a control with the specified name (txtQuantity in this example) and the CType function casts the control to the specified control class (TextBox).

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles Me.Load
```

```
    If PreviousPage IsNot Nothing Then
```

```
        lblQuantity.Text = CType(PreviousPage.FindControl("txtQuantity"), TextBox).Text
```

```
    End If
```

```
End Sub
```

Code Absolute and Relative URLs

Absolute URLs include the domain name of a web site. When used with a Redirect or Transfer, you can display a page from another web site. Example of absolute URLs:

```
Response.Redirect("http://www.murach.com/Default.aspx")
```

```
Response.Redirect("http://www.murach.com/Books/Search.aspx")
```

A Relative URL can be used to display a new page in the same web site by specifying the page relative to the directory that contains the current page. Example of relative URLs:

```
Response.Redirect("Cart.aspx")
```

```
Response.Redirect("Login/Register.aspx")
```

Relative URLs can also be used to navigate up a directory structure of a web site.

This example navigates up one level in the directory by specifying two periods:

```
Response.Redirect("../Register.aspx")
```

This example navigates up two levels in the directory by specifying two periods, a slash, and two more periods.

```
Response.Redirect("../../Register.aspx")
```

This example navigates to the root directory of the host by coding a slash.

```
Response.Redirect("/Register.aspx")
```

This example codes URLs as property values of server controls. The tilde operator (~) causes the URL to be based on the web site's root directory. Cart.aspx is located in the root directory while banner.jpg is a file in the /Images directory. The tilde is best to use when programming property values because it is easier to maintain a URL relative to the root of a web site than one that is relative to another page.

```
PostBackUrl = "~/Cart.aspx"
ImageUrl = "~/Images/banner.jpg"
```

Add a Class to a Web Site

A class object is defined by adding a new item (a class object) to the project. These class objects do not render web pages and need to be stored in the App_Code folder.

Add New Item

There are two ways to add a new item:

- Right-click the App_Code folder if it already exists and choose Add New Item from the shortcut menu.
- Click on the Project Object in the Solution Explorer to select it, and then choose the Website menu, Add New Item option.

Add New Item dialog box:

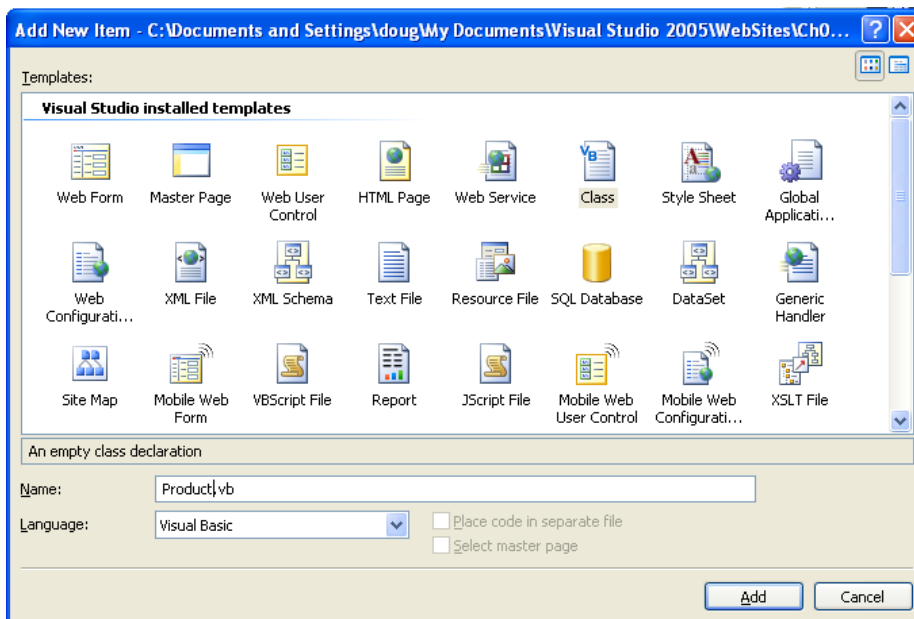
- Select the Class template, name the class (e.g., Product.vb for a Product class), and click the Add button. Visual Studio creates a class with Class and End Class statements.

Visual Studio will warn you if you try to add a class directly to the project and will create the App_Code folder for you.

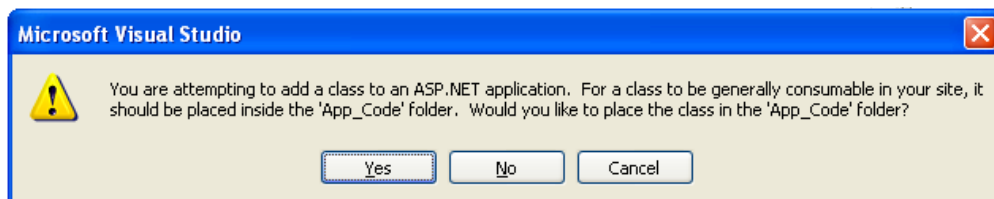


ACTIVITY Practice – Add a Class to the Project

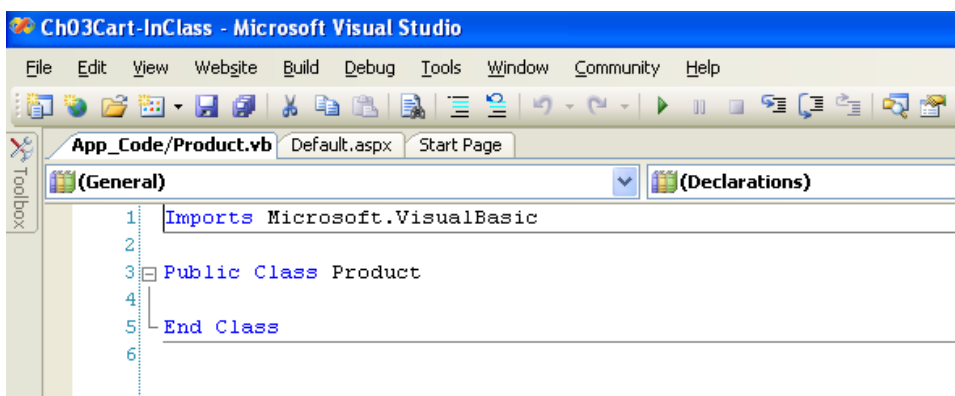
- Select the .aspx file in Solution Explorer – next click the Website menu, Add New Item option.
- Select the Class template and create a class named Product.vb as shown in this Figure.



- Click Yes to add the App_Code folder to the project if you see the warning shown in this Figure.



A class coding page for the Product class is created as shown in this Figure. The Imports statement makes the objects defined in the Microsoft.VisualBasic namespace available to the Class without the need to qualify the object names.



This code shows the definition of the Product class member variables—comments and Member declarations have been added to the code.

Members of a class are similar to module-level variables in a simple Windows VB program. The Public modifier is used to declare the member variables so that their values are available to objects outside of the class such as the Order.aspx form.

```
'Class: Product
```

```
'Defines the Product class members
```

```
Imports Microsoft.VisualBasic
```

```
Public Class Product
```

```
    Public ProductID As String
```

```
    Public Name As String
```

```
    Public ShortDescription As String
```

```
    Public LongDescription As String
```

```
    Public UnitPrice As Decimal
```

```
    Public ImageFile As String
```

```
End Class
```

Add the code to your Product class.

- Add a Class from another web site: If a class object from another web site is to be used in an application, you can add an existing class:
- Right-click the App_Code folder and select Add Existing Item.
- Navigate to the class file to be added, select it, and click the Add button.

Add a class from a class library:

If a class object is part of a class library, you can add a reference to the object:

- Right-click the Project Object in the Solution Explorer window and select Add Reference.
- Click the Browse tab and locate and select the .dll file for the class library – this will add the class library to the project's Bin folder (a Bin folder is created if necessary).

Create and Use Data Sources

The DataSource control is new to ASP.NET 2.0. The control is introduced here, but later modules will cover use of the DataSource in more detail.

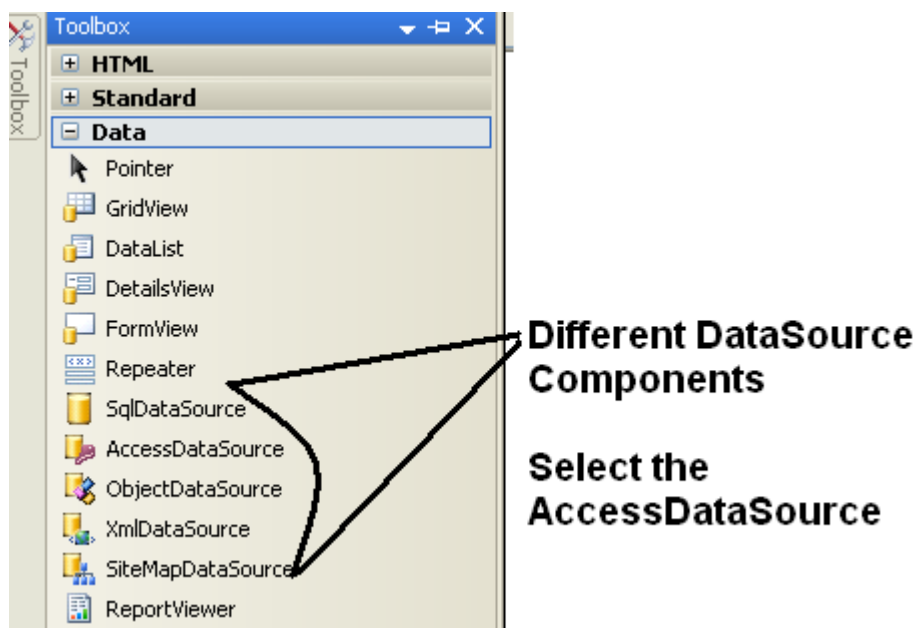
- Add a Microsoft Access Database to a Web Site
- Prior to creating a DataSource for a Microsoft Access database for a web site, you must add the database file to the web site's App_Data folder.
- Right-click the App_Data folder in the Solution Explorer and select Add Existing Item.

- In the Add Existing Item dialog window browse to the Access database location as shown in this Figure, select the database, and click Add. Ensure that you select the file with the .mdb file extension (the .mdf file extension is the SQL Server version of the database).

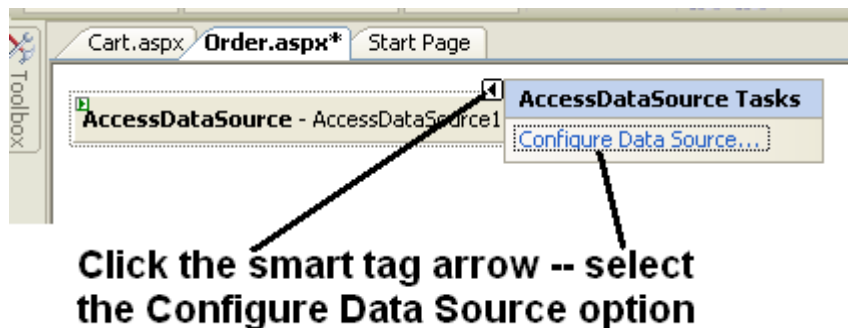
The Solution Explorer will display the database in the App_Data folder as shown in this Figure.

Create and Configure a Microsoft Access Data Source

The Toolbox Data Group contains DataSource tools for different types of database products as shown in this Figure.



- Select the desired web form (we will start with the Order.aspx web form) and drag the AccessDataSource control to the form.
- Click the smart tag arrow and select ConFigure Data Source as shown in this Figure.



The ConFigure Data Source dialog window displays as shown in this Figure. Click the Browse button (or enter the relative URL of the database – it is ~/App_Data/Halloween.mdb) → the Select Microsoft Access Database dialog window displays (also shown in the Figure).

Expand the App_Data folder and then the database. Click OK – note that the relative URL will be entered automatically by this browse operation into the Microsoft Access data file TextBox control on the ConFigure Data Source dialog window.

Click Next on the ConFigure Data Source dialog window.

The wizard moves to the ConFigure the Select Statement option where you will specify how to retrieve data from the database as shown in this Figure.

You will use the SQL (structured query language) SELECT statement to retrieve specific columns from a table (or view) – note that this statement is generated automatically for you for this application.

In the Figure the Products table is selected from the drop-down listing and the columns required are checked.

Additional buttons can be used to specify a more complex SQL SELECT statement.

- The WHERE button allows specifying conditions that limit which records are selected.
- The ORDER BY button allows specifying the sort order of records.
- The ADVANCED button includes INSERT, UPDATE, and DELETES statements.

Click Next – the ConFigure Data Source wizard moves to the Test Query window. Click the Test Query button

If the query succeeds, data from the database's Products table displays in the grid shown.

Click Finish to complete conFigure of the AccessDataSource control.

While the AccessDataSource control is shown in design view on the web page, during program execution the control is hidden (invisible).

Bind a DropDownList Control

A DropDownList control such as the ddlProducts control shown in this Figure will only display data if it is bound to a data source. Binding is accomplished in just a few steps:

- Select the control's smart arrow tag and then the Choose Data Source option. This displays the Data Source ConFigure Wizard.
- Select the DataSource property to the AccessDataSource1 object.

The data field to display in the DropDownList control is the Name column of the Products table – this enables application users to select products by their name. This sets the DropDownList's DataTextField property.

The data field for the value of the DropDownList control is the ProductID column of the Products table because it uniquely identifies each product. This will set the DropDownList's SelectedValue property which can be used to determine the ID of the product to be ordered.

You can also set the DataSourceID, DataTextField, and DataValueField values directly in the Properties window instead of using the wizard.



ACTIVITY Practice – Add a DataSource and Bind a DropDownList Control

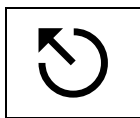
- Practice adding a DataSource control and binding a DropDownList control by completing the tasks described earlier.
- Add a Microsoft Access database to the App_Data folder. Confirm that the database is listed in the Solution Explorer.
- From the Toolbox Data Group add a AccessDataSource component to the Order form.
- Configure the AccessDataSource1 object to link to the Halloween.mdb database file.
- Configure the SELECT statement to select from the Products table. Select the following columns: ProductID, Name, ShortDescription, LongDescription, ImageFile, and UnitPrice. Order the rows by the Name column.
- Test the query.
- Bind the ddlProducts DropDownList control by selecting the control's smart arrow tag and choosing the Choose Data Source option -- DataSourceID = AccessDataSource1; DataTextField = Name (column); DataValueField = ProductID.
- Run the project and confirm that the DropDownList control works.



Self Assessment

1. Inside which HTML element do we put the JavaScript?
 - a. <scripting>
 - b. <javascript>
 - c. <js>
 - d. <script>
2. What is the correct JavaScript syntax to write "Hello World"?
 - a. document.write("Hello World")
 - b. response.write("Hello World")
 - c. ("Hello World")
 - d. request.write(Hello World)
 - e. response ("Hello World")

3. Where is the correct place to insert a JavaScript?
 - a. The <head> section
 - b. Both the <head> section and the section are correct
 - c. The <body> section
 - d. The <Title> Section
 - e. <Script> section
4. What important standard is used to connect client browsers with web servers?
 - a. HTTP
 - b. TCP/IP
 - c. ASP.NET
 - d. HTML
 - e. ASP
5. What is the Web.config file in ASP.NET used for?
 - a. To store the global information and variable definitions for the application
 - b. Configures the time that the server-side codebehind module is called
 - c. To configure the web server
 - d. To configure the web browser
6. What is used to validate complex string patterns like an e-mail address in ASP.Net Application?
 - a. Extended expressions
 - b. Regular expressions
 - c. Irregular expressions
 - d. Basic expressions
 - e. pattern expression
7. What is the file extension used for ASP.NET files?
 - a. asp
 - b. aspx
 - c. web
 - d. Any of the above



Unit Summary

In this unit we have introduced the use java scripting language for validation on the web pages and the Use ASP.Net to develop a web application.



Unit Assignment 4

Develop a web application for a for your future company using ASP.NET

Course Summary

In this course we have learnt the foundation of web-based programming, foundation in web-based technologies, web page creation, web-based architectures – client/server, Web servers Static and dynamic web content and creation – HTML, DHTML, XML, Server side and Client side Scripting and ASP.net programming.

Learner Feedback Form/[insert course code]

Dear Learner,

While studying the units in the course, you may have found certain portions of the text difficult to comprehend. We wish to know your difficulties and suggestions, in order to improve the course. Therefore, we request you to fill out and send the following questionnaire, which pertains to this course. If you find the space provided insufficient, kindly use a separate sheet.

1. How many hours did you need for studying the units?

Unit no.	1	2	3	4	5	6
No. of hours						

2. Please give your reactions to the following items based on your reading of the course

Items	Excellent	Very good	Good	Poor	Give specific examples, if poor
<i>Presentation quality</i>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<i>Language and style</i>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<i>Illustrations used (diagrams, tables, etc.)</i>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<i>Conceptual clarity</i>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<i>Self assessment</i>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<i>Feedback to SA</i>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

3. Any other comments

detach and return to IDL, KNUST

