Web Programming

Client-Side Scripting

A

HTML, CSS and JS Guide

(with JS Applications, e-Commerce Design Using Bootstrap, Theories of Computer Graphics, Photoshop & Flash Animation)

Second Edition

Mamman A. Habeeb

Web Programming

Client-Side Scripting

A

HTML, CSS and JS Guide

(with JS Applications, e-Commerce Design Using Bootstrap, Theories of Computer Graphics, Photoshop & Flash Animation)

Mamman A. Habeeb

Global Publishers/Independent Research Group
Africa Middle East U.K. U.S.A

MH7 Research Group- Reference Manual

Global Publishers/Independent *Research Group*

MH7 Web Programming *Client-Side Scripting A HTML, CSS and JS Guide* (with JS Applications, e-Commerce Design Using Bootstrap, Theories of Computer Graphics, Photoshop & Flash Animation); Reference Manual Published by MH7, a publishing and research group in collaboration with Leanpub, Ruboss Technology Corp.

Library of Congress Catalogue, Nigeria

ISBN 978-978-966-358-3

Habeeb, Mamman A.

MH7 Web Programming *Client-Side Scripting A HTML, CSS and JS Guide* (with JS Applications, e-Commerce Design Using Bootstrap, Theories of Computer Graphics, Photoshop & Flash Animation)***/Mamman A. Habeeb***

Published by **Leanpub, Ruboss Technology Corp**

PREFACE

The book 'MH7 Web Programming *Client-Side Scripting A HTML, CSS and JS Guide* (with JS Applications, e-Commerce Design Using Bootstrap, Theories of Computer Graphics, Photoshop & Flash Animation)*'; i*s a new edition of the Internet Programming for Beginners in *Source Code Writing*, conveys JavaScript Applications and Bootstrap Concepts; and their effects in problem solving and web 3.0 (spiritual Computing) scenarios.

About the Book

This edition covers four chapters and three appendixes. It begins with the basics of web programming and covers significant Web Scripting Languages.

The characteristic chapters on HTML & HTML5, CSS, JavaScript, JS sample Applications and e-commerce with bootstrap app have been covered in an understandable and detailed manner.

The JS sample Applications alone is sufficient for an experience web developer for the purpose of re-use. The book chapters concluded with Theories of Computer Graphics, Photoshop & Flash Animation: where the core importance is on the surface understanding of Computer Graphics in theory and practice.

The book requires Reader's introductory computing skills and elementary programming background.

Web programming and computer programming differ in some aspects, but not in others. Web programming is easier in that web programmers are required to only implement tags and scripts. Of course, there are also differences between the tools and platforms to use. Web programming has a role to play in all areas.

This new edition includes a number of sample web apps to demonstrate the concepts presented in various chapters.

All of the web apps are Client-side Based and the sample programs are scripted in HTML and JS. These development tools will enhance the students' confidence in the subject and enable them to design the sample web apps, either static or dynamic applications in their research areas.

Origin of the Book

This edition evolved from books and research materials prepared by the author for semester courses on "Web Based Information System and Web Technologies," offered to the undergraduate/Postgraduate students in Computer Science and ICT.

Preface

An early version (**Copyright © 2016**) of the book, 'MH7 *Internet* Programming *(HTML, CSS, JavaScript, PHP, E-Commerce Design, ASP.net and Oracle 11g)* for Beginners in Source Code Writing' was also used in a semester course on "Web Technologies and Applications," offered to the undergraduate students in the department of Computer Science/ICT, ESEP-Le Berger Universite, Cotonou-Benin.

Acknowledgments
This new edition had many roots. The author gratefully acknowledges the contributions of many people, who helped him in different ways to complete the book.

All errors that remain, of course, are my responsibility. If you find an error, please send it to me ***habeebmamman@gmail.com***.

May 20, 2018
ESEP-Le Berger University
Cotonou-Benin **Mamman A. Habeeb**

Contents at Glance

CONTENTS

Contents

Contents

Contents

Contents

CHAPTER ONE
INTRODUCING WEB PROGRAMMING

Introduction

A network of networks, joining many government, university and private computers together and providing an infrastructure for the use of E-mail, bulletin boards, search for information over Internet, enjoy Internet surfing , file archives, hypertext documents, databases and other computational resources.

The internet is a *network of networks* that consists of millions of private, public, academic, business, and government networks, of local to global scope, that are linked by a broad array of electronic, wireless and optical networking technologies.
The Internet carries a vast range of information resources and services, such as the interlinked hypertext documents of the World Wide Web (WWW) and the infrastructure to support electronic mail.

History

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

The USSR's launch of Sputnik (artificial satellite-1957) spurred (provoked) the United States to create the Advanced Research Projects Agency (ARPA or DARPA) in February 1958 to regain a technological lead.
ARPA created the Information Processing Technology Office (IPTO) to further the research of the Semi Automatic Ground Environment (SAGE) program, which had networked country-wide radar systems together for the first time. The IPTO's purpose was to find ways to address the US military's concern about survivability of their communications networks, and as a first step interconnect their computers at the Pentagon, Cheyenne Mountain, and Strategic Air Command headquarters (SAC).

J. C. R. Licklider, a promoter of universal networking, was selected to head the IPTO. Licklider moved from the Psycho-Acoustic Laboratory

at Harvard University to (Massachusetts Institute of Technology) MIT in 1950, after becoming interested in information technology.

In 1957 he became a Vice President at (Bolt, Beranek and Newman; an American high tech company in Cambridge) BBN, where he bought the first production (programmed data processor -1) PDP-1 computer and conducted the first public demonstration of time-sharing.
At the IPTO, Licklider's successor Ivan Sutherland in 1965 got Lawrence Roberts to start a project to make a network, and Roberts based the technology on the work of Paul Baran, who had written an exhaustive study for the United States Air Force that recommended packet switching (opposed to circuit switching) to achieve better network robustness and disaster survivability.
Sutherland's successor Robert Taylor convinced Roberts to build on his early packet switching successes and come and be the IPTO Chief Scientist. Once there, Roberts prepared a report called *Resource Sharing Computer Networks* which was approved by Taylor in June 1968 and laid the foundation for the launch of the working ARPANET the following year. In an early sign of future growth, there were already fifteen sites connected to the young ARPANET by the end of 1971.

Intranet

The term “Intranet” is used to describe a network of personal computers (PC) without any personal computers on the network connected to the world outside of the Intranet. The Intranet resides behind a firewall; if it allows access from the Internet, it becomes an Extranet. The firewall helps to control access between the intranet and Internet so that only authorised users will have access to the Intranet. Usually these people are members of the same company or organisation. Like the Internet itself, intranets are used to share information. Secure intranets are now the fastest-growing segment of the Internet because they are much less expensive to build and manage than private network based on proprietary protocols.

Extranet

Extranets are becoming a very popular means for business partners to exchange information. An Extranet is a term used to refer to an intranet that is partially accessible to authorised outsiders. Privacy and security are important issues in extranet use. A firewall is usually provided to help control access between the Intranet and Internet. In this case, the actual server will reside behind a firewall. The level of access can be set to different levels for individuals or groups of outside users.

Internet Access

In order to have access to the vast resources on the Internet, you need to connect your computer to a computer system that is already on the Internet, usually one run by an Internet Service Provider (ISP). There are four major ways of connecting a client (user) computer to the vast resources on the Internet; these are by a dial-up connection using a telephone line or an Integrated Services Digital Network (ISDN), a Digital Subscriber Line (DSL), a cable TV connection or a satellite connection. While rural users may consider installing a satellite dish for Internet connections, urban users may have access to wireless connections. In most offices, users connect their computers via a local area network (LAN) connected to the Internet. Similarly, in many home, users are beginning to connect their computers into Internet-connected LANs, too. The Dial-up access gives a low speed connection to the Internet. High-speed Internet connections, which include DSL, ISDN, leased lines, cable Internet, and satellite, are called broadband connections.

Internet Services and Communication Protocol

The Internet is a global system of interconnected computer networks that use the standard Internet Protocol Suite (TCP/IP) to serve billions of users worldwide. The Internet offers access to data graphics, sound, software, text, to people through a variety of services and tools for communications and data exchange.

Services on the Internet

The Internet carries a vast range of information resources and services, such as the inter-linked hypertext documents of the World Wide Web (WWW) and the infrastructure to support electronic mail.

World Wide Web (WWW)

The World Wide Web is a repository of information spread all over the world and linked together for easy access. It is made up of documents called pages that combine text, pictures, forms, sound, animation and hypertext links into rich communication medium. For several users, The World Wide Web is the most exciting aspect of the Internet, which has accelerated the growth of the Internet by giving it an easy to use, point and click, graphical interface. Users are attracted to the WWW because of it interactive nature. The *WWW* project was initiated by CERN (European Laboratory for Particle Physics) to create a system to handle distributed resources necessary for scientific research as grown today to become many things to millions of users.

It is used as a business place, art gallery, social medium, broadcast medium, library, community centre, school, religious centre, advertise house, publishing house and so on.

WWW information is almost always retrieved using the Hypertext Transfer Protocol (HTTP). In fact HTTP has been in use by the World Wide Web since 1989, and its use has increased steadily over the years. Today there are millions of Websites on the World Wide Web, all of them using HTTP.

The internet consists of two types of computer ***Server* and *Client;***

- Computers which offer information of be read are called **Server.**
- Computers that read the information offered are called **client.**

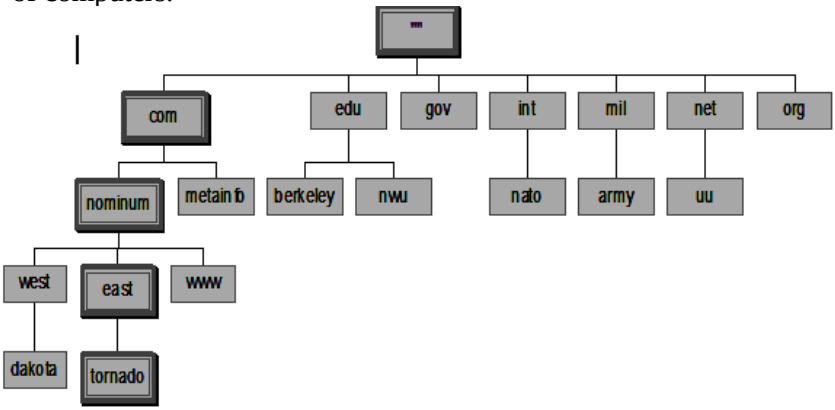
Client-server computing or networking is a distributed application architecture that partitions tasks or workloads between service providers (servers) and service requesters, called clients.

Current Servers

Google- http://www.google.com
Info seek- http://guide.infoseek.com
Alta Vista – http://www.altavista.digital.com
Lycos – http://www.lycos.com
Yahoo! – http://www.yahoo.com

Domain

A domain is logical grouping of computers on a network. It may include multiple networks. It may also just be a subset of a network of computers.



The Top level domains

- com - for commercial entities
- edu - for four-year educational institutions
- gov - for non-military, United States federal government institutions
- mil - for United States military organizations
- net - for network operations and Internet Service Providers (ISP)
- org - for non-profit organizations

Uniform Resource Locator (URL)

Consists of 4 parts:

Protocol → Hyper Text Transfer Protocol (HTTP)
Domain Name → or Internet Protocol (IP) address
Directory
Filename

Specific document filename
index. (s) htm(l) or default.(s)htm(l)

Example: http://www.google.com

Electronic Mail (e-mail)

Electronic mail, commonly called email or e-mail, is a method of exchanging digital messages from an author to one or more recipients. E-mail operates across the Internet or other computer networks. An e-mail message consists of three components namely: (i) the message header (ii) the message envelop, and (iii) the message body.

The message header contains control information, including, minimally, an originator's email address and one or more recipient addresses. Usually descriptive information is also added, such as a subject header field and a message submission date/time stamp. The message body carries the data to be sent. The message's body property usually contains details associated with the message. In addition to the data part, messages carry details that assist in distinguishing messages and selectively receiving them. This detail is made up of a fixed number of fields, which is referred to as the message envelope. These fields are source destination tag communicator. To use email, you should have an email address, which is created by an Internet Service Provider or on a Website such as yahoo, Google, and hotmail. Most e-mail addresses are set up in this manner: your username, followed by "@" (at) symbol, and then a domain name (for instance, .com, .edu.,.net, or .org). When you send e-mail to others, Simple Mail Transfer Protocol (SMTP) is used.

When you receive e-mail, Post Office Protocol (POP, currently POP3) and Internet Message Access Protocol (IMAP) can be used.

Communications Protocols

Protocols are rules that describe how a client and a server communicate with each other over a network. No single protocol makes the Internet and Web work; rather a number of protocols with unique functions are required. The most commonly used protocols are:

- Transmission Control/Internet Protocol (TCP/IP)
- File Transfer Protocol (FTP)
- Hypertext Transfer Protocol (HTTP)
- Email Protocol

Transmission Control Protocol

The Transmission Control Protocol (TCP) is one of the core protocols of the Internet Protocol Suite. It provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer. TCP is the protocol on which major Internet applications such as the World Wide Web, email, remote administration and file transfer rely on. Other applications, which do not require reliable data stream service, may use the User Datagram Protocol (UDP), which provides a datagram service that emphasises reduced latency over reliability.

Internet Protocol

The Internet Protocol (IP) is a set of rules that are more concerned with sending a message to the correct address than with whether the data actually makes it to that receiver. It is therefore, a connectionless protocol, which means that it is an unreliable protocol. IP works by exchanging pieces of information called packets. A packet is a sequence of octets and consists of a header followed by a body. The header describes the packet's destination and, optionally, the routers to use for forwarding until it arrives at its destination. The body contains the data IP is transmitting.

Each device connected to the Internet has a unique numeric IP address. These addresses consist of a set of four groups of numbers, called octet. The current version IP, IPv4 uses 32 bits while IPv6 uses 128 bits. The format of IPv4 is xxx.xxx.xxx.xxx where xxx is a value from 0 to 255. The IP address may correspond to a domain name. The domain name system (DNS) associate these IP address with text-based URLs and domain names you type into a Web browser address box.

It may be easier to type the URL than the IP address. IPv6 is the latest version of the IP routing protocol. It became necessary to introduce a new protocol in order to accommodate the greater demands being placed on the Internet by increasing user and device access.

Domain Name Service (DNS)

An alternative to using the IP address method for locating resources on the Internet is by using the Domain Name Service (DNS) combined with a site's Uniform Resource Locator (URL). URLs are especially formatted names like **www.mammanhabeeb.com** DNS is like a giant phone book where you can find an IP address knowing the URL. On the other hand, you can provide an IP address and the DNS server will like it to the URL.

User Datagram Protocol

An alternative to TCP for communication in the Transport layer is User Datagram Protocol (UDP), UDP is a connectionless protocol (like IP) that operates at the transport layer. It can actually be faster than TCP in some instances because, as a connectionless protocol, it does not have to open a connection with the receiver, and it does not have to do any error correction. Both of these functions are performed by TCP- a connection-oriented, or reliable, protocol – and they take additional overhead in the form of added steps, and they may slow down transmission as a result. However, in cases of large message and faulty connections, errors may occur and retransmission may ultimately make TCP faster than UDP in the long run. UDP does no checks to ensure receipts so it never does automatic retransmission. Missed messages may therefore, result in slower communication over UDP.

Hypertext Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP) is a set of rules for exchanging files such as text, graphics images, sound, video and other multimedia files on the Web. Web browsers and Web Servers usually use this protocol. HTTP is based on the client/server principle. HTTP allows "computer A" (the client) to establish a connection with "computer B" (the server) and make a request. The server accepts the connection initiated by the client and sends back a response. An HTTP request identifies the resources that the client is interested in and tells the server the server what "action" to take on the resources. When the user of a Web browser requests a file by typing a Web site address or clicking a hyperlink, the browser builds an HTTP request and sends it to the server.

The Web server in the destination machine receives the request, does any necessary processing, and responds with the requested file and any associated media files. To retrieve a Web page, the browser sends a request to a Web server using HTTP. On receiving the request, the server interprets it, sometimes using a CGI script (see CGI - Common Gateway Interface), and sends back data. This data can be just about anything, including HTML, text, images, programs, and sound.

E-mail Protocols

Two main servers are required for e-mail messages to be sent and delivered successfully. These are –incoming mail server and an outgoing mail server. Incoming e-mail messages are sent to an e-mail server that stores messages in the recipient's email box. The user retrieves the messages with an e-mail client that uses one of a number of e-mail retrieval protocols. Some clients and servers preferentially use vendor- specific, proprietary protocols, but most support the Internet standard protocols, Simple Mail Transport Protocol (SMTP) for sending e-mail and Post Office Protocol (POP) and Internet Message Access Protocol (IMAP) for retrieving e-mail, allowing interoperability with other servers and clients.

SMTP - Simple Mail Transport Protocol

SMTP controls the transfer of e-mail messages on the Internet. SMTP defines the interaction between Internet hosts that participate in forwarding e-mail from a sender to its destination.

POP - Post Office Protocol

POP allows you to fetch email that is waiting in a mail server mailbox. POP defines a number of operations for how to access and store email on your server.

IMAP - Internet Message Access Protocol

IMAP - Internet Message Access Protocol is an Internet protocol that allows an e-mail client to access email on a remote mail server.

File Transfer Protocol (FTP)

File Transfer Protocol (FTP) is a set of rules that allows files to be exchanged between computers on the Internet. The File Transfer Protocol (FTP) is used widely on the Internet for transferring files to and from a remote host. FTP is commonly used for uploading pages to a Web site and for providing online file archives. Unlike HTTP, which is used by Web browser to request Web pages and their associated files in order to display a Web page, FTP is used simply to move files from one computer to another.

Web developers commonly use FTP to transfer Web page files from their computers to Web servers. FTP is also used to download programs and files from other servers to individual computers. Access to FTP servers can be open or closed. Open access allows anyone to login to the site and download files. This is called anonymous access and it is used frequently for public file archives. Closed access requires that the user provide a username and password to download and upload files. This is the mode of operation for uploading Web pages to a Web site. FTP uses two well-known TCP ports: port 21 is used for the control connection, while port 20 is used for the data connection.

Real Time Streaming Protocol (RTSP)

The Real Time Streaming Protocol (RTSP) is a network control protocol designed for use in entertainment and communications systems such as webcasting to control streaming media servers. Webcasting is the delivery of multimedia data in streaming format across the Internet. Essentially, webcasting is "broadcasting" over the Internet. A webcast can be used to deliver live or on-demand educational and training content or facilitate collaborative applications such as streaming, or chat within an organisation. RTSP is used for establishing and controlling media sessions between end points. Clients of media servers issue Video cassette recorder (VCR)-like commands, such as play and pause, to facilitate real-time control of playback of media files from the server. The transmission of streaming data itself is not a task of the RTSP protocol. To stream data from one location to another simply means that when data is accessed from a source or upon initiation of a data transmission from a source, not all of the data is delivered to the recipient before the data can begin to be viewed at the destination. Steaming utilizes underlying transport and control protocol such as Real-time Transport Protocol (RTP), UDP, and Real-Time Transport Control Protocol (RTCP). RTCP provides out-of-band statistics and control information for an RTP flow. It is similar to the RTP in the delivery and packaging of multimedia data, but does not transport any media streams itself. RTSP is much like HTTP, except that where HTTP will deliver a file from a Web server and then release the connection until the next file is requested, RTSP maintain the connection between a streaming server and the client that is receiving the streamed data.

Network Model

A client may be a program running on the local machine requesting service from a server. A client program is started by the user or another

application program and terminates when the service is complete**.** A server – can sometimes be a program running on the remote machine providing service to the clients. When it starts, it opens the door for incoming request from clients, but it never initiates a service until it is requested to do so.

A network of networks or "Internet" refers to a group of two or more networks that are interconnected and physically capable of communication, share data and act together as a single network. Machine on one network can communicate with machines on other networks, and data, file and other information back and forth. For this to work, the systems must follow some set of rules or protocols. This is a "language" or software that enables different types of machines on separate network to communicate and exchange information. The Internet uses the TCP/IP protocol. The Internet offers access to data, graphics, sound, software, text, and people through a variety of services and tools for communications and data exchange. Some services available on the Internet are as follows:

- Remote login (telnet)
- File transfer (ftp)
- Electronic mail (e-mail)
- News (USENET or network news)
- Hypertext (www)

Computer Network

A network consists of two or more computers connected for the purpose of communicating and sharing resources. There are many types of computer networks, including:

Local-area networks (LANs): This describes the network of computers that are geographically close together (that is, in the same building).

Wide-area networks (WANs): This describes the network of computers that are farther apart and are connected by telephone lines or radio waves.

Campus-area networks (CANs): This describes the network of computers that are within a limited geographic area, such as a university campus or military base.

Metropolitan-area networks MANs): This describes data network designed for a town or city.

Home-area networks (HANs): This describes a network contained within a user's home. Computers on a network are sometimes called nodes.

11 Introducing Web Programming

The common components of a network are:

- Server
- Client workstation computer(s)
- Shared devices such as printers
- Networking devices (hub) and the media that connect them

(For More Information on Networks, See **'MH7 Intelligent Data/Telecommunication Networks'** ***Book)***

WHAT's NEXT?

CHAPTER TWO
HTML

The Basics of HTML and HTML5

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

HTML Tags

HTML Markup tags are usually called HTML tags.
HTML tags are keywords surrounded by angle brackets like <html>.
HTML tags normally come in pairs like and .
The first tag in a pair is the start tag, the second tag is the end tag,
Start and end tags are also called opening tags and closing tags.

HTML Documents = Web Pages

- HTML documents describe web pages
- HTML documents contain HTML tags and plain text
- HTML documents are also called web pages

The purpose of a web browser (like Internet Explorer or Firebox) is to read HTML documents and display them as web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page:

```
<html>
<body>
<h1>My First Heading</h1>
<p>My first paragraph</p>  </body>
</html>
```

The Evolution of HTML

The World-Wide Web Committee (W3C) has developed the following important versions of HTML:

• 1997: **HTML 4** as an SGML-based language,

• 2000: **XHTML 1** as an XML-based clean-up of HTML 4,

• 2014: ***(X)HTML5*** in cooperation (and competition) with the WHAT working group [http://en.wikipedia.org/wiki/WHATWG] supported by browser vendors.

13 Web Programming *Client-Side Scripting*

As the inventor of the Web, Tim Berners-Lee developed a first version of HTML in 1990. In the following years, HTML has been used and gradually extended by a growing community of early WWW adopters. This evolution of HTML, which has led to a messy set of elements and attributes (called "tag soup"), has been mainly controlled by browser vendors and their competition with each other. The development of XHTML in 2000 was an attempt by the W3C to clean up this mess, but it neglected to advance HTML's functionality towards a richer user interface, which was the focus of the WHAT working group led by Ian Hickson who can be considered as the mastermind and main author of HTML5 and many of its accompanying JavaScript APIs that made HTML fit for mobile apps. language. But HTML4 has a lot of purely presentational elements such as font. XHTML has been taking HTML back to its roots, dropping presentational elements and defining a simple and clear syntax, in support of the goals of;

- device independence,
- accessibility, and
- usability.

We adopt the symbolic equation

Requirements

There are absolutely no requirements to start learning HTML, but you will need some tools to help you along the way. There are two tools that are essential to becoming an efficient and professional Web Developer.

Firstly, you will need a Text Editor. Windows users, you can get an awesome text editor from notepad-plusplus.org. As you have probably guessed from the name of the URL, this text editor is Notepad ++ and includes some friendly syntax highlighting! Personally I use Text Notepad.

Mac fans, you can get a text editor from Bare Bones. This handy text editor-Text wrangler also supports syntax highlighting for a range of programming languages.

Linux lovers, you can use the default gnome text editor- gedit. If it’s not already installed, you can get it over here at gnome.org.

Syntax highlighting

Syntax highlighting allows you to easily see certain elements of your code in a designated color. For example, when using a WYSIWYG

(what you see is what you get) editor with syntax highlighting some editors may render the HTML structure as blue, comments as grey and attributes and values as different colors. This allows you to easily distinguish between the sections of code, elements and comments.

Secondly, you will need a browser to render your code. *I recommend Firefox or Internet Explorer*. For the purpose of this Book, go ahead and download and install Mozilla's Firefox (Cross-platform) Browser and the Firebug add-on for Firefox. The Firebug add-on will be your new best friend as a learned web developer.
The Fire-bug Firefox add-on provides the ability to closely "inspect" the elements of an HTML document and see what's going on behind the scenes this will play a big part in your web development career! Alternatively you can use any other browser that supports HTML5-including Safari, Google's Chrome and Opera.

HTML

HTML stands for Hyper Text Mark-up Language. HTML is the basis for all things Web and is a necessary skill for any Web Developer. Almost every website is comprised of HTML whether that is a variation of HTML or plain old HTML.

Structure

A document's structure is established through the use of four tags:

- The `<!DOCTYPE>` tag
- The `<HTML>` container tag
- The `<HEAD>` container tag
- The `<BODY>` container tag

The <!DOCTYPE> Declaration

The stand-alone `<!DOCTYPE>` tag is an optional element to use to declare which level of HTML you're using to author your document. To indicate that you're using HTML 3.2 tags, the first line of your document should be:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
```

This indicates to the browsers that they should use the HTML 3.2 DTD, as specified by the W3C, to parse the document.
You can declare earlier versions of HTML as well. Again, the `<!DOCTYPE>` tag is optional so no browser chokes on a file that doesn't have one.

Basic HTML5 structure:

```
<!DOCTYPE html>
<html>
    <head>
        <title></title>
    </head>
    <body>

    </body>
</html>
```

Instruction:

Type out the above code into a new text file > Save the file with an ***.html*** extension (i.e. structure.html) >Open the html file with your Browser and expect **No Output.**

The browser would not show us any content (output), as we have not yet told the html document to output anything to the browser. All we have told the browser is that we have an HTML document.

Validation

Even though all we have created is a HTML structure and we are not seeing any results (yet), the HTML document we have should validate with W3C's online HTML validation tool. W3C is the World Wide Web Consortium- they set the standards for HTML (and many other Web Based Languages) to provide a similar cross-browser experience; meaning that web browsers will be more inclined to output (or render) data in a similar fashion.

Instruction:

1. Go to W3C's online HTML validation tool,
2. Select the third tab along- "Validate by Direct Input",
3. Copy and paste your HTML5 document code into the window and click
'Check'.
4. Notice how the document passed validation with three warnings.

Let's fix these warnings:

1) The warning- "Using Experimental Feature- HTML5 Conformance Checker" is basically telling us that all major browsers do not officially support HTML5 yet.

2) The next warning- 'No character encoding declared at document level'- this is because we haven't declared our character encoding within the HTML structure.

3) The final warning is telling us that no matter what our character encoding is set to within our HTML document that we are validating, it is going to assume and treat is as (Unicode Transformation Format) UTF-8. The logical way to overcome this last warning is to use the file upload tool, rather than the Direct Input.

Now, let's try and use the W3C validation tool to upload our HTML document, by selecting the second tab on the W3C's online HTML validation tool page and uploading our HTML document.

If you notice we still have our '*Using Experimental Feature- HTML conformance checker*' warning (which is perfectly fine, as we are using HTML5 and it is not yet fully supported by all browsers), the other 2 warnings are related to our character encoding and so is the Error we are now seeing.

Let's fix this, by declaring our Character Set. We will be using the UTF-8 Character encoding and we can do this by adding some simple mark-up to our head section.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>

    </body>
</html>
```

We have just added our first HTML Meta tag!

This Meta tag lets the browser know that we are using UTF-8 character encoding. UTF-8 is the most commonly used character encoding, basically it provides a standard format (encoding) for text (code) that will assist against the problems of endianness, which could result in incorrect or invalid characters displaying (http://en.wikipedia.org/wiki/Endianness).

Tags inside the head element of a HTML document are often used to tell the browser information about the HTML document that we don't need to output as part of our content, such as our HTML title and character encoding.

Comments

In every programming language comments are widely used to help remind other developers what is happening in the code, to make note of extra code that will be added to the web application at a later date and notes to others who may be working on the same project.

In HTML, comments are easily added to the document, by adding the opening and closing comment tags.

```
<! - -
This text will not be rendered by the browser
--!>
```

Our browser will not render anything placed inside the comment tag.

Example of Comments

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title></title>
    </head>
    <body>
        <!--
            This is a comment
        -->
    </body>
</html>
```

HTML5 Structure

Let's go through our HTML document and talk about the structure step-by-step, using comments.

```
<!DOCTYPE html>
<!-- declaring the document type -->
<html>
<!-- opening html tag -->
    <head>
    <!-- opening head tag -->
        <meta charset="UTF-8">
        <!-- declaring our character set -->
        <title>Our Title</title>
        <!-- opening and closing title tags -->
    </head>
    <!-- closing head tag -->
    <body>
    <!-- opening body tag -->
        <!--
```

```
            This is where we put our content
        -->
    </body>
    <!-- closing body tag -->
<!-- closing html tag -->
</html>
```

Nothing is going to be output by the Browser, as we have used comments to explain the structure and have not yet added any "*real"* content.

Instruction:

1. Open up the Fire-bug Firefox add-on. You can do this by right clicking on the Firefox Browser window and selecting "Inspect Element with Fire Bug".
2. Notice how we can see exactly what we have typed into our HTML document on the left-hand side of the Firebug window.

To tell the browser to output some data

It's actually quite simple. But before we add any content to the document, let's talk about the title tag. The title tag allows us to specify the name of the website- more specifically, the web page. It is good practice to be as relevant as you can when giving your web page a title.
As you can see in the previous examples, the title tag is inserted into the head section of the HTML document.

```
<title>Title of the Webpage</title>
```

If we load up this HTML document in our Browser now, we won't see any changes to the webpage. But, have a look at the top of your browser window or current tab- this is where the Title of your HTML document is shown.

Adding Content

We can add our content in-between our body tags like so:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Hello World</title>
    </head>
    <body>
        Hello World in HTML5
    </body>
</html>
```

Test the above script and make sure you alter it in your own way.

HTML Notes:

• All versions of HTML require the basic HTML structure, but the version of HTML depicts a vast difference in the required elements and doc type declarations. HTML4.01, XHTML and HTML5.

• Notice how every tag is closed in order of which they were opened? This is a very important element to valid HTML.

• HTML5 is not currently supported by all major browsers, but provides plenty of extra features for us to work with and stay ahead of the curve. Although all major browsers do not support HTML5, Google's Chrome, Opera and FireFox are currently the most useful tools for Modern Web Development.

• If you are not seeing this "Inspect Element with Fire-bug" option in the dropdown menu, when you right 'click' on your browsers main area- Take a look at this helpful documentation at mozillaZine.

Overview of the HTML Elements

Tables 2.1 through 2.5 provide an overview of all standard HTML elements-both tags and entities. Tables describing tags indicate whether the tag is a container or a stand-alone tag and what the tag's purpose is. Proper tag syntax, including the use of attributes, is discussed over the next several chapters. The entity tables list characters and their associated entities.

***Table** 2.1 HTML Tags Allowable in the Document Head*

Tag	Type	Purpose
<BASE>	Stand-alone	Defines document baseline information
<HEAD>	Container	Denotes the start of the document head
<ISINDEX>	Stand-alone	Indicates that the document is a searchable index
<LINK>	Stand-alone	Establishes linking relationships with other documents
<META>	Stand-alone	Supplies document meta-information
<SCRIPT>	Container	Contains code for a client-side script
<STYLE>	Container	Supplies style sheet information

<TITLE>	Container	Gives the document a descriptive title

***Table** 2.2 HTML Tags Allowable in the Document Body*

Tag	Type	Purpose
<A>	Container	Establishes an anchor
<ADDRESS>	Container	Denotes an address (postal or e-mail)
<APPLET>	Container	Embeds a Java applet in a document
<AREA>	Stand-alone	Defines clickable regions in a client-side image map
<B>	Container	Produces boldface text
<BIG>	Container	Renders text in a larger font size
<BLOCKQUOTE>	Container	Denotes a quoted passage
<BODY>	Container	Denotes the start of the document body
 	Stand-alone	Inserts a line break
<CENTER>	Container	Centers contained items on the page
<CITE>	Container	Indicates the name or title of a cited work
<CODE>	Container	Denotes computer code
<DD>	Container	Denotes a term definition
<DIR>	Container	Initiates a directory listing
<DIV>	Container	Denotes the start of a document division (chapter, appendix, etc.)
<DL>	Container	Initiates a definition list
<DT>	Container	Denotes a term to be defined
<EM>	Container	Signifies text to be emphasized
<FONT>	Container	Modifies font characteristics (size and color)
<H1>	Container	Denotes a level 1 heading
<H2>	Container	Denotes a level 2 heading
<H3>	Container	Denotes a level 3 heading
<H4>	Container	Denotes a level 4 heading
<H5>	Container	Denotes a level 5 heading

<H6>	Container	Denotes a level 6 heading
<HR>	Stand-alone	Places a horizontal line (rule) on a page
<I>	Container	Produces italicized text
<IMG>	Stand-alone	Places an image on a page
<KBD>	Container	Denotes keyboard input
<LI>	Stand-alone	Denotes the start of a list item
<MAP>	Container	Contains definitions of clickable regions for a client-side image map
<MENU>	Container	Initiates a menu list
<OL>	Container	Initiates an ordered (numbered) list
<P>	Container	Denotes the start of a new paragraph
<PRE>	Container	Signifies text to be treated as preformatted
<SAMP>	Container	Denotes sample or literal text
<SMALL>	Container	Renders text in a smaller font
<STRIKE>	Container	Produces strikethrough text
<STRONG>	Container	Denotes text to be strongly emphasized
<SUB>	Container	Renders text as a subscript
<SUP>	Container	Renders text as a superscript
<TT>	Container	Renders text in a fixed-width font (typewriter text)
<UL>	Container	Initiates an unordered (bulleted) list
<VAR>	Container	Denotes a variable name

***Table** 2.3 HTML Tags Allowable in a Form.*

Tag	**Type**	**Purpose**
<FORM>	Container	Denotes the start of a form
<INPUT>	Stand-alone	Specifies a user input field
<OPTION>	Stand-alone	Defines a form menu option
<SELECT>	Container	Contains options in a form menu

<TEXTAREA>	Container	Establishes a window for multiline text input

***Table** 2.4 HTML Tags Allowable in a Table*

Tag	Type	Purpose
<CAPTION>	Container	Denotes a table caption
<TABLE>	Container	Denotes the start of a table
<TD>	Container	Signifies the start of a new table data element
<TH>	Container	Signifies the start of a new table header
<TR>	Container	Signifies the start of a new table row

***Table** 2.5 Reserved Character Entities*

Character	Entity
Ampersand (&)	&amp;
Greater than sign (>)	&gt;
Less than sign (<)	&lt;
Non-breaking space	
Quotation marks (")	&quot;
Copyright symbol (©)	&copy;
Registered symbol (®)	&reg;

Formatting the Document Body

Attributes of the <BODY> Tag

The <BODY> tag is much more than just the element that marks the beginning of the document body. <BODY> takes any or all of the attributes shown in Table 2.6. Note how these attributes allow you to specify many global characteristics of the page including background and text colors.

***Table** 2.6: Attributes of the <BODY> Tag*

Attribute	Function
BACKGROUND	Provides the URL of the image used as the document background
BGCOLOR	Sets the document background color
TEXT	Colors the body text
LINK	Colors unvisited hypertext links

VLINK	Colors visited hypertext links
ALINK	Colors active hypertext links

As you can see from the table, you receive much control over colors.

Specifying Colors in HTML

Computer monitors produce color on the screen by using varying amounts of the primary colors: red, green, and blue. When you specify a color in an HTML document, you need to tell the browser how much of each color to use.

Colors are quantified on computers by using values between 0 and 255.

***Table** 2.7 Allowable English Language Color Names*

Color name	Color name
Aqua	Navy
Black	Olive
Blue	Purple
Fuchsia	Red
Gray	Silver
Green	Teal
Lime	White
Maroon	Yellow

***Table** 2.8: RGB Hexadecimal Triplets for Popular Colors*

Color name	RGB Triplet
Bright Gold	#D9D919
Copper	#B87333
Coral	#FF7F00
Dusty Rose	#856363
Forest Green	#238E23
Khaki	#9F9F5F
Midnight Blue	#2F2F4F
Neon Pink	#FF6EC7
Salmon	#6F4242
Tan	#DB9370

Backgrounds

You have two options when choosing a background for your page. You can set it to a solid color or you can load in an image that tiles to fill the background.

Colors

The BGCOLOR attribute of the <BODY> tag changes the browser's default background color-usually a shade of gray-to whatever color you specify. Set the background color on a Web page to teal with the tag:

```
<BODY BGCOLOR="teal">
```

If you want to use a color that does not have an English language name, you'll have to find out its RGB hexadecimal triplet and set the BGCOLOR attribute accordingly:

```
<BODY BGCOLOR="DF0A82">
```

Images

You can also use the BACKGROUND attribute to read in an image for your document background. BACKGROUND is set equal to the URL of the image file:

```
<BODY BACKGROUND="images/MH7.gif">
```

Colors and Images Together

```
<BODY BGCOLOR="black" BACKGROUND="MH7.jpg">
```

Text Color

The TEXT attribute of the <BODY> tag changes the body text color from its default value (usually black). Like BGCOLOR, TEXT can be set equal to an English language color name or an RGB hexadecimal triplet.

Link Color

Hypertext links come in ***three*** varieties:

- An *unvisited link* is one that the user has yet to click.
- *A visited link* is one that the user has clicked.
- An *active link* is one that the user is clicking at a given moment. Once the user releases the mouse button, the link switches from active to visited.

You can control the color of unvisited, visited, and active links using the LINK, VLINK, and ALINK attributes of the <BODY> tag, respectively. Just as with the other color-related attributes, LINK, VLINK, and ALINK can be set equal to an English language color name or an RGB hexadecimal triplet.

HTML Elements

Paragraph tag

In HTML, the paragraph tag is part of the block level elements group. Block level elements will generally start on a new line and any Mark-up under or after the block level element will also start on a new line.

Here is an example of a paragraph tag in HTML, followed by some text after the ending paragraph tag. Even though all of the text is on one line, the paragraph tag (block level element) will place the text after the closing paragraph tag on a new line.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Block Level Elements</title>
    </head>
    <body>
        <p>I am a paragraph</p>I am directly after the paragraph
    </body>
</html>
```

Output:

I am a paragraph

I am directly after the paragraph

Other Block Level Elements

There are a variety of other block level elements available in HTML; including Headings, logical (or document) divisions, horizontal rules, ordered lists and unordered lists.

So, let's check out some of these block level elements in action.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Block Level Elements</title>
    </head>
    <body>
        <h1>Level 1 Heading</h1>
        <h2>Level 2 Heading</h2>
        <div>This is a logical division</div>
        <hr>
        I will have a horizontal rule above me
        <ol>
            <li>I am part of an ordered list</li>
            <li>li stands for list item</li>
            <li>You can add as many as you want!</li>
        </ol>
        <ul>
            <li>I am still a list item</li>
            <li>This time I am inside an un-ordered list</li>
            <li>Notice how I'm rendered in the browser</li>
        </ul>
    </body>
</html>
```

Output:

Level 1 Heading

Level 2 Heading

This is a logical division

I will have a horizontal rule above me

1. I am part of an ordered list
2. li stands for list item
3. You can add as many as you want!

- I am still a list item
- This time I am inside an un-ordered list
- Notice how I'm rendered in the browser

Declaring Document Divisions

The <DIV> container tag was introduced as part of HTML 3.2 to contain the different logical divisions-such as chapters, appendices, and

bibliographies-within a document. When it was introduced, however, the <DIV> tag was only allowed to take the ALIGN attribute. Just as with the <P> tag, ALIGN can be set to LEFT, CENTER, or RIGHT.
W3C plans to expand the list of attributes that <DIV> takes to make it a more useful tag. The proposed attributes and their functions are shown in Table 2.9.

***Table** 2.9: Proposed Attributes of the <DIV> Tag*

Attribute	**Function**
CLASS	Denotes the type of document division being marked up (chapter, appendix, and so on).
NOWRAP	Turns off auto-wrapping within the division; line breaks are explicitly placed with tags.
CLEAR=LEFT|RIGHT|ALL	Starts the division with empty left, right, or both margin(s).

Lists
HTML supports the following **five** types of lists:

- Unordered or bulleted lists
- Ordered or numbered lists
- Definition lists
- Menu lists
- Directory lists

Changing the Default Bullet Character
The <UL> tag takes the COMPACT attribute which instructs a browser to minimize the spacing between list items. Both <UL> and <LI> take the TYPE attribute. TYPE changes the bullet character that the browser places in front of each list item and sets to DISC for a solid disc, CIRCLE for an open circle, and SQUARE for a square.

Consider the HTML below. It produces the same three-item list, but each with a different bullet character.

```
<UL TYPE="DISC">
<LI>One</LI>
<LI>Two</LI>
<LI>Three</LI>
</UL>
<HR>
<UL TYPE="CIRCLE">
```

```
<LI>One</LI>
<LI>Two</LI>
<LI>Three</LI>
</UL>
<HR>
<UL TYPE="SQUARE">
<LI>One</LI>
<LI>Two</LI>
<LI>Three</LI>
</UL>
```

Changing the Numbering Scheme

Both the <OL> and <LI> tags take the TYPE attribute, giving you control over what numbering scheme to use in your ordered list. TYPE can be set to the values you see in Table 2.10. TYPE="1" is the default setting.

***Table** 2.10: Values of the TYPE Attribute in an Ordered List*

Value	Numbering scheme
TYPE="1"	Counting numbers (1, 2, 3, ...)
TYPE="A"	Uppercase letters (A, B, C, ...)
TYPE="a"	Lowercase letters (a, b, c, ...)
TYPE="I"	Uppercase Roman numerals (I, II, III, ...)
TYPE="I"	Lowercase Roman numerals (I, ii, iii, ...)

You can specify a TYPE for the entire list by placing it in the <OL> tag or for a given list item by placing it in the item's <LI> tag.

Changing the Numbering Sequence

In addition to being able to change the numbering scheme, you can also change the value at which the numbering starts by using the START attribute of the <OL> tag. This is useful in situations where an ordered list is "interrupted" by another element on a page. When you resume the ordered list (by starting with a new <OL> tag), you can set START equal to the appropriate starting value so that it looks like the list is picking up where it left off.

An "interrupted" ordered list gets back on track by using the START attribute of the <OL> tag.

START is always set equal to a number, regardless of the chosen numbering scheme. The browser maps your START value against the numbering scheme it's using and chooses the correct value.

Example: This list could be embedded in another list

```
Other employer-paid benefits include:
      <OL TYPE="A" START=3>
      <LI>Long-term disability</LI>
      <LI>Health club membership</LI>
      <LI>Tuition reimbursement</LI>
      </OL>
```

A browser automatically determines which character to start numbering with, based on the `TYPE` and `START` values you provide.

Changing the value of the numbering sequence in the middle of an ordered list is possible by using the `VALUE` attribute in an `<LI>` tag. An example of one useful application of `VALUE` would be a list of numbers going in descending order. To accomplish this, you'd need a `VALUE` attribute in each `<LI>` like this:

```
      <OL>
      <LI VALUE=3>French hens</LI>
      <LI VALUE=2>Turtle doves</LI>
      <LI VALUE=1>Partridge in a pear tree</LI>
      </OL>
```

Definition Lists

Many documents that are full of technical terms require a glossary so that a user can look up a term if it is not understood. Definition lists make it easy to replicate the term/definition structure found in a glossary.

All terms and definitions in a definition list are found between the `<DL>` and `</DL>` tags. Inside of these tags, you mark up a term with `<DT>` and `</DT>` tags and a definition with `<DD>` and `</DD>` tags. Definitions are indented from the terms above them. The HTML to produce it follows:

```
      <DL>
      <DT>Isosceles triangle</DT>
      <DD>A triangle having two equal sides</DD>
      <DT>Equilateral triangle</DT>
      <DD>A triangle having three equal
      sides</DD>
      <DT>Right triangle</DT>
      <DD>A triangle having one right angle</DD>
      </DL>
```

Definition lists replicate dictionary entries and make it easy to read each term and definition.

Apart from the indenting of definitions, neither terms nor definitions format in any special way. Use the COMPACT attribute in the <DL> tag to decrease the spacing between adjacent terms and definitions.

Menu Lists

Menu lists were originally created for producing menus of short (less than one line) options. Presumably, the menu items would be hypertext links that would take the user to another part of a site.

The options in a menu list are found between the <MENU> and </MENU> container tags. Each list item is again contained between <LI> and </LI> tags. The <MENU> tag takes the COMPACT attribute to reduce inter-item spacing.

Menus look like unordered lists on a browser screen. The different options in a menu list appear with bullets in front of them. The distinction between an unordered list and a menu list is more for the browser than for the end user. In the future, browsers may be programmed to render menu lists in a special format. Additionally, by using style sheets, end users should be able to create their own configurations for the <MENU> tag. For now, though, menu lists look like what you see in the code below. The corresponding HTML is:

```
<MENU>
<LI>What's New!</LI>
<LI>Press Releases</LI>
<LI>Job Opportunities</LI>
<LI>Contact Information</LI>
</MENU>
```

Directory Lists

Directory lists are another type of specialty list without special browser support. Directory lists are intended for lists of short (less than 24 characters) items that are to be displayed in rows. This is like directory listings in UNIX or in DOS with the /W show (a multiple column view of the file names). Like menu lists, however, most browsers simply render a directory list as an unordered list.

The <DIR> container tag creates a directory list. List items are contained by <LI> and </LI> tags. The COMPACT attribute in the <DIR> tag packs the list into a smaller space by reducing the spacing between items. A sample directory list follows:

```
<H1>Employee Directory</H1>
<DIR>
<LI>Lona Dallessandro, x297</LI>
```

```
<LI>Bob Leidich, x324</LI>
<LI>Carolyn McHale, x313</LI>
</DIR>
```

Until they're programmed with special formatting instructions, browsers will continue to display directory lists just like unordered lists.

Quiz

i. Change the content of the code to be about your favourite books.
ii. Add an extra ordered list (containing 7 list items) and a logical division (containing a paragraph element) to the end of the page.
iii. Save the html document as “block_level_elements.html”.
iv. Open the document with your browser and make sure it appears as intended.
v. Upload the html document to the online validator and correct any warnings using the skills you have learned thus far.

Line Breaks vs. Paragraphs

The break element or tag in HTML (as you can guess) provides a line break (or new line). Some people may like to 'over-use' this element, but I suggest using the paragraph element when dealing with text (where possible), to provide formatting and appropriate spacing.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>The Break Tag</title>
    </head>
    <body>
        This text is on one line
        <br>
        This text is on another line
    </body>
</html>
```

We could do the same thing with the paragraph tag, but with a better format.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Block Level Elements</title>
    </head>
    <body>
        <p>This text is on one line</p>
        <p>This text is another line</p>
    </body>
</html>
```

Inline Elements

Now we have an understanding of what block level elements are, it's time to move on to some inline elements.

Text Modifiers- Introducing the strong and em tags.

As you may have guessed, the strong tag is used to define important text and will render text as bold.

The em tag is a little harder to guess. The em tag renders text as *Italic* and is used to *'emphasize'* text. The Strong and *em* tags are both part of the Text Modifiers group.

Example*:*

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Text Modifiers</title>
    </head>
    <body>
        The <strong>strong tag</strong> will render bold text.
        <br>
        The <em>em tag</em> will render italic text.
    </body>
</html>
```

Text modifiers can be a simple way to make certain text stand out or add character to a document. Just *like* **this**!

Font Size and Color

The `<FONT>` container tag was adopted as part of HTML 3.2 standard to give authors control over the size and color of individual characters.

The SIZE attribute of the <FONT> tag modifies the font size in use and the COLOR attribute controls the font color.

SIZE can be used in one of two ways. You can set SIZE equal to a value between 1 and 7, where 1 is the smallest size. The default font size is 3, so changing to a size less than 3 makes text smaller and changing to a size greater than 3 makes text larger. The other way you can use SIZE is to set it equal to the amount of change relative to the current font size. To change to a size two sizes smaller than the current size, for example, use:

```
<FONT SIZE=-2>smaller text</FONT>
```

Similarly, to change to a size one size bigger than the current size, use:

```
<FONT SIZE=+1>bigger text</FONT>
```

A popular effect is to create "small caps" with the SIZE attribute. To make the first letter of a word bigger than the rest, you can use the following HTML:

```
<FONT SIZE=+2>N</FONT>ETSCAPE
```

The COLOR attribute changes the color of the text contained by the <FONT> and </FONT> tags from the default text color, or the text color specified in the TEXT attribute of the <BODY> tag. COLOR can be set equal to an English language color name or a hexadecimal triplet.

This is useful in drawing attention to a particular word in a sentence:

```
<FONT COLOR="red">WARNING!</FONT> The document you have selected is not secure.
```

Images

Images are an important part of any form of content, especially websites. As a web developer, you will find it very helpful and necessary to be able to place images onto a web page.

Graphic Storage Formats (*Pls See Appendix C*)

Technically, Web graphics can be stored in any format, but only two formats display inline through today's popular graphical browsers: GIF and JPEG. Other graphics formats have to be displayed by a helper application, lauched by the browser when it detects a format it can't display. Some browser supports the inline display of Windows Bitmap (.BMP).

GIF

GIF (Graphics Interchange Format) was originally developed for users of CompuServe as a standard for storing image files. The GIF standards have undergone a couple of revisions since their inception.

Graphics stored in GIF are limited to 256 colors. Because full-color photos require many more colors to look sharp, you shouldn't store full-color photos as GIFs. GIF is best used with line art, logos, and icons. If you do store a full-color photo as a GIF, it reduces to just 256 colors and will not look as good on your Web page.

JPEG

JPEG (an acronym for "Joint Picture Experts Group") refers to a set of formats that support full-color images and stores them in a compressed form. JPEG is a 24-bit storage format that allows for 224 or 16,777,216 colors! With that much color data, it's easy to see why some form of compression is necessary.

While JPEG is great for full-color images, it does not permit some of the nice effects that GIF does. Transparency is not possible with JPEG images because the compression tends to make small changes to the image data. If a pixel, originally colored with the transparent color, is given another color, or if a non-transparent pixel is assigned the transparency color, the on-screen results are disastrous.

IMG Tag

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Images</title>
    </head>
    <body>
        <img src="" alt="">
    </body>
</html>
```

Once you have an image stored and ready to be posted on the Web, you need to use the HTML <IMG> tag to place the image on a page. <IMG> is a stand-alone tag that takes the attributes shown in ***Table*** 2.11. According to the HTML, only SRC is mandatory, but you'll quickly find yourself wanting to use many of them.

Table *2.11 Attributes of the <IMG> Tag*

Attribute	Purpose
ALT	Supplies a text-based alternative for the image
ALIGN	Controls alignment of text following the image

BORDER	Specifies the size of the border to place around the image
HEIGHT	Specifies the height of the image in pixels
HSPACE	Controls the amount of white space to the left and right of the image
ISMAP	Denotes an image to be used as part of a server-side imagemap
SRC	Specifies the URL of the file where the image is stored
USEMAP	Specifies a client-side imagemap to use with the image
VSPACE	Controls the amount of white space above and below the image
WIDTH	Specifies the width of the image in pixels

Your basic <IMG> tag then should look like:

```
<IMG SRC="URL_of_image_file" WIDTH=width_in_pixels
HEIGHT=height_in_pixels
ALT="alternative_text_description">
```

```
<IMG SRC="whatever.gif" WIDTH=116 HEIGHT=80
ALT="Reduced image">
```

We would then put the URL for our image inside the src (source) attribute. The URL could be relative or absolute.

*Here is an example of using an absolute URL with the src attribute of the **img** tag:*

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Images</title>
    </head>
    <body>
        <img src="http://glorified.me/image.png" alt="">
    </body>
</html>
```

When working in a Live or Development environment it is good Practice to use relative file Paths, rather than absolute or full file Paths.

- A relative file Path can be defined as being a localised link; using the current directory Structure as a means of navigation between files.
- An absolute file Path is a direct link or URL to a file.

If we had an image titled 'logo.png' in the same folder or directory as our Current html file, we could simply link to that file just by using the files name:

```
<img src="logo.png" alt="">
```

If our image or file were in a directory titled "images" inside our Current folder or directory we would then link to The Image using

```
<img src="images/logo.png" alt="">
```

Sometimes we need to navigate downwards (as opposed to upwards) in our directory Structure. If our directory Structure looked something like:

```
/home/html/Public/Current/
```

And our Current html document is in our "current" folder; we could link to our Image (which Could be located at /home/html/Public/Images/) by using:

```
<img src="../images/logo.png" alt="">
```

../*images/* basically tells the browser to navigate one directory down and then into our Images directory.

Alternate Text

When an image is unable to be displayed by a browser we need a fallback method.
So the alt (alternate text) can be used as our fallback method- meaning we will have some descriptive text to display if the image itself is unable to be displayed for any reason.

An example of an image not displaying could be a HTML email (Gmail will, by default hide any images and ask the user if they want to show images) or the results in a search engine. Search Engines cannot "read" images, so they can only render "alternate" text in Search Engine Result Pages (SERPs).
It is good to be descriptive and short with the alt attribute, like so:

```
<img src="../images/logo.png" alt="HTML5 Logo">
```

Displaying an Image

So let's try out our image tag with a real image!

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Images</title>
    </head>
    <body>
        <img src="html5-logo.png" alt="HTML5 Logo">
    </body>
</html>
```

***Output**:*

Specifying the Size of an Image

We can specify both height and width attributes inside our image tag like so:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Images</title>
    </head>
    <body>
        <img src="html5-logo.png" width="100" height="100" alt="HTML5 Logo">
    </body>
</html>
```

***Note**: The height and width values are in px (pixels).*
So, let's try out this image with the height and width attributes specified!

Border

The BORDER attribute gives you a simple way to instruct the browser to place a border around an image. BORDER is set equal to the number of pixels wide you want the border to be. *Borders create a framed look around photographs.*

Space around Your Image

White space around an image is called *gutter space* or *runaround*. Putting a little extra space around an image is a good way to give it some more breathing room on the page and make it stand out better.
Run around is controlled by the HSPACE and VSPACE attributes. Each is set to the number of pixels of extra space to leave to the right and left of an image (HSPACE) or above and below an image (VSPACE).

ALIGN Attribute and Floating Images

The ALIGN attribute of the <IMG> tag can take on one of the five different values summarized in Table 2.12. TOP, MIDDLE, and BOTTOM refer to how text should be aligned following the image. LEFT and RIGHT create floating images in either the left or right margins.

***Table** 2.12 Values of the ALIGN Attribute in the <IMG> Tag*

Value	Purpose
`TOP`	Aligns the top of subsequent text with the top of the image
`MIDDLE`	Aligns the baseline of subsequent text with the middle of the image
`BOTTOM`	Aligns the baseline of subsequent text with the bottom of the image
`LEFT`	Floats the image in the left margin and allows text to wrap around the right side of the image
`RIGHT`	Floats the image in the right margin and allows text to wrap around the left side of the image

Top, Middle, and Bottom Alignment

One important thing to note with TOP and MIDDLE alignments is that once the text reaches a point where it needs to break, it breaks at a point below the image and leaves some white space between the lines of text.
The <IMG> tags to place the floating images have ALIGN=LEFT or ALIGN=RIGHT attributes.

Images as Hyperlink Anchors

In the next section you will learn how to use the <A> container tag to create hypertext anchors. By clicking the hypertext, you instruct your browser to load the resource at the URL specified in the HREF attribute of the <A> tag.
There's no law that says that hyperlink anchors can only be text. Very often you'll find images serving as anchors as well. By linking images

to other Web pages, you create a button-like effect-the user clicks the button and the browser loads a new page.
To use a graphic as a hyperlink anchor, put the <IMG> tag that places the graphic between <A> and </A> tags:

```
<A HREF="library.html"> <IMG SRC="images/books.gif"
WIDTH=50 HEIGHT=50 ALT="Library"> </A>

<A HREF="index.html">
<IMG SRC="mh7.gif" WIDTH=422 HEIGHT=284 ALT="Photo of
MH7"></A>
```

Images as Bullet Characters
Some people opt to create their own bullet characters for bulleted lists rather than using the characters that browsers provide. To do this, you need to place the bullet graphic with an <IMG> tag and follow it with a list item:

```
<IMG SRC="bullet.gif" WIDTH=12 HEIGHT=12 ALT="*">List
item 1<BR>
<IMG SRC="bullet.gif" WIDTH=12 HEIGHT=12 ALT="*">List
item 2<BR>
<IMG SRC="bullet.gif" WIDTH=12 HEIGHT=12 ALT="*">List
item 3<BR>
```

Image Maps
With a graphical browser, you've probably noticed that many major Websites have a large clickable image on their main page. These images are different from your run-of-the-mill hyperlinked graphic in that your browser loads a different document, depending on where you click. The image is somehow "multiply linked" and can take you to a number of different places. Such a multiply linked image is called an ***Image map***.
The challenge in preparing an image map is defining which parts of the image are linked to which URLs. Linked regions in an image map are called *hot regions* and each hot region is associated with the URL of the document that is to be loaded when the hot region is clicked. Once you decide the hot regions and their associated URLs, you need to determine whether the Web server or the Web client will make the "decision" about which document to load, based on the user's click. This choice is the difference between *server-side image maps* and *client-side image maps*. Either approach is easy to implement once you know the information needed to define the hot regions. ***Readers that interested in Image maps should mail the author.***

Hyperlinks

Anchor Tags/ Hyperlinks

Now, let's move on to the ever-so important anchor tags.
We use an anchor tags like so:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Anchor Tags</title>
    </head>
    <body>
        <a href="http://glorified.me">This is a Hyperlink</a>
    </body>
</html>
```

The above code will render as:

This is a Hyperlink

Let's try a simple link to Google:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Anchor Tags</title>
    </head>
    <body>
        <a href="http://google.com">Go to Google?</a>
    </body>
</html>
```

The above example will render:

Go to Google?

Type out the above code and Try it out, notice that the browser will now load

Google's homepage when you click on the link.
We have covered how to link to a page, but what if we want our users to go to our linked page, but in a new window (so they don't leave our interesting website)?

We can do just this by using the target attribute of the anchor tag and passing in a value of '_blank'.

Opening in a New Window:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Anchor Tags</title>
    </head>
    <body>
        <a href="http://google.com" target="_blank">Go to Google?</a>
    </body>
</html>
```

The above example will render:

Go to Google?

Type out the above code and Try it out!
Notice that the browser will now load Google's homepage in a new window when you click on the link.

Hypertext Reference

The most important attribute for the anchor tag is the href attribute. The ***href*** (Hypertext Reference) attribute will tell the anchor tag where to link to, or where to send the user once clicked on.

This example is slightly different to the previous examples:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Anchor Tags</title>
    </head>
    <body>
        <a href="#">Where to?</a>
    </body>
</html>
```

Linking inside a HTML document

Basically, the '#' symbol can also act as a page anchor, when nothing is assigned to the '#' in the href attribute of an anchor tag- when the user clicks on it, as the link does not have a specific location- it will generally go nowhere.

Now, we can assign id attributes to some of our elements, to use our anchor tags to link to specific parts in our HTML, rather than just having the '#' symbol, we would use:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Anchor Tags and Ids</title>
    </head>
    <body>
        <div id="top">Top of the Page!</div>
        Lorem ipsum dolor sit amet, consectetur adipiscing elit. Proin ultricies ligula eget lorem
malesuada euismod. Phasellus ac interdum mauris. Vivamus hendrerit auctor arcu, vitae iaculis sem
fringilla eget. Pellentesque adipiscing erat at mollis pellentesque. Aenean fringilla lectus et massa
laoreet hendrerit. Proin vehicula ante non mi molestie semper. Vivamus rutrum elit at sem mattis,
quis mollis neque dapibus. Aliquam sit amet justo nunc. Nunc vel pharetra quam. Morbi accumsan velit
at aliquam lobortis. In luctus, ipsum et accumsan viverra, nisi ipsum tristique enim, ut pellentesque
dolor leo quis nisi. Pellentesque porttitor ultrices sollicitudin. Cras ut odio erat. Duis gravida
pretium tellus, lacinia rutrum sapien vestibulum at.

        Proin vestibulum ante mattis, rutrum quam vel, congue nisl. Ut ornare dignissim urna,
consequat sollicitudin urna. Proin condimentum quis risus nec ultricies. Nulla tincidunt massa
ligula, vel volutpat magna consequat eget. Praesent viverra metus a venenatis mollis. Donec laoreet
fermentum lacinia. Donec vitae sodales mauris.
        <a href="#top">Top of Page?</a>
    </body>
</html>
```

I have added a logical or document division at the start of the above example with an id of top.
Ids are a very useful feature of HTML, but for now we are just going to use it for an anchor link (something to link to). We could assign our div with any id value, as long as we reference it in our href attribute of our anchor tag.

Email Links

In HTML we can create a ***matilto***: link, when the user clicks on this link their email client will open and the mailto: value (our email address) will be added to the TO: field.

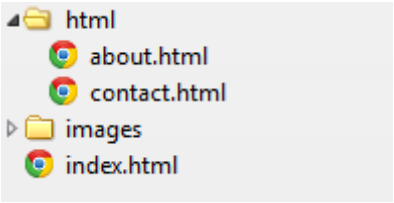
```
<a href="mailto:admin@glorified.me?Subject=Email">Contact Us</a>
```

This makes it easy and enticing for a user to quickly send us some email, regarding our web page.
You may have noticed that I have added '?Subject=Email', this will add “Email” to the subject field within the email. You can change the mailto and Subject values to suit your needs.

Directories

You will often be using a folder structure. The folder structure is a crucial part of good coding practice and helps to tidy up our files

(an images folder and an html folder).
A basic html folder structure would look similar to this:



***Note**: When working with folder in Web Development, we refer to folders as directories.*
As you can see we have a few html files in different locations. We have our index.html file, which; when working in a server environment will automatically load once we navigate or load the specific folder that index.html resides in. As we are not working in a server environment this is not required knowledge at this point in time.
We have an about.html and a contact.html file in our "html" directory. The content of these two files are irrelevant at this point in time. The purpose of the following exercise is to make sure you have a grasp of 'navigating the directory structure'.

Example of linking to the about.html page from index.html:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Anchor Tags</title>
    </head>
    <body>
        <a href="html/about.html">About Us</a>
    </body>
</html>
```

Quiz
1. Create three new html documents named; about.html, contact.html and index.html.
2. Create the same directory structure as shown above and place your new html files in the appropriate directories.
3. Give each html file an appropriate title tag and a level 1 heading (h1).
4. Add a paragraph of appropriate text to each of our html pages and a mailto: link to the contact page.
5. Now that we have some text and a heading for each html document, we need place an image (you can use any image you would like) on each page.

6. Using you skills, you can add an image under the paragraph tag to each html document and a link to each other html document in the directory structure under the image.
7. Test out your html documents by saving them with the same names as detailed above. Make sure the images in your html documents are appearing in the browser and when you click the links under the images, you are taken to the respective html document (i.e. a link to about.html should take you to the about.html page when you click on the link) in your browser.
8. Now that you have completed your three html pages, go ahead and validate them with the online validator and fix any errors that appear.

Inline Elements in Action

Let's check out these inline elements in action.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Inline Elements</title>
    </head>
    <body>
        <strong>Strong text</strong>, <em>Text with emphasis</em>
        <a href="http://glorified.me">Visit glorified.me</a>
        <img src="html5-logo.png" width="100" height="100" alt="HTML5 Logo">
    </body>
</html>
```

Output:

Strong Text, *Text with emphasis* Visit glorified.me

Notice how the contents of the strong, em, anchor and image tags are being displayed on the same line, rather than being displayed on separate lines- like the block level elements.

Inline and Block Elements in Action

So, let's try some block level and inline elements together.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Block Level and Inline Elements</title>
    </head>
```

```
    <body>
        <h2>Level 2 Heading</h2><p>A Paragraph</p><em>and some text with emphasis</em>
        <a href="http://glorified.me">Visit glorified.me</a>
        <img src="html5-logo.png" width="100" height="100" alt="HTML5 Logo">
        <p>Another paragraph</p> and some <strong>strong</strong> text.
    </body>
</html>
```

Output:

Level 2 Heading

A Paragraph

and some text with emphasis Visit glorified.me

Another Paragraph

and some **strong** text.

Tables

Sometimes when we have some information to display on our web page, it makes sense to display that information or data in a table. Tables in HTML are relatively simple. We have the Opening Table Tag and The closing Table tag. Inside of our Table element, we have table rows. Inside the table rows we have tabular data or cells. But, for our table headers, we will be using the table header tags inside our table row.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Tables</title>
    </head>
    <body>
        <table>
            <caption>My HTML5 Table</caption>
            <!-- the caption tag is new in HTML5 -->
            <tr>
                <th>Color</th><th>Width</th><th>Height</th>
```

```
            </tr>
            <tr>
                <td>Black</td><td>55</td><td>99</td>
            </tr>
            <tr>
                <td>Green</td><td>85</td><td>45</td>
            </tr>
        </table>
    </body>
</html>
```

Organizing Tables

If you can keep this breakdown in mind as you read the next few sections, the syntax of the table tags will make much more sense to you. Remember:

- Cells are the basic units of a table; they can contain data elements or column headers.
- Cells are grouped together into rows.
- Rows are grouped together to produce an entire table.

`TABLE` Tag

All table-related tags occur between the `<TABLE>` and `</TABLE>` container tags. Any table-related tags occurring outside of these tags will be ignored.

A good habit you should get into immediately is putting the `</TABLE>` tag into your HTML file when you put the `<TABLE>` tag in. If you don't have a `</TABLE>` tag and you go to a browser to preview your work, the browser won't render the table. This is because browsers read through all of the code to produce a table before rendering it. It has to do this to compute how much space it needs for the table. Once the amount of space is known and allocated, the browser goes back and fills in the cells. Without a `</TABLE>` tag, a browser can't know that it's hit the end of a table and therefore won't render any of it.

Table Row

Tables are made up out of rows, so now you need to know how to define a row. The `<TR>` and `</TR>` tags are used to contain the HTML tags that define the individual cells. You can place as many `<TR>` and `</TR>` tag pairs as you need inside a table, each pair accounting for one row.

So far, then, the code for a basic HTML table with **`m`** rows looks like:

```
<TABLE>
   <TR> ... </TR>    <!-- Row 1 -->
```

```
    <TR> ... </TR>    <!-- Row 2 -->
    ...
    <TR> ... </TR>    <!-- Row m -->
</TABLE>
```

Table Cell

Table cells come in two varieties: *header cells* for headers that appear over a column of data and *data cells* for the individual entries in the entries of the table.

Header Cells

A table header cell is defined with the `<TH>` and `</TH>` tag pair. The contents of a table header cell are automatically centered and appear in boldface, so you typically don't need to format them further.

In a standard table, headers usually comprise the first row so that each column in the table has some type of heading over it. If the basic table you're developing has **`n`** columns of data, the HTML for the table would look like:

```
<TABLE>
    <TR>     <!-- Row 1 -->
        <TH>Header 1</TH>
        <TH>Header 2</TH>
        ...
        <TH>Header n</TH>
    </TR>
    <TR> ... </TR>    <!-- Row 2 -->
    ...
    <TR> ... </TR>    <!-- Row m -->
</TABLE>
```

Data Cells

Data cells usually make up the bulk of the table and are defined by `<TD>` and `</TD>` tags. Text in data cells is left justified by default. Any special formatting like boldface or italics has to be done by including the appropriate formatting tags inside the `<TD>` and `</TD>` pairs.

If we let data cells constitute the rest of the basic table we're constructing, we have the following HTML:

```
<TABLE>
    <TR>     <!-- Row 1 -->
        <TH>Header 1</TH>
        <TH>Header 2</TH>
```

```
            ...
         <TH>Header n</TH>
      </TR>
      <TR>    <!-- Row 2 -->
         <TD>Data element 1</TD>
         <TD>Data element 2</TD>
         ...
         <TD>Data element n</TD>
      </TR>
      ...
      <TR>    <!-- Row m -->
         <TD>Data element 1</TD>
         <TD>Data element 2</TD>
         ...
         <TD>Data element n</TD>
      </TR>
   </TABLE>
```

The HTML above makes for a nice template that you can use whenever starting a table. By filling in the headers and data elements with some genuine information, we can produce a nice table.

```
<TABLE>
   <TR>    <!-- Row 1 -->
      <TH>Player</TH>
      <TH>Goals</TH>
      <TH>Assists</TH>
      <TH>Points</TH>
   </TR>
   <TR>    <!-- Row 2 -->
      <TD>Anne</TD>
      <TD>7</TD>
      <TD>12</TD>
      <TD>19</TD>
   </TR>
   <TR>    <!-- Row 3 -->
      <TD>Eric</TD>
      <TD>4</TD>
      <TD>11</TD>
      <TD>15</TD>
   </TR>
   <TR>    <!-- Row 4 -->
      <TD>Jim</TD>
      <TD>10</TD>
      <TD>14</TD>
```

```
        <TD>24</TD>
    </TR>
</TABLE>
```

Displaying hockey team statistics or any other type of information presented in columns is a good use of HTML tables.

Alignment

The beauty of HTML tables is the precise control you get over the alignment of content in individual cells and over the table itself. There are two types of alignment that you can specify:

- **Horizontal alignment** refers to the alignment of an element across the width of something; for example, the alignment of a header across the width of a cell or the alignment of a table across the width of the page. Horizontal alignment is controlled by the `ALIGN` attribute. You can set `ALIGN` equal to `LEFT`, `CENTER`, or `RIGHT`.
- **Vertical alignment** refers to the alignment of an element between the top and bottom of a cell. You control the vertical alignment of cell contents by setting the `VALIGN` attribute to `TOP`, `MIDDLE`, or `BOTTOM`.

Aligning the Entire Table

You can use the `ALIGN` attribute in the `<TABLE>` tag to specify how the table should be aligned relative to the browser window. Setting `ALIGN` to `LEFT` or `RIGHT` floats the table in the left or right margin, respectively. Floating tables behave much like floating images in that you can wrap text around them.

Alignment within a Row

If you want the vertical or the horizontal alignment to be the same for every cell in a given row, you can use the `VALIGN` and `ALIGN` attributes in the row's `<TR>` tag. Any alignment specified in a `<TR>` tag will override all default alignments.

Alignment within a Cell

HTML permits alignment control all the way down to the cell level. You can prescribe vertical or horizontal alignments in both header and data cells by using the `VALIGN` or `ALIGN` attributes in `<TH>` or `<TD>` tags. Any alignment specified at the cell level overrides any default alignments *and* any alignments specified in a `<TR>` tag.

Setting alignments in individual cells represents the finest level of control you get over table alignment. In theory, you could manually specify vertical and horizontal alignments in every single cell of your tables if you needed to! Unfortunately, it's easy to get lost among all of those VALIGN and ALIGN attributes, especially when it comes to deciding which will take precedence. If you're having trouble mastering table alignment, just remember the following hierarchy:

- Alignments specified in <TD> or <TH> tags override all other alignments, but apply only to the cell being defined.
- Alignments specified in a <TR> tag override default alignments and apply to all cells in a row, unless overridden by an alignment specification in a <TD> or <TH> tag.
- In the absence of alignment specifications in <TR>, <TD>, or <TH> tags, default alignments are used.

Other Table Attributes

These include:

- Captions
- Width of the table
- Borders
- Spacing within and between cells
- How many rows or columns a cell should occupy

Caption

To put a caption on your table, enclose the caption text between the <CAPTION> and </CAPTION> tags. Captions appear centered over the table and the text may be broken to match the table's width. You can also use physical style tags to mark up your caption text. Try the HTML below:

```
<TABLE>
   <CAPTION><B>Team Statistics - 1996-97
Season</B></CAPTION>
   <TR>    <!-- Row 1 -->
      <TH>Player</TH>
      <TH>Goals</TH>
      <TH>Assists</TH>
      <TH>Points</TH>
   </TR>
   <TR>    <!-- Row 2 -->
      <TD>Anne</TD>
      <TD>7</TD>
```

```
            <TD>12</TD>
            <TD>19</TD>
         </TR>
         <TR>    <!-- Row 3 -->
            <TD>Eric</TD>
            <TD>4</TD>
            <TD>11</TD>
            <TD>15</TD>
         </TR>
         <TR>    <!-- Row 4 -->
            <TD>Jim</TD>
            <TD>10</TD>
            <TD>14</TD>
            <TD>24</TD>
         </TR>
      </TABLE>
```

A caption helps give readers a context for the information in your tables.

Width

The `WIDTH` attribute of the `<TABLE>` tag enables you to specify how wide the table should be in the browser window. You can set `WIDTH` to a specific number of pixels or to a percentage of the available screen width.

`WIDTH` is often used to force a table to occupy the entire width of the browser window. If we change the `<TABLE>` tag in the HTML code in the previous section to:

```
<TABLE WIDTH=100%>
```

the table is rendered. The statistics are centered in their columns for easier readability.

Border

You can place a border around your table by using the `BORDER` attribute of the `<TABLE>` tag. `BORDER` is set to the number of pixels wide you want the border to be. A version of our team statistics table with a two pixel border can be acheived. The modified `<TABLE>` tag that accomplishes this effect is:

```
<TABLE WIDTH=100% BORDER=2>
```

Using a border explicitly separates neighboring columns and makes a table more readable.

You can also set BORDER equal to zero. This means that no border will be used and that the browser should give back any space it has reserved to put in a border.

Spacing within a Cell

The distance between an element in a cell and the boundaries of the cell is called *cell padding*. The CELLPADDING attribute of the <TABLE> tag lets you modify the amount of cell padding used in your tables. Typically, Web page authors increase the cell padding from its default value of 1 to put a little extra white space between the contents and the edges of a cell. This has effect of giving the whole table a bit more "room to breathe." The <TABLE> tag used to produce *cell padding* is:

```
<TABLE WIDTH=100% BORDER=2 CELLPADDING=6>
```

You can open your table up with some extra white space by increasing the cell padding.

Spacing between Cells

You also have control over the space between cells. By increasing the value of the CELLSPACING attribute of the <TABLE> tag, you can open a table up even further. Note that the border used between the cells gets accordingly larger as well. The <TABLE> tag used for cellspacing is:

```
<TABLE WIDTH=100% BORDER=2 CELLSPACING=6>
```

Spacing between adjacent cells is controlled by the CELLSPACING attribute.

Spanning Multiple Rows or Columns

By default, a cell occupies or *spans* one row and one column. For most tables, this is usually sufficient. When you start to use tables for layout purposes though, you'll encounter instances where you want a cell to span more than one row or column. HTML supports attributes of the <TH> and <TD> tags that permit this effect.

Using the COLSPAN Attribute

The COLSPAN attribute inside of a <TH> or <TD> tag instructs the browser to make the cell defined by the tag take up more than one column. You set COLSPAN equal to the number of columns the cell is to occupy.

COLSPAN is useful when one row of the table is forcing the table to be a certain number of columns wide while the content in other rows could be accommodated in a smaller number of columns.

The `COLSPAN` attribute lets rows with fewer elements (like the row with the Name field) occupy as many columns as rows with more elements.

Using the `ROWSPAN` Attribute

`ROWSPAN` works in much the same way as `COLSPAN`, except that it allows a cell to take up more than one row.

Table Cell Elements

HTML tables were developed with the intent of presenting columns of information, but that information does not necessarily have to be text-based. There are many types of page elements that you can place in a given table cell:

- **Text**: Text is the most obvious thing to put in a table cell, but don't forget that you can format the text with physical and logical styles, heading styles, list formatting, line and paragraph breaks, and hypertext anchor formatting.
- **Images** You can place an image in a table cell by enclosing an `<IMG>` tag between the `<TD>` and `</TD>` tags that define the cell. This is useful for designing page layout with tables since you aren't constrained to just text.
- **Blank Space** Sometimes it's useful to put a blank cell in a table. You can accomplish this by putting nothing between the cell's defining tags (`<TD></TD>`) or by placing a non-breaking space between the tags (`<TD> </TD>`). Using the non-breaking space is preferable because if you have borders turned on, a cell with a non-breaking space picks up a 3D effect that makes it appear to rise up out of the table.
- **Form Fields** The ability to place form fields inside of a table cell is *very* important, especially when you consider that the prompting text in front of form fields are of varying lengths. By putting prompting text and form fields in a table, you're able to align them all nicely and make the form much more attractive.
- **Other Tables** You can embed one table inside another, though this can induce quite a headache in most people!

Forms

Basics of Form Design

One of the many things you may have noticed on a web page is a contact form for example. We can create a form in HTML by using the form opening and closing tags.

Inside of our Form element we have inputs and a submit button. Some of our inputs will be of type text and our submit button will actually be an input of type submit.
When we work with HTML forms in the real world, there are two attributes that we need to add to our opening form tag.

Method and Action:
For the action attribute, you will enter in the destination of the Form data. The method will take either a POST or GET value. POST and GET are used with server-side processing.
For the action attribute, you would enter in the destination of the Form data.
The method will take either a POST or GET value. You would generally be using POST to submit most forms of data. The main difference between POST and GET is that POST sends data to the server 'behind the scenes', whereas GET will form a query string. A query string consists of name attribute values and the values input by the user. As you can imagine, if we were submitting sensitive data to the server we would definitely not use GET; in the case that we did, all information input by the user would be clearly visible in the URL address bar.
For the purpose of this book, we will be taking a look at forming a query string, using GET.

Type out the following code and select some values from the form and click the submit button. Take a look in your URL Address bar; this is called a query string- the part following the '?'.

```
<form action="#" method="GET">
    <label for="option1G1">Radio Button 1:</label>
    <input id="option1G1" type="radio" name="radio1" value="option1G1">
    <br>
    <label for="option2G1">Radio Button 2:</label>
    <input id="option2G1" type="radio" name="radio1" value="option2G1">
    <br>
    <label for="option1G2">Radio Button 3:</label>
    <input id="option1G2" type="radio" name="radio2" value="option1G2">
    <br>
    <label for="option2G2">Radio Button 4:</label>
    <input id="option2G2" type="radio" name="radio2" value="option2G2">
    <br>
    <label for="cb1">Checkbox 1:</label>
    <input id="cb1" type="checkbox" name="cb1" value="cb1">
    <br>
    <label for="cb2">Checkbox 2:</label>
    <input id="cb2" type="checkbox" name="cb2" value="cb2">
    <br>
    <input type="submit">
    <input type="reset">
</form>
```

Again, type out the following code, load it up in your browser, enter some text into the inputs in the form and click the submit button. Take a look in your URL Address bar.

```
<form action="#" method="GET">
    <label for="first">First Name: </label>
    <input id="first" type="text" name="first">
    <br>
    <label for="last">Last Name: </label>
    <input id="last" type="text" name="last">
    <br>
    <label for="password">Password: </label>
    <input id="password" type="password" name="password">
    <br>
    <input type="submit">
    <input type="reset">
</form>
```

The label tag allows us to add descriptive text regarding the input expected from the user and when the user clicks the label text, focus will be drawn to the input.

To use labels properly, you must give the label a ‘for’ attribute and a value of the ‘for’ attribute that corresponds to the input’s id.

When you use an input of type password, you may think that because the input entered renders as dots that it must be secured. This is a common misconception.
The dots that render for password inputs are only a masking mechanism. Meaning that the input is actually still just the plain text that the user enters, but the password field will mask this to prevent prying eyes when a user is entering in a password. By using GET, you have just seen (in the query string) that the input entered into a password field remains plain text.

Forms and CGI

Forms are the visible or "front-end" portion of interactive pages. Users enter information into form fields and click a button to submit the data. The browser then packages the data, opens an HTTP connection, and sends the data to a server. Things then move to the transparent or "back-end" part of the process.
Web servers are programs that know how to distribute Web pages. They are not programmed to be able to process data from every possible form, so the best they can do is to hand off the form data to a program that *does* know what to do with it. This hand-off occurs with the help of the Common Gateway Interface or CGI-a set of standards by which servers communicate with external programs.
The program that processes the form data is typically called a *CGI script* or a *CGI program*. The script or program performs some manipulation of the data and composes a response-typically an HTML page. The response page is handed back to the server (via CGI) which, in turn, passes it along to the browser that initiated the request.
Forms and CGI are opposite sides of the same coin. Both are essential to create interactive pages, but it is the forms side of the coin that the user sees.
When a CGI script or program composes an HTML page, it is said to be *generating HTML on-the-fly*. The ability to generate pages on-the-fly is what makes custom responses to database and forms submission possible.

Creating Forms

HTML's form support is simple and complete. A handful of HTML tags creates the most popular elements of modern graphical interfaces, including text windows, check boxes and radio buttons, pull-down menus, and push buttons.
Composing HTML forms might sound like a complex task, but you need to master surprisingly few tags to do it.

All form-related tags occur between the <FORM> and </FORM> container tags. If you have more than one form in an HTML document, the closing </FORM> tag is essential for distinguishing between the multiple forms.
Adding a </FORM> tag immediately after creating a <FORM> tag is a good practice; then you can go back to fill in the contents. Following this procedure helps you avoid leaving off the closing tag once you've finished.
Each HTML form has ***three*** main components: the *form header*, one or more named *input fields*, and one or more *action buttons*.

<FORM> Tag

The form header and the <FORM> tag are actually one in the same. The <FORM> tag takes the three attributes shown in Table 2.13. The ACTION attribute is required in every <FORM> tag.

***Table** 2.13: Attributes of the <FORM> Tag*

Attribute	Purpose
ACTION	Specifies the URL of the processing script
ENCTYPE	Supplies the MIME type of a file used as form input
METHOD=GET|POST	Tells the browser how it should send the form data to the server

ACTION

ACTION is set equal to the URL of the processing script so that the browser knows where to send the form data once it is entered. Without it, the browser would have no idea where the form data should go.
The ACTION URL can also contain extra path information at the end of it. The extra path information passes on to the script so that it can correctly process the data. The extra path information is not found anywhere on the form so it is transparent to the user. Allowing for the possibility of extra path information, an ACTION URL has the following form:

- ***protocol*://*server*/*path*/*script_file*/*extra_path_info***

You can use the extra path information to pass an additional file name or directory information to a script. For example, on some servers, the image map facility uses extra path information to specify the name of the map file. The name of the map file follows the path to the image map script.

A sample URL might be ***http://www.your_firm.com/cgi-bin/imagemap/homepage***.
The name of the script is `imagemap`, and `homepage` is the name of the map file used by image map.

METHOD=GET | POST

`METHOD` specifies the HTTP method to use when passing the data to the script and can be set to values of `GET` or `POST`. When you're using the `GET` method, the browser appends the form data to the end of the URL of the processing script. The `POST` method sends the form data to the server in a separate HTTP transaction.
`METHOD` is not a mandatory attribute of the `<FORM>` tag. In the absence of a specified method, the browser uses the `GET` method.

Caution

Some servers may have operating environment limitations that prevent them from processing an URL that exceeds a certain number of characters-typically 1 kilobyte of data. This limitation can be a problem when you're using the `GET` method to pass a large amount of form data. Because the `GET` method appends the data to the end of the processing script URL, you run a greater risk of passing an URL that's too big for the server to handle. If URL size limitations are a concern on your server, you should use the `POST` method to pass form data.

ENCTYPE

The `ENCTYPE` attribute was introduced by Netscape for the purpose of providing a file name to be uploaded as form input. You set `ENCTYPE` equal to the `MIME` type expected for the file being uploaded. `ENCTYPE` does not create the input field for the file name; rather, it just gives the browser a heads-up as to what kind of file it is sending. When prompting for a file to upload, you'll need to use an `<INPUT>` tag with `TYPE` set equal to `FILE`.
As an example of the three `<FORM>` tag attributes, examine the following HTML:

```
<FORM ACTION="process_it.cgi" METHOD=POST
ENCTYPE="text/html">
Enter the name of the HTML file to validate:
<INPUT TYPE="FILE" NAME="html_file">
<INPUT TYPE="SUBMIT" VALUE="Validate it!">
</FORM>
```

The form header of this short form instructs the server to process the form data using the program named `process_it.cgi`. Form data is passed using the `POST` method and the expected type of file being submitted is an HTML file.

Named Input Fields
The named input fields typically comprise the bulk of a form. The fields appear as standard GUI controls such as text boxes, check boxes, radio buttons, and menus. You assign each field a unique name that eventually becomes the variable name used in the processing script.

Warning
If you are not coding your own processing scripts, be sure to sit down with your programmer to agree on variable names. The names used in the form should exactly match those used in coding the script.

You can use several different GUI controls to enter information into forms. The controls for named input fields appear in Table 2.14.

***Table** 2.14: Types of Named Input Fields*

Field Type	*HTML Tag*
Text Box	`<INPUT TYPE="TEXT">`
Password Box	`<INPUT TYPE="PASSWORD">`
Checkbox	`<INPUT TYPE="CHECKBOX">`
Radio Button	`<INPUT TYPE="RADIO">`
Hidden Field	`<INPUT TYPE="HIDDEN">`
Images	`<INPUT TYPE="IMAGE">`
File	`<INPUT TYPE="FILE">`
Text Window	`<TEXTAREA>...</TEXTAREA>`
Menu	`<SELECT>...<OPTION>...</SELECT>`

***<INPUT>* Tag**
You'll note in Table 2.14 that the `<INPUT>` tag handles the majority of named input fields. `<INPUT>` is a stand-alone tag that, thanks to the many values of its `TYPE` attribute, can place most of the fields you need on your forms. `<INPUT>` also takes other attributes, depending on which `TYPE` is in use. These additional attributes are covered for each type, as appropriate, over the next several sections.

Note

The <INPUT> tag and other tags that produce named input fields just create the fields themselves. You, as the form designer, must include some descriptive text next to each field so that users know what information to enter. You may also need to use line breaks, paragraph breaks, and non-breaking space to create the spacing you want between form fields.

`Text` and `Password` Fields

`Text` and `password` fields are simple data entry fields. The only difference between them is that text typed into a `password` field appears on-screen as asterisks (`*`).

Caution

Using a `password` field may protect users' passwords from the people looking over their shoulders, but it does not protect the password as it travels over the Internet. To protect password data as it moves from browser to server, you need to use some type of encryption or similar security measure. Authentication of both the server and client by using signed digital certificates are two other steps you can take to keep Internet transactions secure.

The most general `text` or `password` field is produced by the HTML (attributes in square brackets are optional):

```
<INPUT    TYPE="{TEXT|PASSWORD}"    NAME="Name"
[VALUE="default_text"]
[SIZE="width"] [MAXLENGTH="wmax_idth"]>
```

The `NAME` attribute is mandatory because it provides a unique identifier for the data entered into the field.

The optional `VALUE` attribute allows you to place some default text in the field, rather than have it initially appear blank. This capability is useful if a majority of users will enter a certain text string into the field. In such cases, you can use `VALUE` to put the text into the field, thereby saving most users the effort of typing it.

The optional `SIZE` attribute gives you control over how many characters wide the field should be. The default `SIZE` is typically 20 characters, although this number can vary from browser to browser. `MAXLENGTH` is also optional and allows you to specify the maximum number of characters that can be entered into the field.

Note

Previously, the `SIZE` attribute took the form `SIZE="width,height"`,

where setting a `height` *(other than 1) produced a multiline field. With the advent of the* `<TEXTAREA>...</TEXTAREA>` *tag pair for creating multiline text windows,* `height` *has become something of a remnant and is ignored by most browsers.*

`Text` *and* `password` *fields enable you to create a login page for your site.*

Check Boxes

Check boxes are used to provide users with several choices. Users can select as many choices as they want. An `<INPUT>` tag to produce a check box option has the following syntax:

```
<INPUT TYPE="CHECKBOX" NAME="Name"
VALUE="Value" [CHECKED]>
```

Each check box option is created by its own `<INPUT>` tag and must have its own unique `NAME`. If you give multiple check box options the same `NAME`, the script has no way to determine which choices the user actually made.

The `VALUE` attribute specifies what data is sent to the server if the corresponding check box is chosen. This information is transparent to the user. The optional `CHECKED` attribute pre-selects a commonly selected check box when the form is rendered on the browser screen.

When designing your garden at www.garden.com, you can choose what color flowers and leaves you want and when the plants will bloom.

Each flower and leaf color preference is placed on the page with its own `<INPUT>` *tag with* `TYPE="CHECKBOX"`*.*

Note

If they are selected, check box options show up in the form data sent to the server. Options that are not selected do not appear.

Radio Buttons

When you set up options with a radio button format, you should make sure that the options are mutually exclusive so that a user won't try to select more than one.

The HTML code to produce a set of three radio button options is as follows:

```
<INPUT TYPE="RADIO" NAME="Name" VALUE="VALUE1"
[CHECKED]>Option 1<P>
<INPUT TYPE="RADIO" NAME="Name"
VALUE="VALUE2"> Option 2<P>
```

```
<INPUT TYPE="RADIO" NAME="Name"
VALUE="VALUE3">Option 3<P>
```

The VALUE and CHECKED attributes work exactly the same as they do for check boxes, although you should have only one pre-selected radio button option. A fundamental difference with a set of radio button options is that they all have the same NAME. This is permissible because the user can select only one of the options.

Looking for the closest authorized Adobe reseller? You can choose which type of reseller you want from a set of radio buttons on the search page.

Each radio button option is created by an <INPUT> tag with TYPE set to RADIO.

Hidden Fields

Technically, hidden fields are not meant for data input. You can send information to the server about a form without displaying that information anywhere on the form itself. The general format for including hidden fields is as follows:

```
<INPUT TYPE="HIDDEN" NAME="name" VALUE="value">
```

One possible use of hidden fields is to allow a single general script to process data from several different forms. The script needs to know which form is sending the data, and a hidden field can provide this information without requiring anything on the part of the user.

Another application of hidden fields is for carrying input from one form to another. This lets you split a long form up into several smaller forms and still keep all of the user's input in one place.

Note

Because hidden fields are transparent to users, it doesn't matter where you put them in your HTML code. Just make sure they occur between the <FORM> and </FORM> tags that define the form that contains the hidden fields.

Files

You can upload an entire file to a server by using a form. The first step is to include the ENCTYPE attribute in the <FORM> tag. To enter a file name in a field, the user would need the <INPUT> tag with TYPE set equal to FILE:

```
<FORM ACTION="whatever.cgi"
ENCTYPE="application/x-www-form-urlencoded">
```

```
What file would you like to submit: <INPUT
TYPE="FILE" NAME="your_file">
...
</FORM>
```

Being able to send an entire file is useful when submitting a document produced by another program-for example, an Excel spreadsheet, a ***resumé*** in Word format, or just a plain Notepad text file.

Multiple Line Text Input

Text and password boxes are used for simple, one-line input fields. You can create multiline text windows that function in much the same way by using the <TEXTAREA> and </TEXTAREA> container tags. The HTML syntax for a text window is as follows:

```
<TEXTAREA NAME="Name" [ROWS="rows"]
[COLS="columns"]>
Default_window_text
</TEXTAREA>
```

The NAME attribute gives the text window a unique identifier just as it does with the variations on the <INPUT> tag. The optional ROWS and COLS attributes allow you to specify the dimensions of the text window as it appears on the browser screen. The default number of rows and columns varies by browser.

The text that appears between the <TEXTAREA> and </TEXTAREA> tags shows up in the input window by default. To type in something else, users need to delete the default text and enter their text.

Multiline text windows are ideal for entry of long pieces of text such as feedback comments or e-mail messages. Some corporate sites on the Web that collect information on potential employees may ask you to copy and paste your entire ***resumé*** into multiline text windows!

Menus

The final technique for creating a named input field is to use the <SELECT> and </SELECT> container tags to produce pull-down or scrollable option menus. The HTML code used to create a general menu is as follows:

```
<SELECT NAME="Name" [SIZE="size"] [MULTIPLE]>
<OPTION [SELECTED]>Option 1</OPTION>
<OPTION [SELECTED]>Option 2</OPTION>
<OPTION [SELECTED]>Option 3</OPTION>
...
```

```
<OPTION [SELECTED]>Option n</OPTION>
</SELECT>
```

In the <SELECT> tag, the NAME attribute again gives the input field a unique identifier. The optional SIZE attribute lets you specify how many options should be displayed when the menu renders on the browser screen. If you have more options than you have space to display them, you can access them either by using a pull-down window or by scrolling through the window with scroll bars. The default SIZE is 1. If you want to let users choose more than one menu option, include the MULTIPLE attribute. When MULTIPLE is specified, users can choose multiple options by holding down the Control key and clicking the options they want.

Note
If you specify the MULTIPLE attribute and SIZE=1, a one-line scrollable list box displays instead of a drop-down list box. This box appears because you can select only one item (not multiple items) in a drop-down list box.

Each option in the menu is specified inside of its own <OPTION> container tag. If you want an option to be pre-selected, include the SELECTED attribute in the appropriate <OPTION> tag. The value passed to the server is the menu item that follows the <OPTION> tag unless you supply an alternative using the VALUE attribute. For example:

```
<SELECT NAME="STATE">
<OPTION VALUE="NY">New York</OPTION>
<OPTION VALUE="DC">Washington, DC</OPTION>
<OPTION VALUE="FL">Florida</OPTION>
...
</SELECT>
```

In the menu above, the user clicks a state name, but it is the state's two-letter abbreviation that passes to the server.

Action Buttons
The handy <INPUT> tag returns to provide an easy way of creating the form action buttons you see in many of the preceding figures. Buttons can be of two types: ***Submit*** and ***Reset***. Clicking a Submit button instructs the browser to package the form data and send it to the server. Clicking a Reset button clears out any data entered into the form and sets all the named input fields back to their default values.

Submit and Reset Buttons

Any form you compose should have a Submit button so that users can submit the data they enter. The one exception to this rule is a form containing only one input field. For such a form, pressing Enter automatically submits the data. Reset buttons are technically not necessary but are usually provided as a user courtesy.

To create Submit or Reset buttons, use the <INPUT> tags as follows:

```
<INPUT TYPE="SUBMIT" VALUE="Submit Data">
<INPUT TYPE="RESET" VALUE="Clear Data">
```

Use the VALUE attribute to specify the text that appears on the button. You should set VALUE to a text string that concisely describes the function of the button. If VALUE is not specified, the button text reads Submit Query for Submit buttons and Reset for Reset buttons.

Images as Submit Buttons

You can create a custom image to be a submit button for your forms and set up the image so that clicking it instructs the browser to submit the form data. To do this, you set TYPE equal to IMAGE in your <INPUT> tag and provide the URL of the image you want to use with the SRC attribute:

```
<INPUT TYPE="IMAGE"
SRC="images/submit_button.gif">
```

You can also use the ALIGN attribute in this variation of the <INPUT> tag to control how text appears next to the image (TOP, MIDDLE, or BOTTOM) or to float the image in the left or right margins (LEFT or RIGHT).

Some sites lets you search for flight information by clicking an image rather than a regular submit button.

An <INPUT> tag with TYPE set to IMAGE is the key to creating your own submit button.

Multiple Submit Buttons

It's possible to have more than one submit button on a form, although there is not yet consistent browser support for multiple submit buttons.

You distinguish between submit buttons by using the NAME attribute in the <INPUT> tags used to create the buttons. For example, you might have:

```
<INPUT TYPE="SUBMIT" NAME="SEARCH"
VALUE="Conduct Search">
```

```
<INPUT TYPE="SUBMIT" NAME="ADD" VALUE="Add to
Database">
```

to produce buttons that allow users to search the information they've entered or add the information they've entered to a database.
Because there is only tentative support for multiple submit buttons, you may want to hold off on implementing them until they are standard.

Passing Form Data
Once a user enters some form data and clicks a submit button, the browser does ***two*** things. ***First***, it packages the form data into a single string, a process called encoding. ***Then*** it sends the encoded string to the server by either the `GET` or `POST` HTTP method.

URL Encoding
When a user clicks the Submit button on a form, his or her browser gathers all the data and strings together in `NAME=VALUE` pairs, each separated by an ampersand (`&`) character. This process is called ***encoding***. It is done to package the data into one string that is sent to the server.
Consider the following HTML code:

```
<FORM  ACTION="http://www.habeebmamman.com/cgi-
bin/form.cgi" METHOD="POST">
     <INPUT TYPE="TEXT" NAME="first">
     <INPUT TYPE="TEXT" NAME="last">
     <INPUT TYPE="SUBMIT">
</FORM>
```

If a user named Joe Schmoe enters his name into the form produced by the preceding HTML code, his browser creates the following data string and sends it to the CGI script:

```
first=Joe&last=Schmoe
```

If the `GET` method is used instead of `POST`, the same string is appended to the URL of the processing script, producing the following *encoded URL*:

```
http://www.server.com/cgi-
bin/form.cgi?first=Joe&last=Schmoe
```

A question mark (`?`) separates the script URL from the encoded data string.

Storing Encoded URLs
As you learned in the previous discussion of URL encoding, packaging form data into a single text string follows a few simple formatting rules.

Consequently, you can fake a script into believing that it is receiving form data without using a form. To do so, you simply send the URL that would be constructed if a form were used. This approach may be useful if you frequently run a script with the same data set.
For *example*, suppose you frequently search the Web index Yahoo for new documents related to the scripting language JavaScript. If you are interested in checking for new documents several times a day, you could fill out the Yahoo search query each time. A more efficient way, however, is to store the query URL as a bookmark. Each time you select that item from your bookmarks, a new query generates as if you had filled out the form. The stored URL would look like the following:

- ***http://search.yahoo.com/bin/search?p=JavaScript***

Further encoding occurs with data that is more complex than a single word. Such encoding simply replaces spaces with the plus character and translates any other possibly troublesome character (control characters, the ampersand and equal sign, some punctuation, and so on) to a percent sign followed by its hexadecimal equivalent. Thus, the following string:

```
I love HTML!
```

becomes:

```
I+love+HTML%21
```

HTTP Methods

You have two ways to read the form data submitted to a CGI script, depending on the `METHOD` the form used. The type of `METHOD` the form used-either `GET` or `POST`-is stored in an environment variable called `REQUEST_METHOD`, and based on that, the data should be read in one of the following ways:

- If the data is sent by the `GET` method, the input stream is stored in an environment variable called `QUERY_STRING`. As noted previously, this input stream usually is limited to only about 1 kilobyte of data. This is why `GET` is losing popularity to the more flexible `POST`.
- If the data is submitted by the `POST` method, the input string waits on the server's input device, with the available number of bytes stored in the environment variable `CONTENT_LENGTH`. `POST` accepts data of any length, up into the megabytes, although it is not very common yet for form submissions to be that large.

Example Forms
Web designers have discovered many ways to use forms to enhance users' experiences. This chapter closes with a quick look at some examples of creative uses of Web forms.

Online Searches
Google, Yahoo or Alta Vista has quickly become one of the most prolific online search indexes on the Web. Alta Vista searches frequently return tens of thousands of results and can include Web documents and posts to Usenet newsgroups.

Online Registration
If you went to Macromedia's User Conference in September 1996, you might have registered using a form. The extensive form collects attendee information, which registration option you'd like, what seminar you want to attend, and how you want to pay.
Many technology industry conferences now permit online registration via a Web form.

Creating a Custom Page
Microsoft and other companies now offer customized pages each time users visit their sites. Users can supply information on how to configure the page and the company's server uses this information to generate a fresh and tailored page to the user at each visit.

Online Shopping
As Web surfers gain more confidence in the security of business transactions over the Internet, you'll see more and more online stores cropping up. (***Please See Appendix-B for Sample e-Commerce Website***)

Quiz
1. Create a new .html file titled tables-forms.html.
2. Add a table (4x7) and type in 4 of your favourite bands and seven of their best songs, to fill out the table. By doing this simple exercise, you will learn how to create a HTML table of any size.
3. Under the new table that you have created; add a form.
4. This form will have 4 radio inputs, 3 Check boxes, 1 text and 1 password input field.
5. Appropriately name each input and test that you have done so correctly by entering in text/ making selections and submitting the form (clicking 'submit').
Pay close attention to the query string that has formed in your URL address bar.

6. Create some new inputs of type: color, number, URL, date and email and test them out in your browser!

HTML5 Specific Elements

So far we have only really learned about general HTML tags and elements, now it's time to get into some HTML5 specific elements. As you may notice when you start to work with HTML and build some of your own web pages, it would be really helpful if there was a logical way to define the header and footer of our web page With HTML5, we can just that with the header and footer tags:

Header & Footer

```
<header>This is the Header Element</header>
```

```
<footer>This is the Footer Element</footer>
```

The header element defines a header for our html document. We could also have multiple headers in our html document, to define headers for different parts of our html.

The footer element defines footer for our html document; just as we can with the header element we can also have multiple footer elements within our html document defining footers for different parts of our html.

Along with these new header and footer elements, there are also a few more important elements that have been introduced with HTML5.

Navigation

We can now define a navigational element with the nav tag:

```
<nav>This is the Nav Element</nav>
```

We would use this element to hold our **main** navigational content.

Section

We can now define a ‘section’ of our HTML document. Do keep in mind that the contents of a section should be related.

```
<section>This is the Section Element</section>
```

Article

We can now define an 'article' within our HTML document. Within a section, we could have several articles:

```
<section>
    <article>
        This is the Article Element
    </article>
    <article>
        This is another Article Element
    </article>
</section>
```

Aside

We can now define an 'aside'; a part of our HTML content that is 'aside' from our main content, but is still related:

```
<article>
    This is another Article Element
    <aside>
        This is the Aside Element
    </aside>
</article>
```

The Meter Element

One of the really cool things with HTML5 is the ability to add meters. We define the meter element with the opening and closing meter tags. The HTML5 meter tag will take a few attributes, min, max and value. The Min and Max values will define a range in which our value will be compared. For example, with a min of 0 and a max of 100, a value of 50 will show a ‘gauge’ that is 50% complete:

```
<meter min="0" max="100" value="50">This is the Meter Element</meter>
```

The text that I have entered in between the opening and closing meter tag is fallback text. This fallback text will be rendered in older browsers that do not yet support the meter tag.

The new HTML5 Elements in Action

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>HTML5</title>
    </head>
    <body>
        <header>This is the header element</header>
        <nav>This is the nav element</nav>
        <section>
            <article>
                This is the article element
                <aside>
                    This is the aside element
                </aside>
            </article>
            <article>
                This is another article
            </article>
            <article>
                And, yet another article
            </article>
        </section>
        <footer>This is the footer element</footer>
    </body>
</html>
```

Video

One of the greatest features of HTML5 is the ability to implement a fully featured Video with Controls.

We start out with the opening Video tag, passing in a height & width and the controls attribute, so the video player will have controls. Next we need to add the source tag and actually add the Video URL to the ***src*** attribute of the source tag. Again this can be an absolute or relative URL.

Along with our actual video ***src***, we need to tell the browser the mime-type. A ***mimetype*** basically lets the browser know what kind of format or media type to expect.

To provide a similar cross-browser experience you will need to supply multiple formats, contained within separate source tags.

Here we have the video Element set to 640x480px with controls displayed; some fallback text (to display some text in a browser that doesn't support the video element) and two sources- each have a different format and mime-type.

```
<video width="640" height="480" controls>
    <source src="video.mp4" type="video/mp4">
    <source src="video.webm" type="video/webm">
    Fallback Text
</video>
```

Video Formats

The most important aspect of video formats is deciding which file format to use. Depending on the format you select, it is possible that you will block out potential users of your site. Another important consideration is that some video file formats are more efficient than others at storing video segments. Consequently, the more efficient formats create much smaller file sizes than inefficient formats.

***Table** 2.15: A Comparison of Video File Formats*

File Format	Extension	File Size
QuickTime	QT, MOV	Medium
MPEG	MPEG, MPE, MPG	Small
Indeo	AVI	Large

MPEG

MPEG (Moving Picture Expert Group) is a group of people who are trying to create a standard for digital video. Much of MPEG's work takes it outside the reach of the Internet; however, the group did create the MPEG standard. This standard is, by far, the most efficient way to store video clips. MPEG files are usually 3-10% smaller than files in other formats.

QuickTime

The QuickTime digital video format was developed by Apple computers. This format generally creates moderate sized files for video clips. This is probably the most rapidly spreading video format on the Internet. While not as efficient as MPEG, QuickTime offers reasonable file size and video playback performance.

AVI

AVI, or Intel Indeo, is another popular video file format. As you might have guessed, AVI is created and maintained by Intel, but is supported by other companies. This means Intel defines the specifications for AVI. However, it is left up to Microsoft, and others, to implement those specs.

Hardware Options

There are a number of different ways to get a video image into the computer. Some people may already have this ability built into their computers, but now know it. Others will need to buy some video equipment and, after purchasing such equipment, will probably have to buy an expensive expansion card. This card enables the computer to create video clips.

There are ***four*** basic approaches to transforming a video sequence into a computer-usable format. The first approach is to use a video camera and digitize the video segment directly off the video feed. The second approach is to use a VCR to convert a video playback into digital form. The third approach is to purchase a digital camera that automatically stores video segments in a computer file. The fourth, and easiest, approach is to purchase a computer with the necessary equipment already included.

You can also try

HTML `EMBED` tag and specify the `SRC` attribute to point to your Web page. You can treat the `EMBED` tag as a generic catch-all version of the graphic image `IMG` tag. You can add any sort of video clip by putting the following in your Web page:

```
<EMBED SRC="filename.ext">
```

You can also specify the height and width of the clip by using the `HEIGHT` and `WIDTH` attributes.

Microsoft proposed an attribute extension, namely the `DYNSRC` attribute. This attribute behaves just like the `SRC` one, except that the file specified for it is a video clip.

Audio

As you have just learned about HTML5 Video, HTML5 Audio isn't going to be all that difficult to grasp. HTML5 audio isn't all that different from HTML5 Video (apart from the fact that it's audio and not video). With the audio element, we do not need to set the size, but it is ALWAYS good practise to use the controls attribute so your users can actually operate the Audio Player.

Just as we can have multiple sources in our video element, we do the same thing with our audio element; passing in audio files and appropriate mime-types.

```
<audio controls>
      <source src="audio.mp3" type="audio/mpeg">
      <source src="audio.ogg" type="audio/ogg">
      Fallback Text
</audio>
```

Audio Content Source

The Internet
By using any search engine and key words such as "audio clips," "MIDI files," and "aiff Beetles," you will quickly find many pages with the type of music that you want, already in the proper format.

CDs or DAT tapes
A wealth of music is available on CD. There are many musicians and music compilation houses that will sell you a CD full of public domain music.

Recorded from "Live Sources"
Narration, of course, is something that you can record yourself. Make sure that it meets the quality standards that people throughout your target audience will find acceptable. Creating professional quality audio is a complicated business.

Your Own Music
If you are a musician, any of the previous concerns become less important. You can create your own music with old fashioned analog instruments, and then have the music digitized for use on your Web site.

Audio Formats
There are many file formats available to choose from. Common formats are: Au, Aiff, Wav, MPEG, MIDI, MOD etc.

You can also try

You can include an audio file as a **hyperlink**, just like any other type of file. To include audio make sure that the audio clip is in a format that can be played by all important browsers.
Include the audio clip's name in the <HREF> tag example:

```
<A HREF="openingdoor.aiff">Opening Door Sound
Effect</A>
```

Make sure that the audio clip loads and plays properly on different platforms and with the versions of browsers that you are targeting.

You can include an audio file as a **Background,**

```
<BGSOUND SRC="myvoice.wav">
```

The EMBED tag can be used to play an audio file in the background by including it like this:

```
<EMBED
SRC="http://www.habeebmamman.com/audiofile.aif
f" HIDDEN=TRUE AUTOSTART=TRUE>
```

In the above example, SRC points to the files URL, the argument HIDDEN=TRUE indicates that there will be no controls present, and AUTOSTART=TRUE indicates that the audio will start playing in the background as soon as the file is loaded.

To add an audio control panel to your Web page, use the following tag:

```
<EMBED SRC="URL to audio file" HEIGHT=60
WIDTH=144 CONTROLS=CONSOLE>
```

An actual control panel is implemented like this:

```
<EMBED SRC="moonlight.aiff" HEIGHT=60
WIDTH=144 CONTROLS=CONSOLE>
```

You can also add a smaller control console with the following line:

```
<EMBED SRC="moonlight.aiff" HEIGHT=15 WIDTH=144
CONTROLS=SMALLCONSOLE AUTOSTART=TRUE>
```

Additions to HTML

HTML has been continuously evolving since its introduction in the late 1980s. The HTML standard is an open standard which means, in part, that members of the entire Internet community are welcome to submit proposals for additional HTML tags.

Many individuals and corporations are making proposals to the World Wide Web Consortium (W3C) for consideration in future releases of the standard. Some of these include the following:

- Spyglass
- Sun Microsystems
- Novell
- SoftQuad
- IBM

Expected Additions

- **Better indexing**

Tags to define search ranges within a document will make it easier for Web robots to index your documents.

The <RANGE> tag was proposed for HTML 3.0, but was not made part of the HTML 3.2 standard. According to the proposal, placing a <RANGE> tag in the document head allows you to set up a range in the document for searching. <RANGE> takes the CLASS attribute, which is set equal to SEARCH to set up a search range, and the FROM and UNTIL attributes, which designate the beginning and end of the search range. A sample <RANGE> tag might look like the following:

```
<RANGE CLASS=SEARCH FROM="start" UNTIL="finish">
```

The "start" and "finish" markers are set up in the body of the document using the <SPOT ID="start"> and <SPOT ID="finish"> tags at the points where you want the search range to begin and end, respectively.

- **Creating tab stops**

As HTML adds more support for layout control, you may see tags that enable you to define your own tab stops on a browser screen.

Proposed the addition of a <TAB> tag, which allows you to set up your own tab stops in a document. To use a tab stop, you need first to define it using the ID attribute:

```
My first tab stop is <TAB ID="first">here, followed
by some other text.
```

The preceding HTML sets up the first tab stop in front of the letter "h" in the word "here." To use the tab stop, you use the <TAB> tag with the TO attribute:

```
<TAB TO="first">This sentence starts below the word
"here."
```

On the browser screen, the "T" in the word "This" is aligned directly below the "h" in the word "here."

With the implementation of cascading style sheets, which permit good control over indentation and other layout attributes, it is unclear as to whether the <TAB> tag will receive consideration for later standards.

- **New logical styles**

Expanded logical style tags allow you to better describe the nature of your content.

While there are no new logical styles in HTML 3.2, several of them were proposed as part of HTML 3.0. The styles are shown in ***Table*** 2.16. Because many of these proposals are still under consideration, it's still possible that you'll see any or all of these tags used in the future.

All of the tags shown in *Table* 2.16 are container tags. The closing tags are left off in the interest of space.

***Table** 2.16: New Logical Styles Proposed in HTML 3.0*

Style Name	Tag
Abbreviation	<ABBREV>
Acronym	<ACRONYM>
Author's name	<AU>
Deleted text	<DEL>
Inserted text	<INS>
Person's name	<PERSON>
Short quotation	<Q>

- **Creating non-scrolling regions**

You can have content on the browser screen all of the time if you place it as a *banner*-a region on the page that does not scroll.

Banners are defined as regions in a document that should not scroll.
You could reference externally defined banners by using the <LINK> tag in the document head. The REL attribute is set to BANNER and the HREF attribute is set to the URL of the document containing the banner information. For example:

```
<LINK REL="BANNER"
HREF="http://www.your_firm.com/navigation.html">
```

Referencing an external banner provides the advantage of only having to update one file if changes need to be made.
You could also define a banner right in your document by using the <BANNER> and </BANNER> tags. Any text or graphics between these two tags become banner elements for your page.

- **Placing larger graphics**

The proposed <FIG> tag enables you to place captions and credits around a large image graphic and supports its own version of client-side image mapping.

The <FIG> tag was proposed as an alternative to the <IMG> tag for larger graphics. As you might expect, <FIG> requires the SRC attribute to specify the URL of the image file to be loaded. <FIG> can also take the attributes shown in ***Table*** 2.17. The BLEEDLEFT and BLEEDRIGHT values of the ALIGN attribute align the figure all the way to the left and right edges of the browser window, respectively.

***Table** 2.17: Attributes of the <FIG> Tag*

Attribute	Purpose
`SRC="url"`	Gives the URL of the image file to load
`NOFLOW`	Disables the flow of text around the figure
`ALIGN=LEFT|RIGHT|CENTER|JUSTIFY|BLEEDLEFT|BLEEDRIGHT`	Specifies an alignment for the figure
`UNITS=unit of measure`	Specifies a unit of measure for the WIDTH and HEIGHT attributes (default is pixels)
`WIDTH=width`	Specifies the width of the image in units designated by the UNITS attribute
`HEIGHT=height`	Specifies the height of the image in units designated by the UNITS attribute
`IMAGEMAP`	Denotes the figure as an image map

The <FIG> tag is different from the <IMG> tag in that it has a companion </FIG> tag. Together, <FIG> and </FIG> can contain text, including captions and photo credits, which are rendered with the figure. Captions are enclosed with the <CAPTION> and </CAPTION> tags, and photo credits are enclosed with the <CREDIT> and </CREDIT> tags. Regular text found between the <FIG> and </FIG> tags wraps around the figure unless the NOWRAP attribute is specified.

Another feature proposed for the <FIG> and </FIG> tag pair is the capability to overlay two images. This is accomplished with the <OVERLAY> tag, which specifies a second image to overlay the image given in the <FIG> tag. HTML to produce an overlay might look like the following:

```
<FIG SRC="main_image.gif" WIDTH=250 HEIGHT=186
ALIGN=LEFT>
     <OVERLAY SRC="overlay.gif">
     <P>The image to the left is actually two
images, one on top of the other.</P>
</FIG>
```

According to the proposal, the <FIG> tag provides another method for implementing client-side image maps. The key to using the <FIG> and </FIG> tags for a client-side image map is that these tags can contain text that acts as an alternative to the image being placed by them. Thus, any text between the <FIG> and </FIG> tags is much like text assigned to the ALT attribute of the <IMG> tag. For example, the HTML:

```
<IMG SRC="logo.gif" ALT="Company Logo" WIDTH=120
HEIGHT=80>
```

and

```
<FIG SRC="logo.gif" WIDTH=120 HEIGHT=80>
Company Logo
</FIG>
```

essentially do the same thing.

- **Publishing mathematical content**

After special tags and entities are accepted into standard HTML, much of the headache of publishing mathematical documents to the Web will disappear.

The rendering of mathematical symbols and equations has always been tricky on the Web. Authors used to have to place symbols, Greek letters, and other mathematical characters as *separate images*. When you consider that a browser has to open a separate HTTP connection to download an image, it becomes easy to imagine how long it might take to download a page with heavy mathematical content. Clearly, then, a better way to publish mathematical documents on the Web is needed.

Mathematical Tags

All mathematical content is enclosed between the and tags. <MATH> can take the CLASS attribute if the mathematical content is restricted to a certain mathematical sub-discipline:

```
<MATH CLASS="ALGEBRA.LINEAR">
```

Or it can take the CLASS attribute if the content is restricted to another branch of scientific study:

```
<MATH CLASS="PHYSICS">
```

A number of other tags are valid inside the and tags. These are summarized in Table 2.18.

***Table** 2.18: Mathematical HTML Tags*

Tag	Purpose
<ABOVE>	Places a line, arrow, or symbol over an expression
<ARRAY>	Used to create matrices
<BAR>	Places a bar over an expression
<BELOW>	Places a line, arrow, or symbol under an expression
<BOX>	Used for hidden grouping symbols
<DOT>	Places a single dot over an expression
<DDOT>	Places a double dot over an expression
<HAT>	Places a hat (^) over an expression
<OVER>	Places one expression over another
<ROOT>	Used to render a root other than the square root
<SQRT>	Used to render a square root sign
<SUB>	Used to create a subscript
<SUP>	Used to create a superscript
<TEXT>	Inserts plain text inside a math element
<TILDE>	Places a tilde (~) over an expression
<VEC>	Denotes an expression as a vector by placing an arrow over it

Additionally, there are tags you can use to override the default text formatting inside the and tags. The <B> container tag renders its contents in boldface and the <T> container tag renders its contents in an upright font. <BT> combines the effects of the <B> and <T> tags.

One of the greater obstacles to rendering mathematical content on a browser screen is all of the special characters needed.
Each of these special characters has an HTML entity proposed to represent it in an HTML document. Recall that entities begin with an ampersand (&) and end with a semicolon (;).
For example, you could use the HTML:

```
&int;2x - 1 dx = x^2 - x + c
```

to produce

```
2x - 1 dx = x^2 - x + c
```

CHAPTER THREE
CSS

Basics of CSS

Cascading Style Sheets are now the standard way to define the presentation of your HTML pages, from fonts and colours to the complete layout of a page. They are much more efficient than using HTML on every page to define the look of your site.

CSS is becoming a more important language to know every day.
There are two competing forces in Web page authoring: ***content*** and ***presentation***. When HTML was first released, the tags were largely focused on content and they descriptively defined the various parts of a document: a heading, a paragraph, a list, and so on. Over time, instructions were added to help with presentation issues at the font level. These instructions included tags for boldface, italics, and typewriter styles.

Then, as graphical browsers became standard equipment, there was a greatly magnified focus on presentation. In particular, Netscape began introducing proprietary extensions to HTML that only its browser could render properly. These extensions generally produced attractive effects on pages and users began using Netscape together. This compelled content authors to write to the Netscape Navigator browser-a practice that often produced dreadful results on other browsers.
Not to be left out, Microsoft began producing its own browser-Internet Explorer-and its own proprietary HTML extensions with it. This started the ever-escalating battle between Netscape and Microsoft, each trying to outdo the other in each new beta release of its browser. The content authors watching the battle were frequently left confused and frustrated since it was hard to tell which browser to write for and how long it would be before the next new set of bells and whistles became available.

As designers push for more control over page attributes like margins and line spacing, the evolution of HTML stands at a division in the road. One path sees the continued introduction of proprietary tags by the people making the browsers-a path that will lead HTML into even muddier waters. The other path sees an explicit separation of content and presentation by introducing *HTML style sheets*-documents that provide specifications for how content should look on screen.

By separating these two otherwise competing forces, HTML is free to evolve as a language that describes document content and will be less susceptible to seemingly endless extensions by browser software companies.

The World Wide Web Consortium is already pushing the idea of style sheets. It has reserved a tag for embedding style information within an HTML document. It is also considering proposals for a general style sheet language that could be used to describe how a document should look just as HTML describes what the page contains. This chapter surveys the approaches to style sheets as they have been proposed and implemented.

Definition of CSS

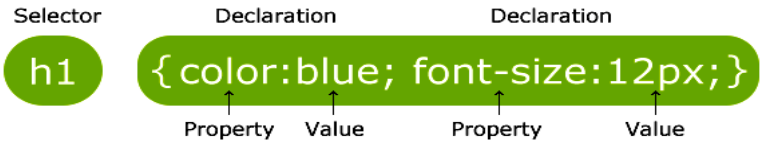
CSS stands for Cascading Style Sheets and provides HTML with layout and design. Along with making things pretty and aesthetically pleasing, CSS also provides a general structure to HTML.

Some of the most important CSS properties are:

- ♠ Color - specifying text color.
- ♠ Font-family - specifying font type.
- ♠ Font-size - specifying font size.
- ♠ Text-decoration - specifying text decorations, such as underline.
- ♠ Font-style - specifying font styling, such as italics.
- ♠ Font-weight - specifying font weight, such as bold.
- ♠ Width - specifying the width of an element.
- ♠ Height - specifying the height of an element.
- ♠ Background - specifying the background.
- ♠ Border - specifying a border.
- ♠ Text-shadow - specifying a shadow for our text.
- ♠ Float - specifying the float of an element, such as left or right.
- ♠ Position - specifying the position of an element, such as absolute or relative.
- ♠ Z-index - specifying the z-index of an element, such as 999; which would put that styled element 'on-top' of all other elements that either have a negative z-index specified or no z-index specified.
- ♠ Padding - specifying padding inside an element, such as padding around text.
- ♠ Margin - specifying the margin between elements.

CSS Syntax

A CSS rule has two main parts: a ***selector***, and one or more ***declarations***:



The selector is normally the HTML element you want to style.
Each declaration consists of a property and a value.
The property is the style attribute you want to change. Each property has a value.

Example:

CSS declarations always end with a semicolon, and declaration groups are surrounded by curly brackets:

```
<html> <head>
<style type="text/css">
P {
color:red;
text-align:center;
}
</style>
</head>
<body>
<p>Hello World!</p>
<p>This paragraph is styled with CSS.</p>
</body>
</html>
```

CSS Implementation

*CSS can be implemented in **three** different ways to our HTML:*

1. Inline
2. Internal
3. External

Using Inline CSS

So, let's use some inline CSS to change a few things in our HTML structure.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Inline CSS</title>
    </head>
    <body>
        <header>
            <h1 style="color:red">My Heading</h1>
        </header>
        <section>
            <article style="color:blue">
                My Article of Content
            </article>
        </section>
        <footer>
            <strong style="color:green">My Bold Text</strong>
            <br>
            <em style="font-weight:100">My Company</em>
        </footer>
    </body>
</html>
```

Output:

Color

As you have probably noticed, we have used the English word for the color that we want to use. But there are two other ways we can define colors in CSS; the rgb color values and something called Hexadecimal. All three types of defining colors in CSS are acceptable; you can read more about colors in CSS here:
http://www.w3schools.com/cssref/css_colors.asp.
We will be using a mixture of the three different ways to define colors in CSS.
CSS has several options for defining colors of both text and background areas on your pages. These options can entirely replace the color

attributes in plain HTML. In addition, you get new options that you just didn't have in plain HTML.

Value	Description
color	<color>
background-color	transparent <color>
background-image	none url(<URL>)
background-repeat	repeat repeat-x repeat-y no-repeat
background-attachment	Scroll fixed
background-position	<percentage> <length> top center bottom left right
background	<background-color> <background-image> <background-repeat> <background-attachment> <background-position>
color	<color>
background-color	transparent <color>
background-image	None url(<URL>)
background-repeat	repeat repeat-x repeat-y no-repeat

Example Program

```
<html><head>
<style type="text/css">
body{
background-color:#d0e4fe;}
h1
{
color:orange;
text-align:center;
}

p{
font-family:"Times New Roman";
font-size:20px;
}</style></head>
<body>
<h1>CSS example!</h1>
<p>This is a paragraph.</p>
</body></html>
```

Using Internal CSS

As you can see, our inline CSS is very effective, but perhaps not very efficient. Inline CSS is good for adding slight changes or specifying colors for different text elements, but it starts to get a little 'wordy' and messy.

When we add CSS to HTML either; externally or in the head section, we can use selectors.

Selectors allow us to 'select' or 'point' to a specific element of our HTML. CSS can use HTML elements as selectors, such as the paragraph, anchor, em and strong tags. If we referred to these elements as selectors in our CSS we would be styling every paragraph, anchor, em and strong element in our HTML.

Let's try the same thing, but this time adding our CSS to the head section of our HTML document and using selectors.

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Internal CSS</title>
        <style>
            h1{
                color: #ff0000;
            }
            strong{
                color: #00ff00;
            }
            em.bottom{
                font-weight: 100;
                font-size: 1.1em;
            }
        </style>
    </head>
    <body>
        <header>
            <h1>My Heading</h1>
        </header>
        <footer>
            <em>My Italic Text</em>
            <strong>My Bold Text</strong>
            <br>
            <em class="bottom">My Company</em>
        </footer>
    </body>
</html>
```

*I have added an additional **em** tag, to demonstrate using classes as selectors in CSS.*

Using Ids in CSS

As you may have guessed, we are using a class to identify our bottom em tag. The dot notation before the class name allows us to select or target an element in our HTML by its class name.

We can use ids like so:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>Internal CSS</title>
        <style>
            h1{
                color: #ff0000;
            }
            strong{
                color: #00ff00;
            }
            em#bottom{
                font-weight: 100;
                font-size: 1.1em;
            }
        </style>
    </head>
    <body>
        <header>
            <h1>My Heading</h1>
        </header>
        <footer>
            <em>My Italic Text</em>
            <strong>My Bold Text</strong>
            <br>
            <em id="bottom">My Company</em>
        </footer>
    </body>
</html>
```

All we have to do is change 'class' to 'id' for the element we are referring to, and change the '.' in front of our CSS selector (in the head element) to the '#' symbol.

The # symbol (when not used as ***href*** attribute) is generally used to signify an id within the HTML. The major different between using classes and id's is; classes can be re-used time and time again in the

same HTML document, whereas id's can only be used once in a single HTML document. You can think of a class as a group or multiple items, and an id as a single identification.
The output of the above code is the same (we have also set a font-size for the #bottom em tag) as the output for the previous code example, we are getting the same results as we are basically telling the browser the same thing, just in a different way.
There are several ways to make selectors 'unique' or point to only 'some' parts of the HTML.

A class is an effective way of referencing a specific part of our HTML; we can basically pinpoint the section of our code that contains the content we wish to style.
***Note**: When using em in CSS it's slightly different to the em tag in HTML. In HTML the em tag renders italic text. In CSS the em value can be used as a unit of measurement. A font size with a value greater than 1em will generate text larger than the default for that web page or User Agent, but does not render text as italic.*
Ids must be unique, we can only use the same id only once in our HTML page.
With classes, we can 'reuse' the class several times in our HTML page.

Creating External CSS
To add an external CSS to our HTML, we need to tell the HTML all about it- what relation it has to our HTML, the type of file it is and its location and name.

Remember the Meta tags from before?
Well this is implemented in the same way (completely different concepts), by adding a line of code into our head section of our HTML document. We use a rel value to tell HTML what the CSS file's relation is to the HTML, a type value to tell the HTML the type of file it is and a href value telling the HTML where the file is located and its name.

Note**: CSS files have a file extension of **.css
We can add an external style sheet to our HTML by using link tag.
So, let’s create a small CSS file, to use externally.

```
header{
    color: rgb(255,0,0);
    font-size: 2em;
    text-decoration: underline;
}
section article{
    font-weight: 200;
    font-size: 1.1em;
}
footer{
    font-size: 0.8em;
}
```

Go ahead and save the above styling into a new CSS file, titled style.css.

Linking to External CSS

If our style sheet (CSS) were located in the same directory (or folder) as our HTML file, we would add the tag to the head section of the HTML document, like so:

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>External CSS</title>
        <link rel="stylesheet" type="text/css" href="style.css">
    </head>
    <body>
        <header>
            <h1>My Heading</h1>
        </header>
        <section>
            <article>
                My Article of Content
            </article>
        </section>
        <footer>
            <em>My Footer</em>
        </footer>
    </body>
</html>
```

Just as our CSS example before, no matter of its location (inline, internal and external), the CSS will tell the browser to render the styles for our HTML in the same way.

Inefficient Selectors

Let's have a look at some inefficient CSS selectors with some external CSS:

```
header{
    color: red;
    font-size: 2em;
    text-decoration: underline;
}

section article{
    font-weight:200;
    font-size: 1.1em;
    color: red;
}

footer{
    font-size: 0.8em;
    font-weight: 200;
    color: red;
}
```

The advantages of using external CSS include the ability to completely separate the HTML from the CSS, to reduce individual file size and length, make things more readable and use effective selectors; meaning selectors that target multiple elements where necessary. Rather than having a large portion of CSS repeating and applying the same styling to different elements, we could 'join' the selectors together to create a smaller file size or to simply be more efficient with our use of CSS.

We can target or select multiple elements by separating the selectors with a comma in the CSS.

Efficient Selectors

```
header{
    font-size: 2em;
    text-decoration: underline;
}

section article{
    font-size: 1.1em;
}

footer{
    font-size: 0.8em;
}

header, section article, footer{
     color: red;
}

section article, footer{
    font-weight: 200;
}
```

HTML Element State

With our new CSS abilities, we are able to style a HTML element, based on its 'state'. HTML Element state refers to the 'state' that the elements are in; some of these include: Hover and Active.

You may have noticed that when you hover over a link on a web page, that the link will change color (among other aspects). We can do this with almost any HTML elements.

```
article:hover, article:hover a{
    background-color: #ff0000;
    color: rgb(255,255,255);
}
```

In the above example we have styled all 'hovered' over articles and the anchor tags, when the article is being 'hovered' over with a background of red and a text color of white.

Along with defining hover style, we can define active style. Active is defined by an element that is 'actively' being clicked on, i.e. if you are clicking a button, that button's state is now active.

```
article:active{
    background-color: #111111;
}
```

In the above example, an article that is 'active' will have a background color of almost black.

CSS Background

CSS background properties are used to define the background effects of an element.

CSS properties used for background effects are:

background-color
background-image
background-repeat
background-attachment
background-position

Background Color

The background-color property specifies the background color of an element.

```
body {background-color:#b0c4de;}
```

Background Image

The background-image property specifies an image to use as the background of an element. By default, the image is repeated so it covers the entire element.

```
body {background-image:url('paper.gif');
```

Background Image - Repeat Horizontally or Vertically

By default, the background-image property repeats an image both horizontally and vertically. Some images should be repeated only horizontally or vertically, or they will look strange, like this:

```
body
{
background-image:url('gradient2.png');
}
```

CSS Background Properties

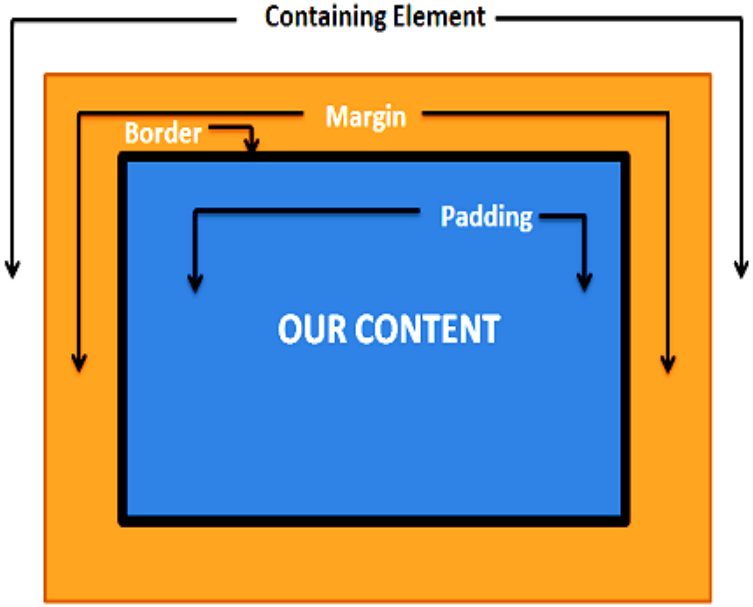
Property	Description	Values
background	Sets all the background properties in one declaration	*background-color* *background-image* *background-repeat* *background-attachment* *background-position* inherit
background-attachment	Sets whether a background image is fixed or scrolls with the rest of the page	scroll fixed inherit
background-color	Sets the background color of an element	*color-rgb* *color-hex* *color-name* transparent inherit
background-image	Sets the background image for an element	url(*URL*) none inherit
background-position	Sets the starting position of a background image	top left top center top right center left center center center right bottom left bottom center bottom right *x%* *y%* *xpos* *ypos* inherit

background-repeat	Sets if/how a background image will be repeated	repeat repeat-x repeat-y no-repeat inherit

CSS Box Model

One of the fundamental understandings of CSS is the Box Model. The Box model helps us to understand the layout and design of HTML and CSS.

The CSS Box model is made up of content, Padding, Borders and Margins.



So, what are Padding, Margins and Borders?

As you can see, padding is the space that surrounds our content; borders are what surround the padding and margins are what surround the borders.

By definition:

-The padding is the area that separates the content from the border.

-The border is the area that separates the padding from the margin.

-The margin is the area that separates our box from surrounding elements.

How do we define these?

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="UTF-8">
        <title>The Box Model</title>
        <style>
            section article{
                height: 50px;
                width: 500px;
                border: 1px solid #000000;
                padding: 50px 30px;
                margin: 0 auto;
            }
        </style>
    </head>
    <body>
        <section>
            <article>
                The CSS Box Model
            </article>
        </section>
    </body>
</html>
```

In the above example, we have set the padding for the top and bottom of the element to 50px and the left and right to 30px. The margin has been set to 0px for the top and bottom and the left and right margin is set to auto. When we set a value of auto in our margins, it will basically 'centre' the element within the containing or parent element.

As you may have noticed, we have also set our border. Our border is 1px wide for each side, is solid and has a color of black.

The above code renders the following output:

***Note** that the rendered output will be 152px in height (top and bottom padding, plus the width of our borders and the specified height of our element) and 562px in width (left and right padding, plus the width of our borders and the specified width of our element).*

Fonts

When using CSS we can change the font-family of our text. We can specify multiple font-families for any given element. If the user has the first specified font on their system; that is the font that will be used. If the user does not have our first specified font on their system, the browser will attempt to render the next font and so on until one of the fonts are located on the users system. These font-families are separated with a comma and the proceeding fonts are referred to as fallback fonts.

```
header{
    font-family: "Helvetica Neue", Helvetica, sans-serif;
}
article, footer{
    font-family: Arial, sans-serif;
}
```

In the above example we are saying that our header should have a font-family of Helvetica Neue, if that is not located on the users system, we will try Helvetica. If Helvetica is not located on the user's system, we will 'fall-back' to a generic sans-serif font.
You can find out more about sans-serif fonts here: http://en.wikipedia.org/wiki/Sansserif.

Text Color

The color property is used to set the color of the text. The color can be specified by:

name - a color name, like "red"
RGB - an RGB value, like "rgb(255,0,0)"
Hex - a hex value, like "#ff0000"

Example

```
body {color:blue;}
h1 {color:#00ff00;}
h2 {color:rgb(255,0,0);
```

Property	Description	Values
color	Sets the color of a text	*color*
direction	Sets the text direction	ltr rtl
line-height	Sets the distance between lines	normal *number* *length*
letter-spacing	Increase or decrease the space between characters	normal *length*
text-align	Aligns the text in an element	left right center justify
text-decoration	Adds decoration to text	none underline overline line-through blink
text-indent	Indents the first line of text in an element	*length* *%*
text-shadow		none *color* *length*
text-transform	Controls the letters in an element	none capitalize uppercase lowercase
unicode-bidi		normal embed bidi-override

vertical-align	Sets the vertical alignment of an element	baseline sub super top,text-top middle bottom text-bottom *length* *%*
white-space	Sets how white space inside an element is handled	normal pre,nowrap
word-spacing	Increase or decrease the space between words	normal *length*

Font Families

In CSS, there are two types of font family names:generic family - a group of font families with a similar look (like "Serif" or "Monospace") font family - a specific font family (like "Times New Roman" or "Arial").

Example

Property	**Description**	**Values**
font-family	Specifies the font family for text	*family-name* *generic-family* inherit
font-size	Specifies the font size of text	xx-small x-small small medium large x-large xx-large smaller larger *length* *%* inherit

font-style	Specifies the font style for text	normal italic oblique inherit
font-variant	Specifies whether or not a text should be displayed in a small-caps font	normal small-caps inherit
font-weight	Specifies the weight of a font	normal bold

Example Program

```
<html> <head>
<style type="text/css">
p.normal {font-weight:normal;}
p.light {font-weight:lighter;}
p.thick {font-weight:bold;}
p.thicker {font-weight:900;}
</style></head>
<body>
<p class="normal">This is a paragraph.</p>
<p class="light">This is a paragraph.</p>
<p class="thick">This is a paragraph.</p>
<p class="thicker">This is a paragraph.</p>
</body></html>
```

Quiz

1. Create a new HTML file and an external Style sheet.
2. Using all of the techniques, concepts and code that you have learned; create your first Website. This exercise is to get you started with HTML5 and CSS- in the real world!

WHAT's NEXT?

CHAPTER FOUR
JS

Introduction

JS (JavaScript) allows you to embed commands in an HTML page. When a compatible Web browser downloads the page, your JavaScript commands are loaded by the Web browser as a part of the HTML document. These commands can be triggered when the user clicks on page items, manipulates gadgets and fields in an HTML form, or moves through the page history list.

Some computer languages are *compiled*; you run your program through a compiler, which performs a one-time translation of the human-readable program into a binary that the computer can execute. JavaScript is an *interpreted* language; the computer must evaluate the program every time it's run. You embed your JavaScript commands within an HTML page, and any browser that supports JavaScript can interpret the commands and act on them.

JavaScript is powerful and simple. If you've ever programmed in BASIC or Visual Basic, you'll find JavaScript easy to pick up.

Scripting Language

HTML provides a good deal of flexibility to page authors, but HTML by itself is static; once written, HTML documents can't interact with the user other than by presenting hyperlinks. Creative use of CGI scripts (which run on Web servers) has made it possible to create more interesting and effective interactive sites, but some applications really demand programs or scripts that are executed by the client.

JavaScript allows Web authors to write small scripts that execute on the users' browsers instead of on the server. For example, an application that collects data from a form and then posts it to the server can validate the data for completeness and correctness before sending it to the server. This can greatly improve the performance of the browsing session since users don't have to send data to the server until it's been verified as correct.

Another important use of Web browser scripting languages like JavaScript comes as a result of the increased functionality being introduced for Web browsers in the form of Java applets, plug-ins, ActiveX Controls, and VRML objects and worlds. Each of these things can be used to add extra functions and interactivity to a Web page. Scripting languages act as the glue that binds everything together. A Web page might use an HTML form to get some user input and then set

a parameter for an ActiveX Control based on that input. It is a script that will usually actually carry this out.

History

JavaScript was developed in 10 days in May 1995 by Brendan Eich, then working at Netscape, as the HTML scripting language for their browser *Navigator 2*. Brendan Eich said (at the O'Reilly Fluent conference in San Francisco in April 2015): "*I did JavaScript in such a hurry; I never dreamed it would become the assembly language for the Web*".

JavaScript is a dynamic functional object-oriented programming language that can be used for:

1. Enriching a web page by
   - generating browser-specific HTML content or CSS styling,
   - inserting dynamic HTML content,
   - producing special audio-visual effects (animations).
2. Enriching a web user interface by
   - implementing advanced user interface components,
   - validating user input on the client side,
   - automatically pre-filling certain form fields.
3. Implementing a front-end web application with local or remote data storage, as described in the book Building Front-End Web Apps with Plain JavaScript [***http://web-engineering.info/JsFrontendApp-Book***].
4. Implementing a front-end component for a distributed web application with remote data storage managed by a back-end component, which is a server-side program that is traditionally written in a server-side language such as PHP, Java or C#, but can nowadays also be written in JavaScript with NodeJS.
5. Implementing a complete distributed web application where both the front-end and the back-end components are JavaScript programs. The version of JavaScript that is currently supported by web browsers is called "ECMAScript 5.1", or simply "ES5", but the next two versions, called "ES6" and "ES7" (or "ES 2015" and "ES 2016", as new versions are planned on a yearly basis), with lots of added functionality and improved syntaxes, are around the corner (and already partially supported by current browsers and back-end JS environments).

JavaScript

JavaScript provides a fairly complete set of built-in functions and commands, allowing you to perform math calculations, manipulate

strings, play sounds, open up new windows and new URLs, and access and verify user input to your Web forms.
Code to perform these actions can be embedded in a page and executed when the page is loaded. You can also write functions containing code that is triggered by events you specify. For example, you can write a JavaScript method that is called when the user clicks the Submit button of a form, or one that is activated when the user clicks a hyperlink on the active page.
JavaScript can also set the attributes, or *properties*, of ActiveX Controls, Java applets, and other objects present in the browser. This way, you can change the behavior of plug-ins or other objects without having to rewrite them. For example, your JavaScript code could automatically set the text of an ActiveX Label Control based on what time the page is viewed. JavaScript and VB Script are very similar, with similar syntax and capabilities.
However, JavaScript and VB Script are different languages and you should be careful not to mix them up when you are programming.

JavaScript Features
JavaScript commands are embedded in your HTML documents. Embedding JavaScript in your pages requires only one new HTML element: `<SCRIPT>` and `</SCRIPT>`. The `<SCRIPT>` element takes the attributes `LANGUAGE`, which specifies the scripting language to use when evaluating the script.
JavaScript itself resembles many other computer languages. If you're familiar with C, C++, Pascal, HyperTalk or Visual Basic, you'll recognize the similarities. If not, don't worry-the following are some simple rules that will help you understand how the language is structured:

- JavaScript is case-sensitive.
- JavaScript is pretty flexible about statements. A single statement can cover multiple lines and you can put multiple short statements on a single line-just make sure to add a semicolon at the end of each statement.
- Braces (the { and } characters) group statements into blocks; a *block* may be the body of a function or a section of code that gets executed in a loop or as part of a conditional test.

If you're a Java, C, or C++ programmer, you might be puzzled when looking at JavaScript programs-sometimes, each line ends with a semicolon, sometimes not. In JavaScript, unlike those other languages, the semicolon is not required at the end of each line.

JavaScript Instructions Conventions

Even though JavaScript is a simple language, it's quite expressive. Here, you learn a small number of simple rules and conventions that will ease your learning process and speed your use of JavaScript.

Hiding Your Scripts

You'll probably be designing pages that may be seen by browsers that don't support JavaScript. To keep those browsers from interpreting your JavaScript commands as HTML-and displaying them-wrap your scripts as follows:

```
<SCRIPT LANGUAGE="JavaScript">
<!-- This line opens an HTML comment
document.write("You can see this script's
output, but not its source.")
<!-- This line opens and closes a comment -->
</SCRIPT>
```

The opening <!-- comment causes Web browsers that do not support JavaScript to disregard all text they encounter until they find a matching -->, so they don't display your script. You do have to be careful with the <SCRIPT> tag, though; if you put your <SCRIPT> and </SCRIPT> block inside the comments, the Web browser will ignore them also.

Comments

Including comments in your programs to explain what they do is usually good practice-JavaScript is no exception. The JavaScript interpreter ignores any text marked as comments, so don't be shy about including them. You can use two types of comments: single-line and multiple-line.

Single-line comments start with two slashes (//), and they're limited to one line. Multiple-line comments must start with /* on the first line and end with */ on the last line. Here are a few examples:

```
   // this is a legal comment
/ illegal -- comments start with two slashes
/* Multiple-line comments can
   be spread across more than one line, as long
as they end. */
/* illegal -- this comment doesn't have an end!
/// this comment's OK, because extra slashes are
ignored //
```

Using <*NOSCRIPT*>

You can improve the compatibility of your JavaScript Web pages through the use of the <NOSCRIPT>...</NOSCRIPT> HTML tags. Any HTML code that is placed between these container tags will not appear on a JavaScript-compatible Web browser but will be displayed on one that is not able to understand JavaScript. This allows you to include alternative content for your users that are using Web browsers that don't understand JavaScript. At the very least, you can let them know that they are missing something, as in this example:

```
<NOSCRIPT>
<HR>If you are seeing this text, then your Web
browser
   doesn't speak JavaScript!<HR>
</NOSCRIPT>
```

JavaScript Language

JavaScript was designed to resemble Java, which in turn looks a lot like C and C++. The difference is that Java was built as a general-purpose object language, while JavaScript is intended to provide a quicker and simpler language for enhancing Web pages and servers. In this section, you learn the building blocks of JavaScript and how to combine them into legal JavaScript programs. JavaScript was developed by the Netscape Corporation, which maintains a great set of examples and documentation for it.

Identifiers

An *identifier* is just a unique name that JavaScript uses to identify a variable, method, or object in your program. As with other programming languages, JavaScript imposes some rules on what names you can use. All JavaScript names must start with a letter or the underscore character, and they can contain both upper- and lowercase letters and the digits 0 through 9.

JavaScript supports two different ways for you to represent values in your scripts: literals and variables. As their names imply, *literals* are fixed values that don't change while the script is executing, and *variables* hold data that can change at any time.

Literals and variables have several different types; the type is determined by the kind of data that the literal or variable contains. The following are some of the types supported in JavaScript:

- **Integers**: Integer literals are made up of a sequence of digits only; integer variables can contain any whole-number value. Octal

(base 8) and hexadecimal (base 16) integers can be specified by prefixing them with a leading "0" or "**0x**," respectively.

- **Floating-point numbers**: The number 10 is an integer, but 10.5 is a floating-point number. Floating-point literals can be positive or negative and they can contain either positive or negative exponents (which are indicated by an ***e*** in the number). For example, 3.14159265 is a floating-point literal, as is 6.023e23 (6.023[times] 10^{23} or Avogadro's number).
- **Strings**: Strings can represent words, phrases, or data, and they're set off by either double or single quotation marks. If you start a string with one type of quotation mark, you must close it with the same type. Special characters, such as \n and \t, can also be utilized in strings.
- **Booleans**: Boolean literals can have values of either TRUE or FALSE; other statements in the JavaScript language can return Boolean values.

Functions, Objects and Properties

JavaScript is modelled after Java, an object-oriented language. An *object* is a collection of data and functions that have been grouped together. A *function* is a piece of code that plays a sound, calculates an equation, or sends a piece of e-mail, and so on. The object's functions are called *methods* and its data are called its *properties*. The JavaScript programs you write will have properties and methods and will interact with objects provided by the Web browser, its plug-ins, Java applets, ActiveX Controls, and other things.

Though the terms function and method are often used interchangeably, they are not the same. A method is a function that is part of an object. For instance, **`writeln`** is one of the methods of the object document.

Built-In Objects and Functions

Individual JavaScript elements are ***objects***. For example, string literals are string objects and they have methods that you can use to change their case, and so on. JavaScript can also use the objects that represent the Web browser in which it is executing, the currently displayed page, and other elements of the browsing session.

You access objects by specifying their name. For example, the active document object is named document. To use document's properties or methods, you add a period and the name of the method or property you want. For example, **`document.title`** is the title property of the

document object, and **explorer.length** calls the length member of the string object named explorer. Remember, literals are objects, too.

Properties

Every object has properties, even literals. To access a property, just use the object name followed by a period and the property name. To get the length of a string object named address, you can write the following:

```
address.length
```

You get back an integer that equals the number of characters in the string. If the object you're using has properties that can be modified, you can change them in the same way. To set the color property of a house object, just use the following line:

```
house.color = "blue"
```

You can also create new properties for an object just by naming them. For example, say you define a class called customer for one of your pages. You can add new properties to the customer object as follows:

```
customer.name = "Joe Smith"
customer.address = "123 Elm Street"
customer.zip = "90210"
```

Finally, knowing that an object's methods are just properties is important. You can easily add new properties to an object by writing your own function and creating a new object property using your own function name. If you want to add a Bill method to your customer object, you can do so by writing a function named BillCustomer and setting the object's property as follows:

```
customer.Bill = BillCustomer;
```

To call the new method, you use the following:

```
customer.Bill()
```

Array and Object Properties

JavaScript objects store their properties in an internal table that you can access in two ways. You've already seen the first way-just use the properties' names. The second way, *arrays*, allows you to access all of an object's properties in sequence. The following function prints out all the properties of the specified object:

```
function DumpProperties(obj, obj_name) {
   result = ""     // set the result string to
blank
   for (i in obj)
      result += obj_name + "." + i + " = " +
obj[i] + "\n"
```

```
return result
}
```

So not only can you access all of the properties of the document object, for instance, by property name, using the dot operator (for example, **`document.href`**), you can also use the objects property array (for example, **`document[1],`** though this may not be the same property as **`document.href`**). JavaScript provides another method of array access that combines the two, known as *associative arrays*. An associative array associates a left- and right-side element, and the value of the right side can be used by specifying the value of the left side as the index. Objects are set up by JavaScript as associative arrays with the property names as the left side, and their values as the right. So the **`href`** property of the document object could be accessed using **`document["href"].`**

Programming with JavaScript

JavaScript has a lot to offer page authors. It's not as flexible as C or C++, but it's quick and simple. Most importantly, it's easily embedded in your WWW pages so that you can maximize their impact with a little JavaScript seasoning. This section covers a detailed explanation of the language's features.

Expressions

An *expression* is anything that can be evaluated to get a single value. Expressions can contain string or numeric literals, variables, operators, and other expressions, and they can range from simple to quite complex. For example, the following are expressions that use the assignment operator to assign numerical or string values to variables:

```
x = 7;
str = "Hello, World!";
```

By contrast, the following is a more complex expression whose final value depends on the values of the quitFlag and formComplete variables:

```
(quitFlag == TRUE) & (formComplete == FALSE)
```

Operators

Operators do just what their name suggests: they operate on variables or literals. The items that an operator acts on are called its *operands.* Operators come in the two following types:

- **Unary operators**: These operators require only one operand and the operator can come before or after the operand. The -- operator, which

subtracts one from the operand, is a good example. Both --count and count-- subtract one from the variable count.

- **Binary operators**: These operators need two operands. The four math operators (+ for addition, - for subtraction, * for multiplication, and / for division) are all binary operators, as is the = assignment operator you saw earlier.

Assignment Operators

Assignment operators take the result of an expression and assign it to a variable. JavaScript doesn't allow you to assign the result of an expression to a literal. One feature of JavaScript that is not found in most other programming languages is that you can change a variable's type on the fly. Consider the HTML document shown in Listing below.

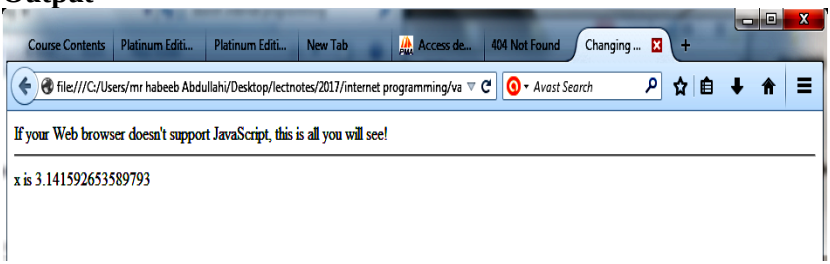
***Var-fly.htm**-JavaScript Allows You to Change the Data Type of Variables*

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- Hide this script from incompatible Web
browsers!
function typedemo() {
   var x;
   document.writeln("<HR>");
   x = Math.PI;
   document.writeln("x is " + x + "<BR>");
   x = FALSE;
   document.writeln("x is " + x + "<BR>");
   document.writeln("<HR>");
}
<!-- -->
</SCRIPT>
<TITLE>Changing Data Types on the Fly!</TITLE>
</HEAD>
<BODY BGCOLOR=#FFFFFF>
If your Web browser doesn't support JavaScript,
this is all you will see!
<SCRIPT LANGUAGE="JavaScript">
<!-- Hide this script from incompatible Web
browsers!
typedemo();
<!-- -->
```

```
</SCRIPT>
</BODY>
</HTML>
```

This short program first prints the (correct) value of pi in the variable x. In most other languages, though, trying to set a floating-point variable to a Boolean value would either generate a compiler error or a runtime error. JavaScript happily accepts the change and print x's new value: FALSE (see output).

Output



Because JavaScript variables are loosely typed, not only their value can be changed, but also their data type.

Operator	What It Does	Two Equivalent Expressions
+=	Adds two values	x+=y and x=x+y
	Adds two strings	string += "HTML" and string = string + "HTML"
-=	Subtracts two values	x-=y and x=x-y
*=	Multiples two values	a*=b and a=a*b
/=	Divides two values	e/=b and e=e/b

***Table** 4.1: Assignment Operators that Provide Shortcuts to Doing Assignments and Math Operations at the Same Time*

The most common assignment operator, =, simply assigns the value of an expression's right side to its left side. In the previous example, the variable x got the integer value 7 after the expression was evaluated. For convenience, JavaScript also defines some other operators that combine common math operations with assignment; they're shown in Table 4.1.

Math Operators

The preceding sections gave you a sneak preview of the math operators that JavaScript furnishes. You can either combine math operations with assignments, as shown in Table 4.1, or use them individually. As you would expect, the standard four math functions (addition, subtraction, multiplication, and division) work just as they do on an ordinary calculator. The negation operator, -, is a unary operator that negates the sign of its operand. Another useful binary math operator is the modulus operator, %. This operator returns the remainder after the integer division of two integer numbers. For instance, in the expression

```
x = 13%5;
```

the variable x would be given the value of 3.

JavaScript also adds two useful unary operators: -- and ++, called, respectively, the *decrement* and *increment* operators. These two operators modify the value of their operand, and they return the new value. They also share a unique property: they can be used either before or after their operand. If you put the operator after the operand, JavaScript returns the operand's value and then modifies it. If you take the opposite route and put the operator before the operand, JavaScript modifies it and returns the modified value. The following short example might help clarify this seemingly odd behavior:

```
x = 7;   // set x to 7
a = --x; // set x to x-1, and return the new
x; a = 6
b = a++; // set b to a, so b = 6, then add 1
to a; a = 7
x++;     // add one to x; ignore the returned
value
```

Comparison Operators

Comparing the value of two expressions to see whether one is larger, smaller, or equal to another is often necessary. JavaScript supplies several comparison operators that take two operands and return TRUE if the comparison is TRUE, and FALSE if it's not. (Remember, you can use literals, variables, or expressions with operators that require expressions.) Table 4.2 shows the JavaScript comparison operators.

***Table 4**.2: Comparison Operators That Allow Two JavaScript Operands to Be Compared in a Variety of Ways*

Operator	Read It As	Returns TRUE When
==	Equals	The two operands are equal
!=	Does not equal	The two operands are unequal
<	Less than	The left operand is less than the right operand
<=	Less than or	The left operand is
	equal to	less than or equal to
		the right operand
>	Greater than	The left operand is greater than the right operand
>=	Greater than or	The left operand is
	equal to	greater than or equal
		to the right operand

Thinking of the comparison operators as questions may be helpful. When you write the following:

```
(x >= 10)
```

you're really saying, "Is the value of variable x greater than or equal to 10?" The return value answers the question, TRUE or FALSE.

Logical Operators

Comparison operators compare quantity or content for numeric and string expressions, but sometimes you need to test a logical value, like whether a comparison operator returns TRUE or FALSE. JavaScript's logical operators allow you to compare expressions that return logical values. The following are JavaScript's logical operators:

- &&, read as "and." The && operator returns TRUE if both its input expressions are TRUE. If the first operand evaluates to FALSE, && returns FALSE immediately, without evaluating the second operand. Here's an example:

```
x = TRUE && TRUE;       // x is TRUE
x = FALSE && FALSE;     // x is FALSE
x = FALSE && TRUE;      // x is FALSE
```

- ||, read as "or." This operator returns TRUE if either of its operands is TRUE. If the first operand is TRUE, || returns TRUE without evaluating the second operand. Here's an example:

```
x = TRUE || TRUE;       // x is TRUE
x = FALSE || TRUE;      // x is TRUE
x = FALSE || FALSE;     // x is FALSE
```

- !, read as "not." This operator takes only one expression, and it returns the opposite of that expression, so !TRUE returns FALSE, and !FALSE returns TRUE.

Note that the "and" and "or" operators don't evaluate the second operand if the first operand provides enough information for the operator to return a value. This process, called *short-circuit evaluation*, can be significant when the second operand is a function call. For example,

```
keepGoing = (userCancelled == FALSE) &&
(theForm.Submit())
```

If userCancelled is TRUE, the second operand, which submits the active form, isn't called.

String Operators

A few of the operators that were listed above can be used for string manipulation as well. All of the comparison operators can be used on strings, too; the results depend on standard lexicographic ordering, but comparisons aren't case-sensitive. Additionally, the + operator can also be used to concatenate strings. The expression

```
str = "Hello, " + "World!";
```

assigns the resulting string Hello, World! to the variable **str**.

JavaScript Control Structures

Some scripts are plain; they'll execute the same way every time, once per page. For example, if you add a JavaScript to play a sound when users visit your home page, it doesn't need to evaluate any conditions or do anything more than once. More sophisticated scripts might require that you take different actions under different circumstances. You might also want to repeat the execution of a block of code-perhaps by a set number of times, or as long as some condition is TRUE. JavaScript provides constructs for controlling the execution flow of your script based on conditions, as well as repeating a sequence of operations.

Testing Conditions

JavaScript provides a single type of control statement for making decisions: the **`if...else`** statement. To make a decision, you supply an expression that evaluates to TRUE or FALSE; which code is executed depends on what your expression evaluates to.

The simplest form of **`if...else`** uses only the if part. If the specified condition is TRUE, the code following the condition is executed; if not, it's skipped. For example, in the following code fragment, the message appears only if the condition (that the **`lastModified.year`** property of the document object says it was modified before 1995) is TRUE:

```
if (document.lastModified.year < 1995)
   document.write("Danger! This is a mighty old
document.")
```

You can use any expression as the condition. Since expressions can be nested and combined with the logical operators, your tests can be pretty sophisticated. For example:

```
if ((document.lastModified.year >= 1995) &&
(document.lastModified.month >= 10))
   document.write("This document is reasonably
current.")
```

The else clause allows you to specify a set of statements to execute when the condition is FALSE, for instance

```
if ((document.lastModified.year >= 1995) &&
(document.lastModified.month >= 10))
   document.write("This document is reasonably
current.")
else
   document.write("This document is quite old.")
```

Repeating Actions

JavaScript provides two different loop constructs that you can use to repeat a set of operations. The first, called a **`for loop`**, executes a set of statements some number of times. You specify three expressions: an *initial* expression that sets the values of any variables you need to use, a *condition* that tells the loop how to see when it's done, and an *increment* expression that modifies any variables that need it. Here's a simple example:

```
for (count=0; count < 100; count++)
```

```
   document.write("Count is ", count);
```

This loop executes 100 times and prints out a number each time. The initial expression sets the counter, count, to zero. The condition tests to see whether count is less than 100 and the increment expression increments count.

You can use several statements for any of these expressions, as follows:

```
for (count=0, numFound = 0; (count < 100) &&
(numFound < 3); count++)
   if (someObject.found()) numFound++;
```

This loop either loops 100 times or as many times as it takes to "find" three items-the loop condition terminates when count >= 100 or when numFound >= 3.

The second form of loop is the while loop. It executes statements as long as its condition is TRUE. For example, you can rewrite the first for loop in the preceding example as follows:

```
count = 0
while (count < 100) {
   if (someObject.found()) numFound++;
   document.write("Count is ", count)
}
```

Which form you use depends on what you're doing; for loops are useful when you want to perform an action a set number of times, and while loops are best when you want to keep doing something as long as a particular condition remains TRUE. Notice that by using curly braces, you can include more than one command to be executed by the while loop (this is also TRUE of for loops and if...else constructs).

Reserved Words

JavaScript reserves some keywords for its own use. You cannot define your own methods or properties with the same name as any of these keywords; if you do, the JavaScript interpreter complains.

JavaScript's reserved keywords are shown in Table 4.3.

***Table** 4.3: JavaScript Reserved Keywords Should Not Be Used in Your JavaScripts*

abstract	double	instanceof	super
boolean	else	int	switch
break	extends	interface	synchronized
byte	FALSE	long	this

case	final	native	throw
catch	finally	new	throws
char	float	null	transient
class	for	package	TRUE
const	function	private	try
continue	goto	protected	var
default	if	public	void
do import static	implements short	return with	while

Because JavaScript is still being developed and refined, the list of reserved keywords might change or grow over time. Whenever a new version of JavaScript is released, it might be a good idea to look over its new capabilities with an eye towards conflicts with your JavaScript programs.

Other JavaScript Statements

This section provides a quick reference to some of the other JavaScript commands. The commands are listed in alphabetical order-many have examples. Here's what the formatting of these entries mean:

-All JavaScript keywords are in italics font.

- Words in italics represent user-defined names or statements.
- Any portions enclosed in square brackets ([and]) are optional.
- {statements} indicates a block of statements, which can consist of a single statement or multiple statements enclosed by curly braces.

***break* statement**

The break statement terminates the current while or for loop and transfers program control to the statement following the terminated loop.

Syntax

```
break
```

Example

The following function scans the list of URLs in the current document

and stops when it has seen all URLs or when it finds a URL that matches the input parameter searchName:

```
function findURL(searchName) {
   var i = 0;
   for (i=0; i < document.links.length; i++) {
      if (document.links[i] == searchName) {
         document.writeln(document.links[i]  +
"<br>")
         break;
      }
   }
}
```

***continue* statement**

The continue statement stops executing the statements in a while or for loop, and skips to the next iteration of the loop. It doesn't stop the loop altogether like the break statement; instead, in a while loop, it jumps back to the condition, and in a for loop, it jumps to the update expression.

Syntax

```
continue
```

Example

The following function prints the odd numbers between 1 and x; it has a continue statement that goes to the next iteration when i is even:

```
function printOddNumbers(x) {
   var i = 0
   while (i < x) {
      i++;
      if ((i % 2) == 0) // the % operator
divides & returns the remainder
         continue
      else
         document.write(i, "\n")
   }
}
```

***for* loop**

A for loop consists of three optional expressions, enclosed in parentheses and separated by semicolons, followed by a block of statements executed in the loop. These parts do the following:

- The starting expression, initial_expr, is evaluated before the loop starts. It is most often used to initialize loop counter variables, and you're free to use the var keyword here to declare new variables.
- A condition is evaluated on each pass through the loop. If the condition evaluates to TRUE, the statements in the loop body are executed. You can leave the condition out, and it always evaluates to TRUE. If you do so, make sure to use break in your loop when it's time to exit.
- An update expression, update_expr, is usually used to update or increment the counter variable or other variables used in the condition. This expression is optional; you can update variables as needed within the body of the loop if you prefer.
- A block of statements are executed as long as the condition is TRUE. This block can have one or multiple statements in it.

Syntax

```
for ([initial_expr;] [condition;]
[update_expr]) {
   statements
}
```

Example

This simple for statement prints out the numbers from 0 to 9. It starts by declaring a loop counter variable, i, and initializing it to zero. As long as i is less than 9, the update expression increments i, and the statements in the loop body are executed.

```
for (var i = 0; i <= 9; i++) {
   document.write(i);
}
```

for...in loop

The for...in loop is a special form of the for loop that iterates the variable variable-name over all the properties of the object named object-name. For each distinct property, it executes the statements in the loop body.

Syntax

```
for (var in obj) {
   statements
}
```

Example

The following function takes as its arguments an object and the object's name. It then uses the for...in loop to iterate through all the object's properties, and writes them into the current Web page.

```
function dump_props(obj,obj_name) {
   for (i in obj)
      document.writeln(obj_name + "." + i
+ " = " + obj[i] + "<br>");
}
```

function statement

The function statement declares a JavaScript function; the function may optionally accept one or more parameters. To return a value, the function must have a return statement that specifies the value to return. All parameters are passed to functions *by value*-the function gets the value of the parameter but cannot change the original value in the caller.

Syntax

```
function name([param] [, param] [..., 
param]) {
   statements
}
```

Example

```
function PageNameMatches(theString) {
   return (document.title == theString)
}
```

if...else statement

The if...else statement is a conditional statement that executes the statements in block1 if condition is TRUE. In the optional else clause, it executes the statements in block2 if condition is FALSE. The blocks of statements can contain any JavaScript statements, including further nested if statements.

Syntax

```
if (condition) {
    statements
}
[else {
    statements}]
```

Example

```
if (Message.IsEncrypted()) {
    Message.Decrypt(SecretKey);
```

```
}
else {
    Message.Display();
}
```

***new* statement**

The new statement is the way that new objects are created in JavaScript. For instance, if you defined the following function to create a house object

```
function house (rms,stl,yr,garp) { // define a
house object
   this.room = rms;         // number of rooms
(integer)
   this.style = stl;       // style (string)
   this.yearBuilt  =  yr;       //  year  built
(integer)
   this.hasGarage  =  garp;  //  has  garage?
(boolean)
}
```

you could then create an instance of a house object using the new statement, as in the following:

```
var myhouse = new
house(3,"Tenement",1962,false);
```

A few notes about this example. First, note that the function used to create the object doesn't actually return a value. The reason it is able to work is that it makes use of the this object, which always refers to the current object. Second, while the function defines how to create the house object, none is actually created until the function is called using the new statement.

***return* statement**

The return statement specifies the value to be returned by a function.

Syntax

```
return expression;
```

Example

The following simple function returns the square of its argument, x, where x is any number.

```
function square( x ) {
   return x * x;
}
```

this statement

You use this to access methods or properties of an object within the object's methods. The this statement always refers to the current object.

Syntax

```
this.property
```

Example

If setSize is a method of the document object, then this refers to the specific object whose setSize method is called:

```
function setSize(x,y) {
   this.horizSize = x;
   this.vertSize = y;
}
```

This method sets the size for an object when called as follows:

```
document.setSize(640,480);
```

var statement

The var statement declares a variable varname, optionally initializing it to have value. The variable name varname can be any JavaScript identifier, and value can be any legal expression (including literals).

Syntax

```
var varname [= value] [, var varname [= value] ] [..., var varname [= value] ]
```

Example

```
var num_hits = 0, var cust_no = 0;
```

while statement

The while statement contains a condition and a block of statements. The while statement evaluates the condition; if condition is TRUE, it executes the statements in the loop body. It then re-evaluates condition and continues to execute the statement block as long as condition is TRUE. When condition evaluates to FALSE, execution continues with the next statement following the block.

Syntax

```
while (condition) {
   statements
}
```

Example

The following simple while loop iterates until it finds a form in the current document object whose name is "OrderForm", or until it runs out of forms in the document:

```
x = 0;
while ((x < document.forms[].length) &&
(document.forms[x].name
[ic:ccc]!= "OrderForm")) {
   x++
}
```

***with* statement**

The with statement establishes object as the default object for the statements in block. Any property references without an object are then assumed to be for object.

Syntax

```
with object {
   statements
}
```

Example

```
with document {
   write "Inside a with block, you don't need to
specify the object.";
   bgColor = gray;
}
```

JavaScript and Web Browsers

The most important thing you will be doing with your JavaScripts is interacting with the content and information on your Web pages, and through it, with your user. JavaScript interacts with your Web browser through the browsers object model. Different aspects of the Web browser exist as different objects, with properties and methods that can be accessed by JavaScript. For instance, `document.write()` uses the write method of the document object. Understanding this Web browser object model is crucial to using JavaScript effectively. Also, understanding how the Web browser processes and executes your scripts is also necessary.

Scripts Execution

When you put JavaScript code in a page, the Web browser evaluates the code as soon as it's encountered. Functions, however, don't get executed when they're evaluated; they just get stored for later use. You still have to call functions explicitly to make them work. Some functions are attached to objects, like buttons or text fields on forms, and they are called when some event happens on the button or field. You might also have functions that you want to execute during page evaluation. You can do so by putting a call to the function at the appropriate place in the page.

Where to Put Your Scripts

You can put scripts anywhere within your HTML page, as long as they're surrounded with the <SCRIPT>...</SCRIPT> tags. One good system is to put functions that will be executed more than once into the <HEAD> element of their pages; this element provides a convenient storage place. Since the <HEAD> element is at the beginning of the file, functions and VB Script code that you put there will be evaluated before the rest of the document is loaded. Then you can execute the function at the appropriate point in your Web page by calling it, as in the following:

```
<SCRIPT language="JavaScript">
<!-- Hide this script from incompatible
Web browsers!
myFunction();
<!-- -->
</SCRIPT>
```

Another way to execute scripts is to attach them to HTML elements that support scripts. When scripts are matched with events attached to these elements, the script is executed when the event occurs. This can be done with HTML elements, such as forms, buttons, or links. Consider code below, which shows a very simple example of attaching a JavaScript function to the `onClick` attribute of a HTML forms button (see Figure 4.1).

Code: *Button1.htm-Calling a JavaScript Function with the Click of a Button*

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
```

```
<!-- Hide this script from incompatible
Web browsers!
function pressed() {
    alert("I said Don't Press Me!");
}
<!-- -->
</SCRIPT>
<TITLE>JavaScripts Attached to HTML
Elements</TITLE>
</HEAD>
<BODY BGCOLOR=#ffffff>
<FORM NAME="Form1">
    <INPUT TYPE="button" NAME="Button1"
VALUE="Don't Press Me!"
        onClick="pressed()">
</FORM>
</BODY>
</HTML>
```

JavaScript also provides you with an alternate way to attach functions to objects and their events. For simple actions, you can attach the JavaScript directly to the attribute of the HTML form element, as shown in code below. Each of these listings will produce the output shown in Figure 4.1 above.

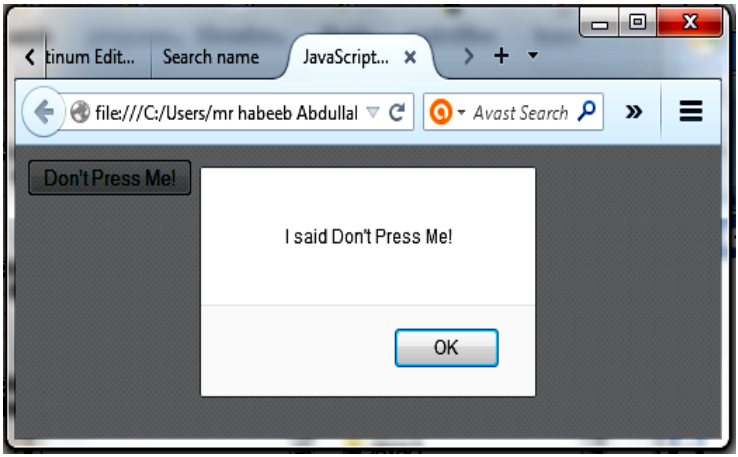


Figure 4.1

JavaScript functions can be attached to form fields through several different methods.

***Code**: Button2.htm-Simple VB Scripts Can Be Attached Right to a Form Element*

```
<HTML>
<HEAD>
<TITLE>JavaScripts Attached to HTML
Elements</TITLE>
</HEAD>
<BODY BGCOLOR=#FFFFFF>
<FORM NAME="Form1">
    <INPUT TYPE="button" NAME="Button1"
VALUE="Don't Press Me!"
        onClick="alert('I said Don\'t
Press Me!')">
</FORM>
</BODY>
</HTML>
```

Sometimes, though, you have code that shouldn't be evaluated or executed until after all the page's HTML has been parsed and displayed. An example would be a function to print out all the URLs referenced in the page. If this function is evaluated before all the HTML on the page has been loaded, it misses some URLs, so the call to the function should come at the page's end. The function itself can be defined anywhere in the HTML document; it is the function call that should be at the end of the page.

JavaScript code to modify the actual HTML contents of a document (as opposed to merely changing the text in a form text input field, for instance) must be executed during page evaluation.

JavaScript Applications

A common example scripting application generally involves interaction with HTML forms in order to perform client-side validation of the forms data before submission. JavaScript can perform this function very well.

Manipulating Windows

This example shows how it is possible to create an HTML forms-based control panel that uses JavaScript to load and execute other JavaScripts in their own windows. This is done through the use of the window Web browser object, its properties and methods.

The next **Code** shows the "main program", the top-level HTML document giving access to the control panel (see Figure 4.2).

The JavaScript in this example is very simple, and is included in the onClick attribute of the forms <input> tag. Clicking on the button executes the JavaScript window method open:

```
window.open('cp.htm','ControlPanel','width=300
,height=250')
```

This creates a window named "ControlPanel" that is 300[ts]250 pixels in size, and loads the HTML document Cp.htm.

Code: *Cpmain.htm-A JavaScript Attached Right to a forms Button Will Create a New Window when Clicked*

```
<HTML>
<HEAD>
<TITLE>JavaScript Window Example</TITLE>
</HEAD>
<BODY BGCOLOR=#FFFFFF>
<CENTER><H3>Activate the control panel by
clicking below</H3></CENTER>
<HR>
<FORM>
<CENTER>
<TABLE>
<TR><TD><INPUT                    TYPE="button"
NAME="ControlButton" VALUE="Control Panel"

onClick="window.open('cp.htm','ControlPanel',

  'width=300,height=250')"></TD></TR>
  </TABLE>
  </CENTER>
  </FORM>
  </BODY>
  </HTML>
```

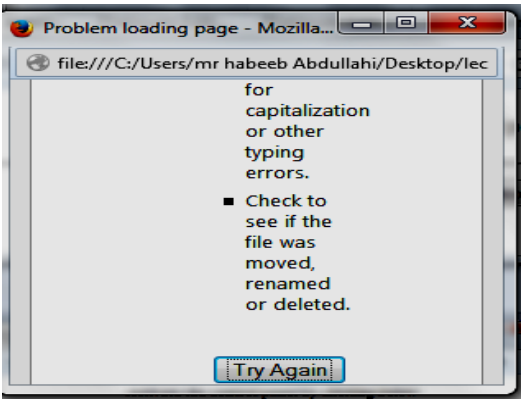


Figure 4.2

The Control Panel button calls a JavaScript and creates a new browser window.

When the button is clicked, Cp.htm is loaded into its own window, as shown in Figure 4.2 (note that in this figure and the next, the windows have been manually rearranged so that they all can be seen). This HTML document uses an interface of an HTML form organized in a table to give access through this control panel to other JavaScript applications, namely a timer and a real-time clock. The next ***Code*** shows Cp.htm. The JavaScript functions openTimer(), openClock(), closeTimer(), and closeClock() are used to open and close windows for a JavaScript timer and clock, respectively. These functions are attached to forms buttons that make up the control panel. Note that JavaScript variables timerw and clockw because they are defined outside of any of the functions, they can be used anywhere in the JavaScript document. They are used to remember whether or not the timer and clock windows are opened.

Code*: Cp.htm-This HTML form Calls JavaScripts to Create and Destroy Windows for a Timer and or a Real-Time Clock*

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- Hide this script from incompatible Web
browsers!
var timerw = null;
var clockw = null;
function openTimer() {
   if(!timerw)
```

```
 timerw                                                =
open("cptimer.htm","TimerWindow","width=300,he
ight=100");
}
function openClock() {
   if(!clockw)
      clockw                                          =
open("cpclock.htm","ClockWindow","width=50,hei
ght=25");
}
function closeTimer() {
   if(timerw) {
      timerw.close();
      timerw = null;
   }
}
function closeClock() {
   if(clockw) {
      clockw.close();
      clockw = null;
   }
}
<!-- -->
</SCRIPT>
</HEAD>
<BODY BGCOLOR=#EEEEEE>
<FORM>
<CENTER>
<TABLE>
<TR><TD>To Open Timer...</TD>
    <TD ALIGN=CENTER>
       <INPUT                          TYPE="button"
NAME="ControlButton" VALUE="Click Here!"
          onClick="openTimer()"></TD></TR>
<TR><TD>To Close Timer...</TD>
    <TD ALIGN=CENTER>
       <INPUT                          TYPE="button"
NAME="ControlButton" VALUE="Click Here!"
          onClick="closeTimer()"></TD></TR>
<TR><TD>To Open Clock...</TD>
    <TD ALIGN=CENTER>
```

```
 <INPUT TYPE="button" NAME="ControlButton"
VALUE="Click Here!"
          onClick="openClock()"></TD></TR>
<TR><TD>To Close Clock...</TD>
    <TD ALIGN=CENTER>
       <INPUT TYPE="button"
NAME="ControlButton" VALUE="Click Here!"
          onClick="closeClock()"></TD></TR>
<TR><TD>To Open Both...</TD>
    <TD ALIGN=CENTER>
       <INPUT TYPE="button"
NAME="ControlButton" VALUE="Click Here!"
onClick="openTimer();openClock();"></TD></TR>
<TR><TD>To Close Both...</TD>
    <TD ALIGN=CENTER>
       <INPUT TYPE="button"
NAME="ControlButton" VALUE="Click Here!"
onClick="closeTimer();closeClock();"></TD></TR
>
<TR><TD></TD></TR>
<TR><TD>To Close Everything...</TD>
    <TD ALIGN=CENTER>
       <INPUT TYPE="button"
NAME="ControlButton" VALUE="Click Here!"
onClick="closeTimer();closeClock();self.close(
);"></TD></TR>
</TABLE></CENTER></FORM></BODY></HTML>
```

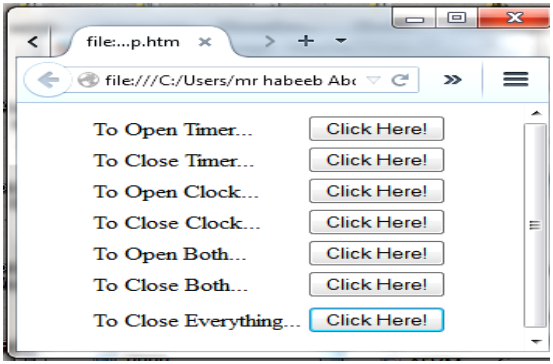


Figure 4.3

JavaScript can create new Web browser windows, with definable widths and heights.

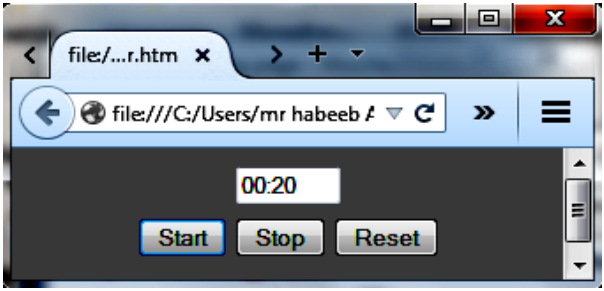
Codes below show cptimer.htm and cpclock.htm, the HTML documents to implement the JavaScript timer and real-time clock. Note that each uses the properties of the JavaScript Date object to access time information. Figure 4.4 shows the Web page with the control panel, timer, and real-time clock windows all open.

***Code**: Cptimer.htm-The JavaScript Date Object Can Be Used to Keep Track of Relative Time*

```
<HTML>
<HEAD>
<SCRIPT LANGUAGE="JavaScript">
<!-- Hide this script from incompatible
Web browsers!
var timerID = 0;
var tStart  = null;
function UpdateTimer() {
   if(timerID) {
      clearTimeout(timerID);
      clockID  = 0;
   }
   if(!tStart)
      tStart   = new Date();
   var tDate = new Date();
   var tDiff = tDate.getTime() -
tStart.getTime();
   var str;
   tDate.setTime(tDiff);
   str = ""
   if (tDate.getMinutes() < 10)
      str += "0" + tDate.getMinutes() +
":";
   else
      str += tDate.getMinutes() + ":";
   if (tDate.getSeconds() < 10)
      str += "0" + tDate.getSeconds();
   else
      str += tDate.getSeconds();
   document.theTimer.theTime.value = str;
```

```
  timerID = setTimeout("UpdateTimer()",
1000);
}
function Start() {
   tStart = new Date();
   document.theTimer.theTime.value =
"00:00";
   timerID = setTimeout("UpdateTimer()",
1000);
}
function Stop() {
   if(timerID) {
      clearTimeout(timerID);
      timerID  = 0;
   }
   tStart = null;
}
function Reset() {
   tStart = null;
   document.theTimer.theTime.value =
"00:00";
}
<!-- -->
</SCRIPT>
</HEAD>
<BODY BGCOLOR=#AAAAAA
onload="Reset();Start()"
onunload="Stop()">
<FORM NAME="theTimer">
<CENTER>
<TABLE>
<TR><TD COLSPAN=3 ALIGN=CENTER>
       <INPUT TYPE=TEXT NAME="theTime"
SIZE=5></TD></TR>
<TR><TD></TD></TR>
<TR><TD><INPUT TYPE=BUTTON NAME="start"
VALUE="Start"
            onclick="Start()"></TD>
    <TD><INPUT TYPE=BUTTON NAME="stop"
VALUE="Stop"
            onclick="Stop()"></TD>
```

```
 <TD><INPUT TYPE=BUTTON NAME="reset"
VALUE="Reset"
            onclick="Reset()"></TD>
    </TR>
</TABLE>
</CENTER>
</FORM>
</BODY>
</HTML>
```



***Code**: Cpclock.htm The Date Object Can Also Be Used to Access the Real-Time Clock of the Client System*

```
<HTML>
<HEAD>
<TITLE>Clock</TITLE>
<SCRIPT LANGUAGE="JavaScript">
<!-- Hide this script from incompatible
Web browsers!
var clockID = 0;
function UpdateClock() {
   if(clockID) {
      clearTimeout(clockID);
      clockID  = 0;
   }
   var tDate = new Date();
   var str;
   str = "";
   if (tDate.getHours() < 10)
      str += "0" + tDate.getHours() + ":";
   else
      str += tDate.getHours() + ":";
   if (tDate.getMinutes() < 10)
```

```
      str += "0" + tDate.getMinutes() +
":";
   else
      str += tDate.getMinutes() + ":";
   if (tDate.getSeconds() < 10)
      str += "0" + tDate.getSeconds();
   else
      str += tDate.getSeconds();
   document.theClock.theTime.value = str;
   clockID = setTimeout("UpdateClock()",
1000);
}
function StartClock() {
   clockID = setTimeout("UpdateClock()",
500);
}
function KillClock() {
   if(clockID) {
      clearTimeout(clockID);
      clockID  = 0;
   }
}
<!-- -->
</SCRIPT>
</HEAD>
<BODY BGCOLOR=#CCCCCC
onload="StartClock()"
onunload="KillClock()">
<CENTER>
<FORM NAME="theClock">
   <INPUT TYPE=TEXT NAME="theTime" SIZE=8>
</FORM>
</CENTER>
</BODY>
</HTML>
```

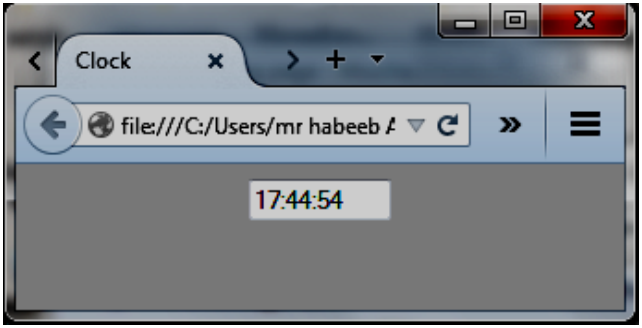


Figure *4.4*

Multiple browser windows can be created by JavaScript, each running its own JavaScripts and performing its functions independently.

For More JS Apps See Appendix A!!!

WHAT's NEXT?

REFERENCES

Andrew Glassner, Computer Graphics, Microsoft Research, Olin Lathrop, Cognivision, Inc.

Andrew H. Watt, Jinjer L. Simon, Jonathan Watt: *Teach Yourself JavaScript in 21 Days,* Pearson Education, ISBN 0672322978. *Copyright 2015.*

Angel, Edward, 2012, Interactive computer graphics: a top-down approach with shader-based OpenGL /Edward Angel, David Shreiner.—6th ed.p. cm. ISBN-13: 978-0-13-254523-5 (alk. paper)

Animation, https://www.brownbagfilms.com/labs/entry/flash-animation-tutorial-part-1

***Ashley Menhennett, Pablo Farias Navarro,* 'A Guide to HTML5 and CSS3' *Copyright* ZENVA 2014.**

Chris Bates, "Web Programming, Building Internet applications", 2nd edition, WILEY Dreamtech, *Copyright 2013.*

Daniel Schuller 2011, C# Game Programming: For Serious Game Creation, Course Technology, a part of Cengage Learning 20 Channel Center Street Boston.

Danny Goodman, Scott Markel, "*JavaScript and DHTML", Cookbook,* O'Reilly & Associates, ISBN 0596004672. *Copyright 2015.*

Gerd Wagner, "JavaScript Front-End Web App Tutorial" e-book Retrieved February 2016.

***Jim O'Donnell, Web Design Book* –JavaScript e-book Retrieved 2012.**

JS history: http://en.wikipedia.org/wiki/Brendan_Eich.

JS history: http://www.w3.org/community/webed/wiki/A_Short_History_of_JavaScript.

JS https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number/isInteger.

JSON: https://developer.mozilla.org/en/JavaScript/Reference/Global_Objects/JSON.

K.Dharmarajan M.Sc, M.Phil., Asst Professsor, Web Technology Notes. Vels University, Chennai-117.

Michael Bailey, Introduction to Computer Graphics, Course Notes for SIGGRAPH '99, Course Organizer University of California at San Diego, and San Diego Supercomputer Center.

Norman I. Badler , Computer Animation Techniques, Platinum Edition, e-book.

Russell Chun, Adobe Flash Professional CC Classroom in a Book (2014 release) 2015 Adobe Systems Incorporated and its licensors. All rights reserved.

Steve Bark, An Introduction to Adobe Photoshop Copyright 2012 ISBN 978-87-403-0016-1

APPENDIX A
JS Applications

PULL DOWN MENU

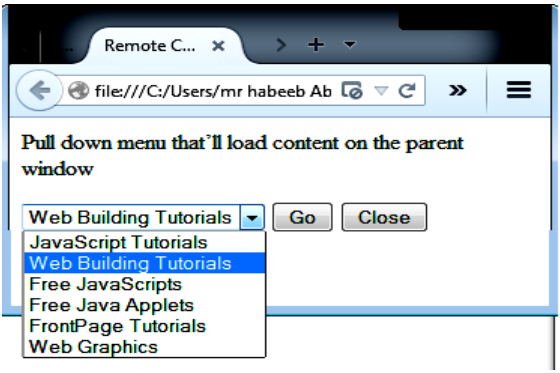
Pull down menu that'll load content on the parent window

```
<script>
<!--
var
win3=window.open("remote2.htm","","width=300,hei
ght=30,resizable")
win3.creator=self
//-->
</script>
```

```
<html>
<head>
<title>Remote Control</title>
</head>
<script>
<!--
function gogo()
{
creator.location=document.combo.go.options[docum
ent.combo.go.selectedIndex].value
}
//-->
</script>
<body bgcolor="#FFFFFF">
```

```
<form name="combo">
  <p><select name="go" size="1"
onChange="gogo()">
    <option selected
value="../javaindex.htm">JavaScript
Tutorials</option>
    <option value="../howto/webbuild.htm">Web
Building Tutorials</option>
 <option value="../cutpastejava.htm">Free
JavaScripts</option>
```

```
  <option value="../java/javafront.htm">Free
Java Applets</option>
    <option value="../frontpage.htm">FrontPage
Tutorials</option>
    <option value="../backgr.htm">Web
Graphics</option>
  </select> <input type="button" value="Go"
onClick="gogo()"> <input type="button"
  value="Close" onClick="window.close()"> </p>
</form>
</body>
</html>
```



INSTALL INFORMATION VALIDATION

```
<!-- TWO STEPS TO INSTALL INFORMATION
VALIDATION:

1.  Paste the coding into the HEAD of your HTML
document
2.  Put the last code into the BODY of your HTML
document  -->

<!--  STEP ONE: Copy this code into the HEAD of
your HTML document  -->

<HEAD>

<SCRIPT LANGUAGE="JavaScript">
<!-- This script and many more are available
online from -->
<!-- The JavaScript Source!!
http://javascriptsource.com -->
```

```
<!-- Begin
function validate(){
var digits="0123456789"
var temp
if (document.testform.Name.value=="") {
alert("No Name !")
return false
}
if (document.testform.age.value=="") {
alert("Invalid Age !")
return false
}
for (var
i=0;i<document.testform.age.value.length;i++){
temp=document.testform.age.value.substring(i,i+1
)
if (digits.indexOf(temp)==-1){
alert("Invalid Age !")
return false
      }
   }
return true
}
// End -->
</SCRIPT>
<!--  STEP TWO: Copy this code into the BODY of
your HTML document  -->
<BODY>
<FORM name="testform" onSubmit="return
validate()">
Name:<input type="text" size=30 name="Name">
Age:<input type="text" size=3 name="age">
<input type="submit" value="Submit">
</FORM>
```

MULTIPLE USERS LOGIN

```
<!-- TWO STEPS TO INSTALL MULTIPLE USERS:
   1.  Copy the first code into the HEAD of your
HTML document
```

```
  2.  Put the last coding into the BODY of your
HTML document  -->

<!-- STEP ONE: Copy this code into the HEAD of
your login HTML document  -->

<HEAD>

<SCRIPT LANGUAGE="JavaScript">
<!--Total Java Scripts 99 - Next Step Software--
>

<!-- Begin
function Login(){
var done=0;
var username=document.login.username.value;
username=username.toLowerCase();
var password=document.login.password.value;
password=password.toLowerCase();
if (username=="millenium" &&
password=="millenium") {
window.location="page1.htm"; done=1; }
if (username=="member2" &&
password=="password2") {
window.location="page2.htm"; done=1; }
if (username=="member3" &&
password=="password3") {
window.location="page3.htm"; done=1; }
if (done==0) { alert("Invalid login!"); }
}
// End -->
</SCRIPT>

<!-- STEP TWO: Paste this code into the BODY of
your HTML document  -->

<BODY>
<center>
<form name=login>
<table width=225 border=1 cellpadding=3>
<tr><td colspan=2><center><font
size="+2"><b>Members-Only
Area!</b></font></center></td></tr>
<tr><td>Username:</td><td><input name=username
></td></tr>
```

```
<tr><td>Password:</td><td><input name=password
></td></tr>
<tr><td colspan=2 align=middle><input
type=button value="Login!"
onClick="Login()"></td></tr>
</table>
</form>
</center>
<P>
<!-- Script Size:  1.60 KB  --></P>
<P>Username: millenium</P>
<P><FONT style="BACKGROUND-COLOR:
#ffffff">Password: millenium
</FONT></P>
```

THREE TRIES LOGIN

```
<!-- TWO STEPS TO INSTALL THREE TRIES:

   1.  Put the first code into the BODY of your
HTML document
   2.  Change protectedpage.html to your
protected filename  -->

<!-- STEP ONE: Copy this code into the BODY of
your HTML document  -->

<BODY>
<SCRIPT LANGUAGE="JavaScript">

<!-- This script and many more are available
online from -->
<!-- The JavaScript Source!!
http://javascriptsource.com -->

<!-- Begin
function password() {
var testV = 1;
var pass1 = prompt('Please Enter Your
Password','');
while (testV < 3) {
if (!pass1)
history.go(-1);
if (pass1 == "password") {
```

```
alert('You Got it Right!');

//  Change the following URL to your protected
filename

window.open('protectedpage.html');
break;
}
testV+=1;
var pass1 =
prompt('Access Denied - Password Incorrect,
Please Try Again.','Password');
}
if (pass1!="password" & testV ==3)
history.go(-1);
return " ";
}
document.write(password());
</SCRIPT>
<CENTER>
<FORM>
<input type="button" value="Enter Password
Protected Area" onClick="password()">
</FORM>
</CENTER>
```

BLOCK IP ADDRESS FROM YOUR PAGE

```
<html>
<body bgcolor=ffffff>
<script language="javascript">
var ip = '24.0.217.223'
if (ip == '206.186.23.178') {
alert("STOP! You are viewing this page from an
IP address that is not allowed!");
alert("Why can't you guys just leave me
alone?");
if (confirm("Do you want to leave peacefully? Or
will I have to help you?"))
{location.href="http://www.mydesktop.com" } else
{ ("OK you choose I don't care! Bye bye! Don't
```

```
come back!");
{location.href="http://www.mydesktop.com" }} }
</SCRIPT>
</body>
</html>
```

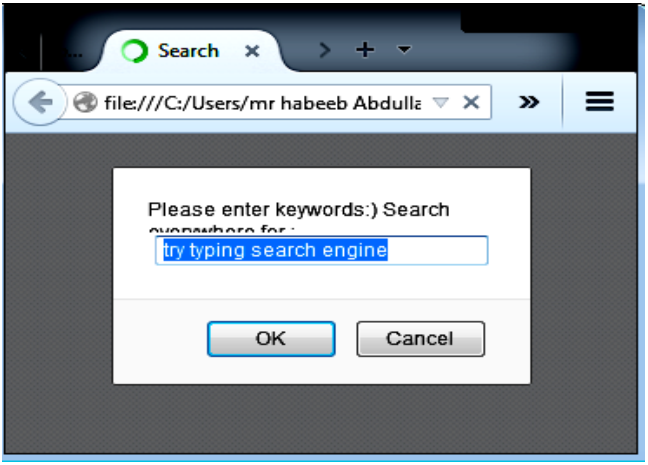
SEARCH ENGINE

```
<script language="javascript">
       var key = "";
        function makeEntry (){
                this.Date = "";
                this.Name="";
                this.URL = "";
                this.Desc = "";
                this.Category = "";
                return this;
        }
 function makeArray(n) {
                this.length = n;
           for (var k = 1; k <= n; k++) {
                           this[k] = "";
                }
                return this;
        }
        function makeLinks(size) {
                      this.length = size;
                  for (var r=1; r<= size; r++)
{
this[r] = new makeEntry();
this[r].Date = datesArray[r];
this[r].Name = namesArray[r];
this[r].URL = urlsArray[r];
this[r].Desc = descArray[r];
}
           return this;
                        }
var linksize=0
datesArray = new makeArray(linksize);
```

```
namesArray = new makeArray(linksize);
urlsArray = new makeArray(linksize);
descArray = new makeArray(linksize);

var arraycount=0

arraycount += 1
datesArray[arraycount] = "1/1/97 "
urlsArray[arraycount] = "http://www.yahoo.com"
namesArray[arraycount] = "Yahoo"
descArray[arraycount] = "An excellent search
engine available free on the web"
//alert(arraycount)

arraycount += 1
datesArray[arraycount] = "1/1/97 "
urlsArray[arraycount] = "http://www.lycos.com"
namesArray[arraycount] = "Lycos"
descArray[arraycount] = "An extensive search
engine, great alternative to Yahoo"
//alert(arraycount)

arraycount += 1
datesArray[arraycount] = "1/1/97 "
urlsArray[arraycount] =
"http://www.webcrawler.com"
namesArray[arraycount] = "Webcrawler"
descArray[arraycount] = "A great search engine
from the makers of AOL"
//alert(arraycount)

arraycount += 1
datesArray[arraycount] = "1/1/97 "
urlsArray[arraycount] = "http://www.search.com"
namesArray[arraycount] = "Search.com"
descArray[arraycount] = "A collection of
hundreds of search engines; from Yahoo to a
seach engine which looks up phone numbers."
//alert(arraycount)

arraycount += 1
datesArray[arraycount] = "1/1/97 "
urlsArray[arraycount] =
"http://altavista.digital.com"
namesArray[arraycount] = "AltaVista"
```

```
descArray[arraycount] = "This search engine has
the largest database of websites of all search
engines on the web"
//alert(arraycount)
linksize = arraycount;
// ----end data -------
function showAll(linkobj) {
   for (var s=1; s<= linkobj.length; s++) {
showLink(linkobj,s);
                                    }
                              }
function showLink (links, index) {
        //document.write("<table border>");
        document.write("<tr><td>" +
links[index].Date +"</td>");
        document.write("<td><a href=" +
links[index].URL +">" + links[index].Name +
"</a></td>");
        document.write("<td>" +
links[index].Desc + "</td></tr>");
        //document.write("</table>");
        }
function searchLinks(links, keyword){
    document.write("Search results for keyword:"
+keyword +"<br>");
        document.write("<table border>");
        for (var q=1; q<=links.length; q++) {
                  //document.write(q+".")
      if (links[q].URL.indexOf(keyword) != -1){
                        //
document.write("Search Results for keyword:
"+keyword+ "<br>" +links[q].Name +"<p>")
                        //document.write("Search
Results for keyword: "+keyword+ "<br>" );

showLink(links,q);
                       continue;
                        }
     if (links[q].Desc.indexOf(keyword) != -1) {
  showLink(links,q);
```

```
   continue;
                          }
  if (links[q].Date.indexOf(keyword) != -1) {
showLink(links,q);
                       continue;
                          }
     if (links[q].Name.indexOf(keyword) != -1) {
showLink(links,q);
                continue;
                          }
                  }
         document.write("</table>");
         }
// final stuff
// the main program ---

jsi = new makeLinks(linksize);
document.write("<title>Search</title><body
bgcolor=white>");

searchLinks(jsi, prompt("Please enter keywords:)
\rSearch everywhere for :","try typing search
engine"));
document.write("<hr>");
document.write("This searches all areas (Date,
Name, URL, and Description) for matches ");
document.write("and returns a list of hits.  The
keyword is case sensitive. This script can be
easily modified to fit your needs. ");
document.write("Click <b>search again</b> for
another search. <hr>");
document.write("<form><input type=button
onClick='history.go(0)' value='Search
Again'></form>");

// show all the links
//document.write("<table border>");
//showAll(jsi);
//document.write("</table>");
</script>
```



SHOPPING CART

Shopping Cart Instructions

Edited JShop by MH7 2018 is built around six HTML files as follows:
bindex.htm - sets up the two frames needed for the shopping basket
bmenu.htm - menu of options for the shopping basket (bottom frame)
bitem.htm - a page containing some dummy items (more about this in a minute).
bbasket.htm - page which displays the contents of the basket etc.
bbuy.htm - first stage of ordering, this then should go to another page on a secure server for entering credit card details etc.
bfinish.htm - the final ordering page which would normally reside on a secure server for the user to enter their credit card details.
The index file sets up two frames, the bottom one called 'menu' (which should be left as menu in order for the JavaScript routines to work) and a top one called 'main'. At the moment index.htm is set up to load bmenu.htm into the bottom frame (which should not be changed) and bitem.htm into the top frame (this can be changed if you wish).
bitem.htm calls routines for adding items to the shopping basket from bmenu.htm. Whatever way you set up your items on these pages the function you need to call in order to add items to the shopping basket is `buyItem()`. It can either be used as a link:

```
<A HREF="javascript:top.menu.buyItem(product,
price,quantity)"> Buy</A>
```

or as an ***onClick*** statement on a form button:

```
<INPUT TYPE=BUTTON
onClick="javascript:top.menu.buyItem(prod,price,
quant)">
```

You'll notice in the code that quantities are picked up from a form box. For instance, the quantity box for the first item is called `cbtquant`. So all you have to do is pick up the value in that box when you call `buyItem`().

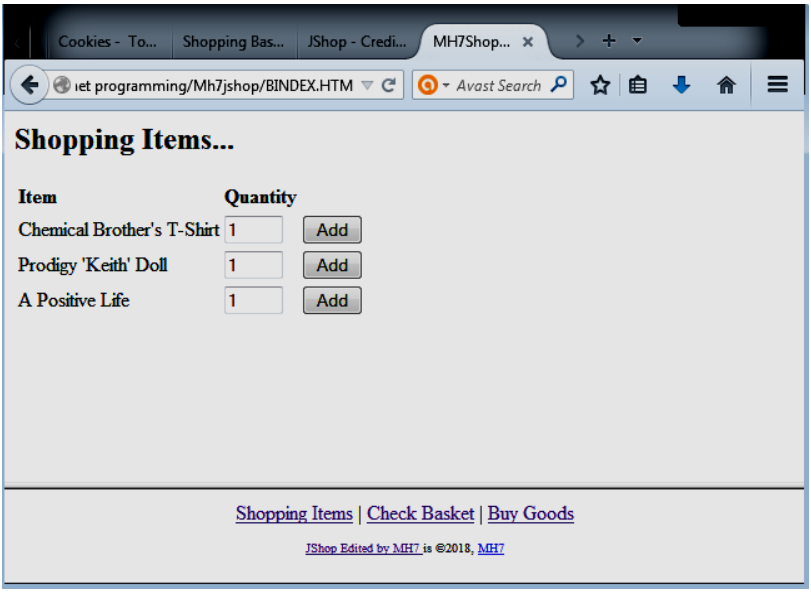
```
top.menu.buyItem('Chemical Brothers T-Shirt',
'15.95', document.itemsform.cbtquant.value)
```

Finally, in ***bbuy.htm*** and ***bfinish.htm*** you'll have to amend the FORM tag in order to point to your correct form mailing script. Contact your web space provider for more details if you're not sure or contact us and we'll try and help you out.

And that's really all there is to it. There is some commenting in the code to help you understand what's happening.

BINDEX.htm

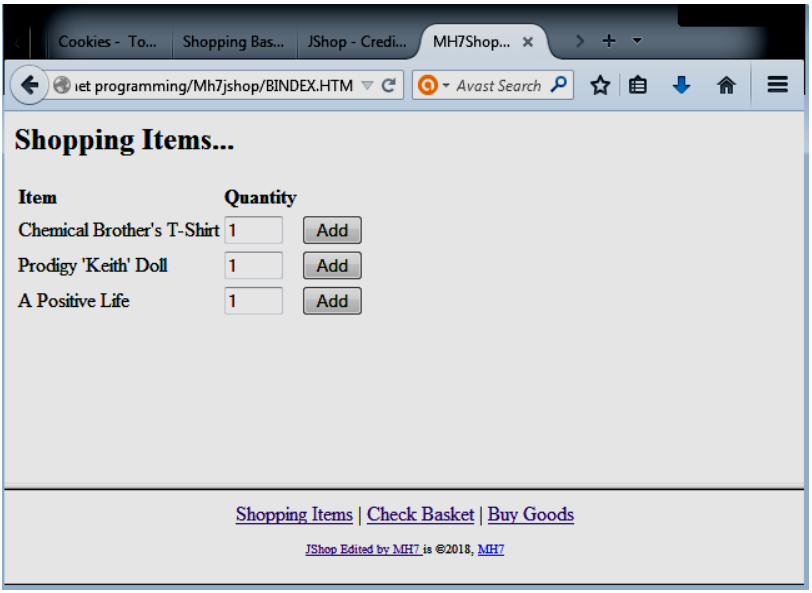
```
<HTML>
<!-- JShop Edited by MH7 2018                -->
<!-- by MH7 @ (www.habeebmamman.com)      -->
<!-- FRAME SETUP                             -->
<!-- bottom frame is called 'menu'           -->
<!-- top frame is called 'main'              -->
<HEAD>
<TITLE>MH7Shop (January 2018)</TITLE>
</HEAD>
<SCRIPT LANGUAGE="JavaScript">
</SCRIPT>
<frameset rows="80%,20%">
<frame src="bitem.htm" name="main">
<frame src="bmenu.htm" name="menu" scrolling=no>
</frameset>
<BODY>
</BODY>
</HTML>
```



BITEM.htm

```
<HTML>
<!-- JShop Edited by MH7 2018                -->
<!-- by MH7 @ (www.habeebmamman.com)      -->
<!-- ITEM PAGE
-->
<!-- you can have as many item pages as you
like, simply   -->
<!-- use the same methods to call the buyItem
function      -->
<!-- NOTE: buyItem function is in bmenu.htm and
is called  -->
<!-- by referencing the frame it's in,     -->
<!-- e.g. top.menu.buyitem()             -->
<HEAD>
<TITLE>Shopping Items</TITLE>
</HEAD>
<BODY>
<h2>Shopping Items...</h2>
<p>
<FORM NAME="itemsform">
<!-- Put items in a table -->
```

```
<table>
<tr><th
align=left>Item</th><th>Quantity</th></tr>
<tr><td>Chemical Brother's T-Shirt</td><td>
<!-- Display quantity box -->
<INPUT TYPE="value" NAME="cbtquant" VALUE="1"
SIZE=3>
</td><td>
<!-- Display button that calls buyItem() -->
<INPUT TYPE="button" NAME="cbtadd" VALUE="Add"
onclick="top.menu.buyItem('Chemical Brothers T-
Shirt','0.45',
document.itemsform.cbtquant.value)">
</td></tr>
<!-- Same as above for any other items, just
remember to -->
<!-- change the input names etc.      -->
<tr><td>Prodigy 'Keith' Doll</td><td>
<INPUT TYPE="value" NAME="pkdquant" VALUE="1"
SIZE=3>
</td><td>
<INPUT TYPE="button" NAME="pkdadd" VALUE="Add"
onclick="top.menu.buyItem('Prodigy Keith
Doll','24.44',
document.itemsform.pkdquant.value)">
</td></tr>
<tr><td>A Positive Life</td><td>
<INPUT TYPE="value" NAME="aplquant" VALUE="1"
SIZE=3>
</td><td>
<INPUT TYPE="button" NAME="apladd" VALUE="Add"
onclick="top.menu.buyItem('A Positive Life',
'1000.45', document.itemsform.aplquant.value)">
</td></tr>
</TABLE>
</BODY>
</HTML>
```



BMENU.htm

```
<HTML>
<!-- JShop Edited by MH7 2018                    -->
<!-- by MH7 @ (www.habeebmamman.com)          -->
<!-- BASKET MENU
-->
<HEAD>
<TITLE>Shopping Menu</TITLE>
</HEAD>

<BODY>
<center>
<SCRIPT LANGUAGE="JavaScript">
      function alterError(value) {
            if (value<=0.99) {
                  newPounds = '0';
            } else {
                  newPounds = parseInt(value);
            }
newPence = parseInt((value+.0008 - newPounds)*
100);
            if (eval(newPence) <= 9)
newPence='0'+newPence;
```

```
newString = newPounds + '.' + newPence;
            return (newString);
      }

      // buyItem - adds an item to the shooping
basket
      function buyItem(newItem, newPrice,
newQuantity) {
            if (newQuantity <= 0) {
                  rc = alert('The quantity
entered is incorrect');
                  return false;
            }
if (confirm('Add '+newQuantity+' x '+newItem+'
to basket')) {
index = document.cookie.indexOf("TheBasket");
                  countbegin =
(document.cookie.indexOf("=", index) + 1);
 countend = document.cookie.indexOf(";", index);
                  if (countend == -1) {
              countend = document.cookie.length;
                  }
document.cookie="TheBasket="+document.cookie.sub
string(countbegin,
countend)+"["+newItem+","+newPrice+"#"+newQuanti
ty+"]";
            }
            return true;
      }

      //resetShoppingBasket - resets to shopping
basket to empty
      function resetShoppingBasket() {
index = document.cookie.indexOf("TheBasket");
            document.cookie="TheBasket=.";
      }
</SCRIPT>
<a href="bitem.htm" target="main">Shopping
Items</a> |
<a href="bbasket.htm" target="main">Check
Basket</a> |
```

```
<a href="bbuy.htm" target="main">Buy
Goods</a><br>
<font size=-2><br><a
href="http://www.habeebmamman.com/">JShop Edited
by MH7 </a> is ©2018, <a href="http://www.
habeebmamman.com/" target="_top">MH7</a></font>
</form>
<script
language="JavaScript">resetShoppingBasket()</SCR
IPT>
</center>
</BODY>
</HTML>
```

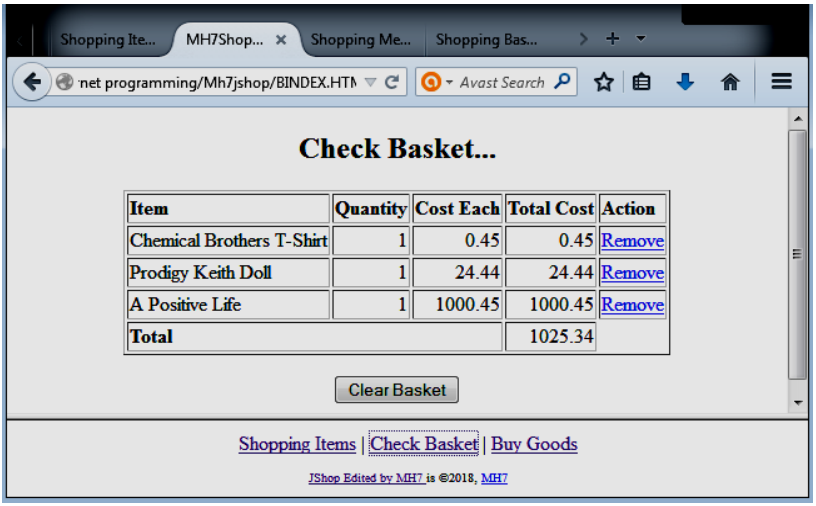
BBASKET.htm

```
<HTML>
<!-- JShop Edited by MH7 2018                -->
<!-- by MH7 @ (www.habeebmamman.com)      -->
<!-- SHOPPING BASKET
-->
<HEAD>
<TITLE>Shopping Basket</TITLE>
</HEAD>
<SCRIPT LANGUAGE="JavaScript">

      // showItems() - displays shopping basket
in a table
      function showItems() {
            index =
document.cookie.indexOf("TheBasket");
            countbegin =
(document.cookie.indexOf("=", index) + 1);
            countend =
document.cookie.indexOf(";", index);
            if (countend == -1) {
            countend = document.cookie.length;
```

```
}
            fulllist =
document.cookie.substring(countbegin, countend);
            totprice = 0;
            document.writeln('<TABLE BORDER>');
document.writeln('<TR><TD><b>Item</b></TD><TD><b
>Quantity</b></TD><TD><b>Cost
Each</b></TD><td><b>Total
Cost</b><TD><b>Action</b></TD></TR>');
            itemlist = 0;
            for (var i = 0; i <= fulllist.length;
i++) {
      if (fulllist.substring(i,i+1) == '[') {
                        itemstart = i+1;
                  } else if
(fulllist.substring(i,i+1) == ']') {
                        itemend = i;
                        thequantity =
fulllist.substring(itemstart, itemend);
                        itemtotal = 0;
      itemtotal = (eval(theprice*thequantity));
            temptotal = itemtotal * 100;
            totprice = totprice + itemtotal;
                        itemlist=itemlist+1;
document.writeln('<tr><td>'+theitem+'</td><td
align=right>'+thequantity+'</td><td
align=right>'+theprice+'</td><td
align=right>'+top.menu.alterError(itemtotal)+'</
td><td><a
href="javascript:removeItem('+itemlist+')">Remov
e</a></td></tr>');
                  } else if
(fulllist.substring(i,i+1) == ',') {
theitem = fulllist.substring(itemstart, i);
                        itemstart = i+1;
                  } else if
(fulllist.substring(i,i+1) == '#') {
theprice = fulllist.substring(itemstart, i);
                        itemstart = i+1;
                  }
            }
```

```
document.writeln('<tr><td colspan=3><b>Total</b>
</td><td
align=right>'+top.menu.alterError(totprice)+'</t
d><td></td></tr>');
            document.writeln('</TABLE>');
      }
      function removeItem(itemno) {
            newItemList = null;
            itemlist = 0;
            for (var i = 0; i <= fulllist.length;
i++) {
      if (fulllist.substring(i,i+1) == '[') {
                        itemstart = i+1;
                  } else if
(fulllist.substring(i,i+1) == ']') {
                        itemend = i;
                        theitem =
fulllist.substring(itemstart, itemend);
                        itemlist=itemlist+1;
                        if (itemlist != itemno) {
                              newItemList =
newItemList+'['+fulllist.substring(itemstart,
itemend)+']';
                        }
                  }
            }
index = document.cookie.indexOf("TheBasket");
      document.cookie="TheBasket="+newItemList;
      top.frames[0].location = "bbasket.htm";
      }
// clearBasket() - removes all items from the
basket
      function clearBasket() {
            if (confirm('Are you sure you wish to
clear the basket')) {
index = document.cookie.indexOf("TheBasket");
                  document.cookie="TheBasket=.";
      top.frames[0].location = "bbasket.htm";
            }
      }
</SCRIPT>
```

```
<BODY>
<center>
<h2>Check Basket...</h2>
<p>

<!-- all these next few line do is call
showItems() which -->
<!-- just creates a table - do what you like
with the      -->
<!-- presentation around it!            -->

<SCRIPT LANGUAGE="JavaScript">
      showItems();
</SCRIPT>
<p>
<form>
<input type="button" name="clear" value="Clear
Basket" onClick="clearBasket()">
</form>
</center>
</BODY>
</HTML>
```



BBUY.htm

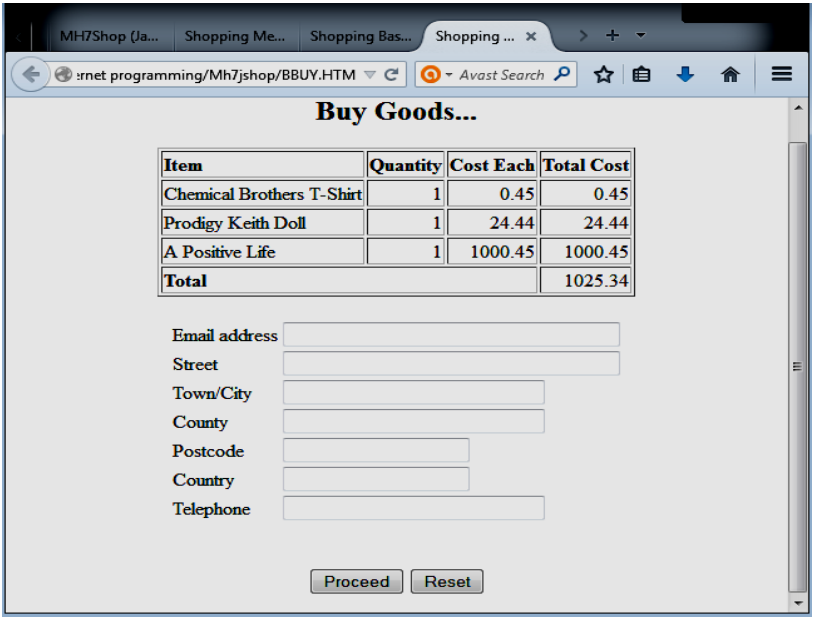
```
<HTML>
<!-- JShop Edited by MH7 2018            -->
<!-- by MH7 @ (www.habeebmamman.com)     -->
```

```
<!-- BUY GOODS                                    -->
<HEAD>
<TITLE>Shopping Buy</TITLE>
</HEAD>
<BODY>
<SCRIPT LANGUAGE="JavaScript">
      function alterError(value) {
            if (value<=0.99) {
                  newPounds = '0';
            } else {
                  newPounds = parseInt(value);
            }
newPence = parseInt((value+.0008 - newPounds)*
100);
      if (eval(newPence) <= 9)
newPence='0'+newPence;
      newString = newPounds + '.' + newPence;
            return (newString);
      }
      // showItems () - creates a table of items
in the basket and
      // creates the start of a form which sets
information for
      // basket items.
      function showItems() {
index = document.cookie.indexOf("TheBasket");
      countbegin = (document.cookie.indexOf("=",
index) + 1);
       countend = document.cookie.indexOf(";",
index);
            if (countend == -1) {
          countend = document.cookie.length;
            }
            fulllist =
document.cookie.substring(countbegin, countend);
            totprice = 0;
            document.writeln('<FORM
action="bfinish.htm" target="_top">');
      document.writeln('<TABLE BORDER COLS=4>');
```

```
document.writeln('<TR><TD><b>Item</b></TD><TD><b
>Quantity</b></TD><TD><b>Cost
Each</b></TD><td><b>Total Cost</b></TR>');
           itemlist = 0;
     for (var i = 0; i <= fulllist.length; i++)
{
     if (fulllist.substring(i,i+1) == '[') {
                       itemstart = i+1;
                  } else if
(fulllist.substring(i,i+1) == ']') {
                       itemend = i;
     thequantity =
fulllist.substring(itemstart, itemend);
                       itemtotal = 0;
     itemtotal = (eval(theprice*thequantity));
     temptotal = itemtotal * 100;
          totprice = totprice + itemtotal;
               itemlist=itemlist+1;
document.writeln('<tr><td>'+theitem+'</td><td
align=right>'+thequantity+'</td><td
align=right>'+theprice+'</td><td
align=right>'+alterError(itemtotal)+'</td></tr>'
);
          document.writeln('<INPUT
TYPE="hidden" NAME="item'+itemlist+'"
VALUE="'+theitem+'" SIZE="40">');
     document.writeln('<INPUT TYPE="hidden"
NAME="quantity'+itemlist+'"
VALUE="'+thequantity+'" SIZE="40">');
          document.writeln('<INPUT
TYPE="hidden" NAME="price each'+itemlist+'"
VALUE="'+theprice+'" SIZE="40">');
     document.writeln('<INPUT TYPE="hidden"
NAME="total cost'+itemlist+'"
VALUE="'+alterError(itemtotal)+'" SIZE="40">');
                  } else if
(fulllist.substring(i,i+1) == ',') {
theitem = fulllist.substring(itemstart, i);
                       itemstart = i+1;
                  } else if
(fulllist.substring(i,i+1) == '#') {
```

```
theprice = fulllist.substring(itemstart, i);
                        itemstart = i+1;
                    }
            }
            document.writeln('<tr><td
colspan=3><b>Total</b></td><td
align=right>'+alterError(totprice)+'</td></tr>')
;
      document.writeln('<INPUT TYPE="hidden"
NAME="Goods Total"
VALUE="'+alterError(totprice)+'" SIZE="40">');
            document.writeln('</TABLE>');
      }
</SCRIPT>
</SCRIPT>
<center>
<h2>Buy Goods...</h2>
<!-- call showItems to show items in basket -->
<SCRIPT LANGUAGE="JavaScript">
      showItems();
</SCRIPT>
<br>
<!-- finish off the form with other details and
a /FORM tag -->
<table cols=2>
<tr><td>Email address</td><td><input type=text
name="email" size=40></td></tr>
<tr><td>Street</td><td><input type=text
name="street" size=40></td></tr>
<tr><td>Town/City</td><td><input type=text
name="towncity" size=30></td></tr>
<tr><Td>County</td><td><input type=text
name="county" size=30></td></tr>
<tr><td>Postcode</td><td><input type=text
name="postcode" size=20></td></tr>
<tr><td>Country</td><td><input type=text
name="country" size=20></td></tr>
<tr><td>Telephone</td><td><input type=text
name="telephone" size=30></td></tr>
</table>
```

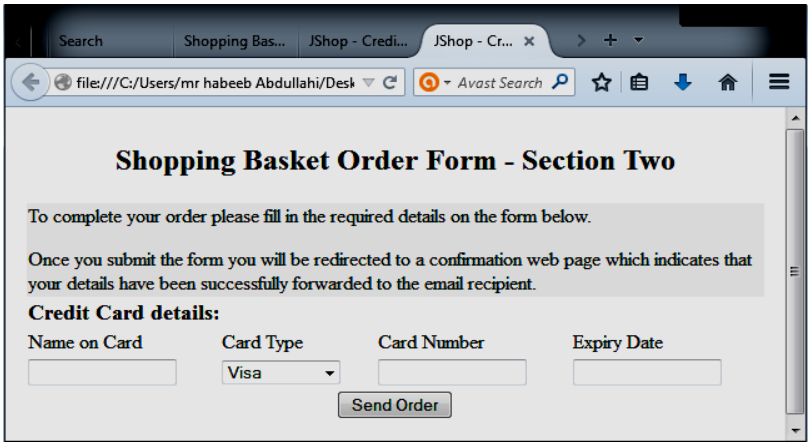
```
<br><br>
<!-- Insert field for email recipient (use your
email address-->
<INPUT TYPE=HIDDEN NAME="recipient"
VALUE="cartdemo@generate74.com">
<!-- Insert field to redirect page to the secure
server to take credit card details -->
<INPUT TYPE=HIDDEN NAME="redirect"
VALUE="https://www.generate74.com/cart/bfinish.h
tm">
<!-- Insert field for referrer and agent details
-->
<input type=hidden name="env_report"
value="REMOTE_HOST,HTTP_USER_AGENT">
<input type="submit" value="Proceed">
<input type="reset" value="Reset">
</center>
</form>
</BODY>
</HTML>
```



BFINISH.htm

```
<HTML>
<!-- JShop Edited by MH7 2018             -->
<!-- by MH7 @ (www.habeebmamman.com)      -->
<!-- SHOPPING BASKET                      -->
<HEAD>
<TITLE>JShop - Credit Card Details (should
reside on secure server really!)</TITLE>
</HEAD>
<BODY BGCOLOR="#FFFFFF">
<center>
<form action="edemo.htm">
<input type="hidden" name="subject"
value="Shopping Cart Demo - Ordering Part 2">
<input type="hidden" name="recipient"
value="cartdemo@generate74.com">
<input type="hidden" name="pgpuserid"
value="securer">
<center>
<table width=600><td><center><h2>Shopping Basket
Order Form - Section Two</h2></center>
<table cols=4><tr><td colspan=4
bgcolor="F4F4F4">
To complete your order please fill in the
required details on the form below.
<p>
Once you submit the form you will be redirected
to a confirmation web page which indicates that
your details have been successfully forwarded to
the email recipient.
</td></tr><p>
<tr><td colspan=2align=left><font
size=+1><b>Credit Card details:
</b></font></td></tr>
<tr><td>Name on Card</td><td>Card
Type</td><td>Card Number</td><td>Expiry
Date</td></tr>
<tr><td><input type=text name="Name_on_card"
size=15></td>
<td><select
name="type"><option>Visa<option>Mastercard</sele
ct></td>
```

```
<td><input type=text name="Card_Number"
size=15></td>
<td><input type=text name="Expiry_Date"
size=15></td></tr>
<tr><td colspan=4 align=center><p><input
type="submit" value="Send
Order"></form></td></tr></table>
</table>
</BODY>
</HTML>
```



APPENDIX B
BUILDING AN E-COMMERCE WEBSITE WITH BOOTSTRAP

Introduction

In this appendix, we will create an e-commerce website that will help you get to grips with web designing using Bootstrap. Initially, we will build the parts step by step and increase the level of difficulty gradually so that you do not get overwhelmed with an information overload. We will first build the ecommerce.html page and then create the other pages such as category.html, account.html, and product.html, resulting in a responsive website.

With Notepad, you cannot determine whether each element has been closed, resulting in an incorrect output. It is a good practice to use Notepad++ or any advanced editor for your projects, as it streamlines your web designing experience.

Designing the ecommerce.html page

Create four web pages, namely ecommerce.html, category.html, account.html, and product.html for the e-commerce, product categories, account, and the product pages respectively.

Initially, we will look at ecommerce.html.

For now, let's paste the following basic Bootstrap code in the `ecommerce.html` file:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8">
  <meta http-equiv="X-UA-Compatible" content="IE=edge">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta name="description" content="">
  <meta name="author" content="">
   <title>Bootstrap Store</title>

   <!-- Bootstrap Core CSS -->
   <link href="css/bootstrap.css" rel="stylesheet">

   <!-- Custom CSS -->
   <link href="style.css" rel="stylesheet">

   <!-- HTML5 Shim and Respond.js IE8 support of HTML5 elements and
media queries -->
   <!-- WARNING: Respond.js doesn't work if you view the page via
file:// -->
```

```
    <!--[if lt IE 9]>
      <script src="https://oss.maxcdn.com/libs/html5shiv/3.7.0/
  html5shiv.js"></script>
      <script src="https://oss.maxcdn.com/libs/respond.js/1.4.2/respond.
  min.js"></script>
    <![endif]-->
  </head>
       <body>
         <!-- jQuery Version 1.11.0 -->
         <script src="js/jquery-1.11.1.js"></script>

         <!-- Bootstrap Core JavaScript -->
         <script src="js/bootstrap.js"></script>
       </body>
       </html>
```

As you can see, we have also added `respond.min.js`, the jQuery file, and HTML shiv along with the Bootstrap files.

We will now create a navbar for the web page.

Similar to the process of creating a navbar in the preceding chapters, you define the navbar between the <body> tags. We define navbar-brand as Bootstrap Store and use the .navbar-inverse class, resulting in a black background and white text for the navbar:

```
<body>
<!-- Navigation -->
  <nav class="navbar-inverse" role="navigation">
    <div class="container-fluid">
      <!-- Brand and toggle get grouped for better mobile display -->
      <div class="navbar-header">
        <button type="button" class="navbar-toggle collapsed" data-
toggle="collapse" data-target="#bs-example-navbar-collapse-1">
          <span class="sr-only">Toggle navigation</span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
          <span class="icon-bar"></span>
        </button>
        <a class="navbar-brand" href="ecommerce.html">Bootstrap
Store</a>
      </div>

      <!-- Collect the nav links, forms, and other content for
toggling -->
      <div class="collapse navbar-collapse" id="bs-example-navbar-
collapse-1">
      </div><!-- /.navbar-collapse -->
    </div><!-- /.container-fluid -->
  </nav>
```

```
    <!-- jQuery Version 1.11.0 -->
    <script src="js/jquery-1.11.1.js"></script>

    <!-- Bootstrap Core JavaScript -->
    <script src="js/bootstrap.js"></script>
  </body>
```

The output of the code on execution will be as follows:

Further on, we will add the categories and navigation links to this navbar. The code has to be inserted after the <!-- Collect the nav links, forms, and other content for toggling --> comment and before the element containing the <!-- /.navbar-collapse --> comment.

In the code, we define the **Categories** link in addition to the other navigation links. We create a dropdown for the **Categories** link, wherein we define the various types of products by their genre such as baby products, electronics, and shoes.

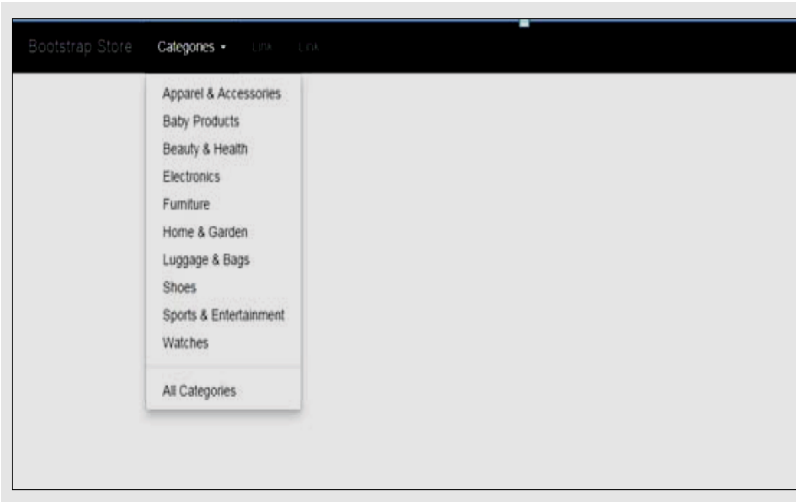
Take a look at the following code to understand it better:

```
<div class="collapse navbar-collapse" id="bs-example-navbar-
collapse-1">
<ul class="nav navbar-nav">
          <li class="dropdown">
            <a href="#"  class="active dropdown-toggle" data-
toggle="dropdown">Categories <span class="caret"></span></a>
            <ul class="dropdown-menu" role="menu">
              <li><a href="category.html">Apparel  &amp; Accessories</
a></li>
              <li><a href="category.html">Baby Products</a></li>
              <li><a href="category.html">Beauty  &amp; Health</a></
li>
             <li><a href="category.html">Electronics</a></li>
             <li><a href="category.html">Furniture</a></li>
             <li><a href="category.html">Home  &amp; Garden</a></li>
             <li><a href="category.html">Luggage  &amp; Bags</a></li>
             <li><a href="category.html">Shoes</a></li>
             <li><a href="category.html">Sports  &amp;
Entertainment</a></li>
             <li><a href="category.html">Watches</a></li>
```

```
        <li class="divider"></li>
        <li><a href="ecommerce.html">All Categories</a></li>
      </ul>
    </li>
    <li><a href="#">Link</a></li>
    <li><a href="#">Link</a></li>
  </ul>
```

The output of the code will be as follows:

Thus, we have defined the **Categories** menu and the drop-down menu, which is visible on clicking the caret.



Now, we add the **Sign in** link, user account link, and the shopping cart with a badge to the right-hand side of the navbar and specify Glyphicons for them. Remember that this snippet needs to be added after the links defined prior to it and before the element containing the <!-- /.navbar-collapse --> comment:

```
<ul class="nav navbar-nav navbar-right">
          <li><a href="#"><span class="badge pull-right">4</span><i
class="glyphicon glyphicon-shopping-cart"></i></a></li>
          <li><a href="account.html"><i class="glyphicon glyphicon-
user"></i></a></li>
          <li><a href="#" data-toggle="modal" data-
target="#myModal">Sign in</a></li>
        </ul>
```

The output of the added code on execution will result in the following screenshot:

For the **Sign in** part, we will create a modal that will be displayed by clicking on the **Sign in** link.
The following code snippet needs to be added after the closed </nav> element:

```
<!-- Modal -->
  <div class="modal fade" id="myModal" tabindex="-1" role="dialog"
aria-labelledby="myModalLabel" aria-hidden="true">
    <div class="modal-dialog">
      <div class="modal-content">
        <div class="modal-header">
          <button type="button" class="close" data-

dismiss="modal"><span aria-hidden="true">&times;</span><span
class="sr-only">Close</span></button>
          <h2 class="modal-title" id="myModalLabel">Sign in</h2>
        </div>
        <div class="modal-body">
          <form class="form-signin" role="form">
            <h3 class="form-signin-heading">Sign in with your email
address</h3>
              <div class="form-group">
                <input type="email" class="form-control"
    placeholder="Email address" required autofocus>
              </div>
              <div class="form-group">
                <input type="password" class="form-control"
    placeholder="Password" required>
            </div>
            <div class="checkbox">
              <label>
                <input type="checkbox" value="remember-me"> Remember
me
              </label>
            </div>
```

```
                <button class="btn btn-lg btn-primary btn-block"
    type="submit">Sign in</button>
              </form>
            </div>

            <div class="row">
              <div class="col-xs-3">
                <a href="#" class="btn btn-facebook btn-large btn-caps
btn-block">Facebook <span class="icon-facebook"></span></a>
            </div>
            <div class="col-xs-3">
              <a href="" class="btn btn-twitter btn-large btn-caps btn-
block">Twitter <span class="icon-twitter"></span></a>
            </div>
            <div class="col-xs-3">


              <a href="" class="btn btn-lg btn-caps btn-block"><span
class="icon-dribbble"></span></a>
            </div>
          </div>

          <div class="modal-footer">
          </div>
        </div>
      </div>
    </div>
```

If you click on the **Sign in** link, the following dialog box will be displayed:

From the preceding code and the output, you can see that we have defined a form to get the user authentication input and have inserted it inside the code for the modal.
We will now create a carousel for the web page. After the modal code has been defined, we define the page content. We start with the <!-- Page Content --> comment for easy readability, which indicates that the entire content will be defined within the element using the .container class and content as the ID for it.
Further on, we define the carousel within that container using the following code snippet:

```
<div id="content" class="container">

    <div class="row carousel-holder">

      <div class="col-md-12">
        <div id="carousel-example-generic" class="carousel slide"
data-ride="carousel">


            <ol class="carousel-indicators">
              <li data-target="#carousel-example-generic" data-slide-
  to="0" class="active"></li>
              <li data-target="#carousel-example-generic" data-slide-
  to="1"></li>
              <li data-target="#carousel-example-generic" data-slide-
  to="2"></li>
            </ol>
            <div class="carousel-inner">
              <div class="item active">
                <img class="slide-image" src="http://placehold.
it/1140x350" alt="" />
              </div>
              <div class="item">
                <img class="slide-image" src="http://placehold.
it/1140x350" alt="" />
              </div>
              <div class="item">
                <img class="slide-image" src="http://placehold.
it/1140x350" alt="" />
              </div>
            </div>
```

```
          <a class="left carousel-control" href="#carousel-example-
generic" data-slide="prev">
            <span class="glyphicon glyphicon-chevron-left"></span>
          </a>
          <a class="right carousel-control" href="#carousel-example-
generic" data-slide="next">
            <span class="glyphicon glyphicon-chevron-right"></span>
          </a>
        </div>
      </div>

    </div>
</div><!-- /.container class with content as the id-->
```

Now that we have defined the carousel, the output of the code upon execution will be as follows:

Let's now define the product categories after the carousel code using the following code snippet:

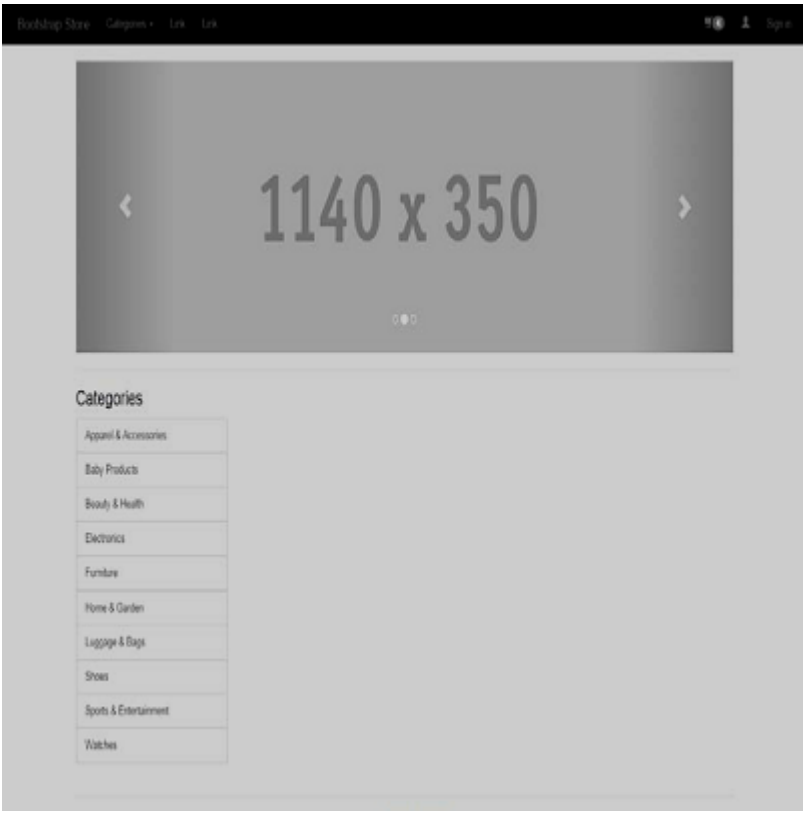
```
<hr />
    <div class="row">
      <div class="col-sm-4 col-md-3">
        <h3>Categories</h3>
        <div class="list-group">
          <a href="category.html" class="list-group-item">Apparel &
Accessories</a>
          <a href="category.html" class="list-group-item">Baby
Products</a>
          <a href="category.html" class="list-group-item">Beauty &
Health</a>
          <a href="category.html" class="list-group-
item">Electronics</a>
```

```
          <a href="category.html" class="list-group-item">Furniture</
a>
          <a href="category.html" class="list-group-item">Home &
Garden</a>
          <a href="category.html" class="list-group-item">Luggage &
Bags</a>
          <a href="category.html" class="list-group-item">Shoes</a>
          <a href="category.html" class="list-group-item">Sports &
 Entertainment</a>
           <a href="category.html" class="list-group-item">Watches</a>
         </div>
       </div>
     </div>
   </div><!-- /.container class with content as the id-->
```

After all the code we have written so far, let's create a footer for the web page by defining the element using the .container class only. Take a look at the following code snippet to understand it better:

```
      <div class="container">
       <hr />
       <!-- Footer -->
       <footer>
         <div class="row">
           <div class="col-lg-12">
             <p>Copyright &copy; <a href="ecommerce.html">Packt
  Publishing</a> 2014</p>
           </div>
         </div>
       </footer>
   </div>
```

The output of the code on adding the product categories and the footer will be as follows:



Now we will proceed further and add the products along with a brief description so that they are reflected on your web page. The element with the categories was defined with the .col-sm-4 col-md-3 class, and it occupies three columns of the web page on the left-hand side on a medium-sized display screen and four columns on a small-screen device. Since it is a 12-column grid, the remaining space will be used to display individual products.

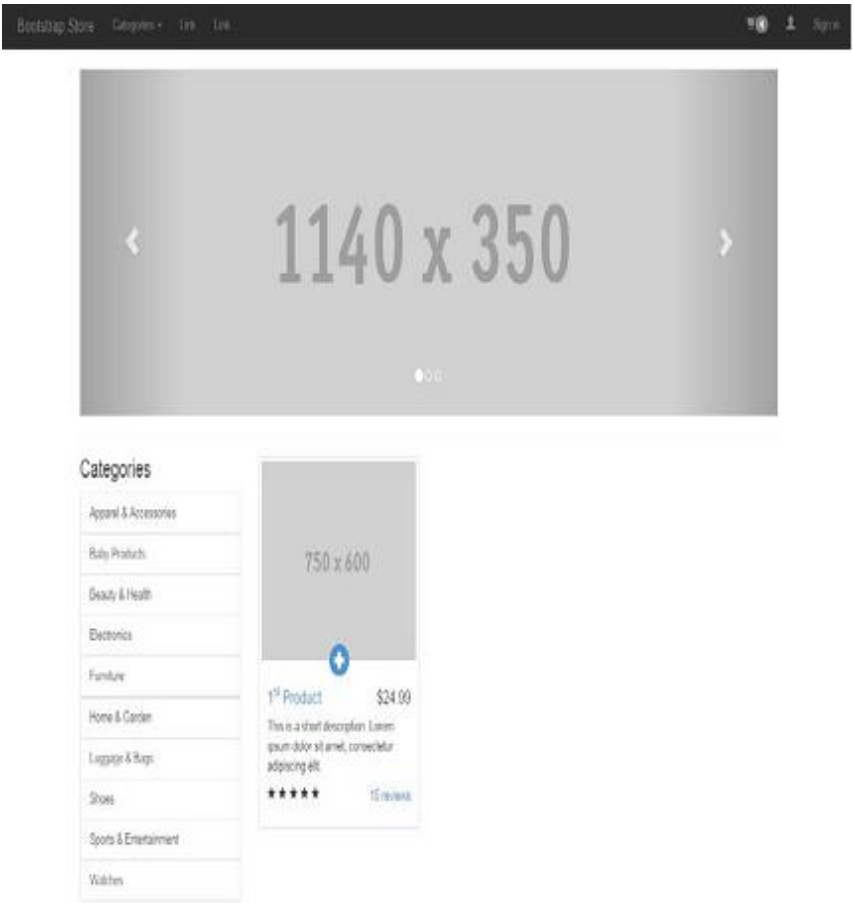
Therefore, after the categories have been defined, let's add the following code to it:

```
<div class="col-sm-8 col-md-9">
        <div class="row">

        </div>
</div>
```

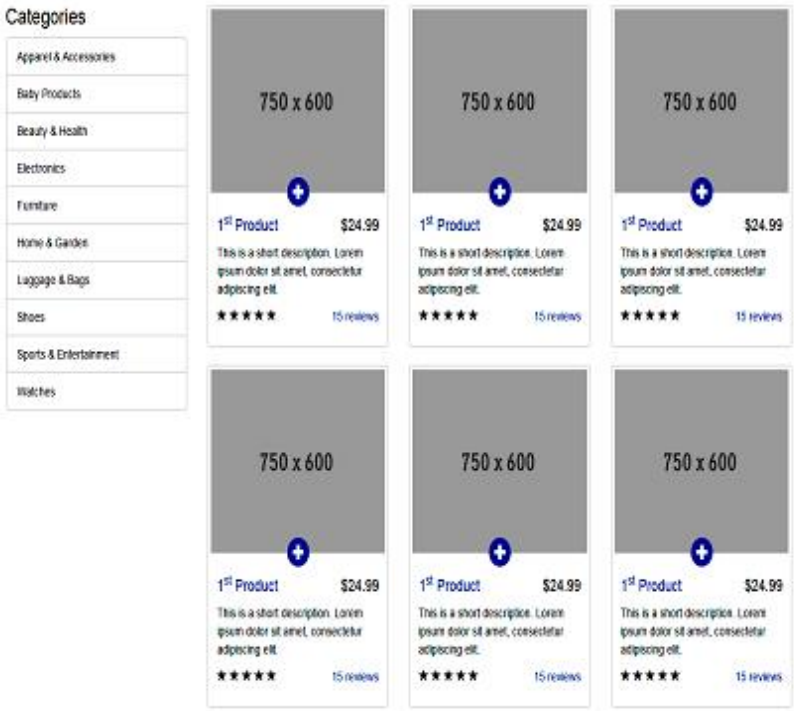
Insert the following code in the preceding nested element with the .row class to add a product to the right-hand side of the **Categories** menu:

```
<div class="col-sm-6 col-md-4">
            <div class="thumbnail">
              <img src="http://placehold.it/750x600" alt="">
              <div class="add-to-cart">
                <a href="#" class="glyphicon glyphicon-plus-sign"
data-toggle="tooltip" data-placement="top" title="Add to cart"></a>
              </div>
              <div class="caption">
                <h4 class="pull-right">$24.99</h4>
                <h4><a href="product.html">1<sup>st</sup> Product</a>
                </h4>
                <p>This is a short description. Lorem ipsum dolor sit
amet, consectetur adipiscing elit.</p>
                 <div class="ratings">
                   <p class="pull-right"><a href="product.
html#comments">15 reviews</a></p>
                   <p>
                     <span class="glyphicon glyphicon-star"></span>
                     <span class="glyphicon glyphicon-star"></span>
                     <span class="glyphicon glyphicon-star"></span>
                     <span class="glyphicon glyphicon-star"></span>
                    <span class="glyphicon glyphicon-star"></span>
                  </p>
                </div>
              </div>
            </div>
          </div>
```

The output after defining the product is as follows:



Copy the code several times and paste it to get a grid of replicated products. On execution of the code, you can see the following screen below the carousel:

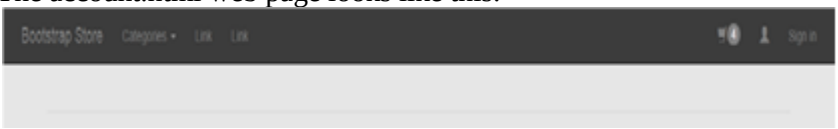


Therefore, we have succeeded in building a web page with the navbar with links, categories to the left-hand side, and products ranging from the center to the right-hand side of the page. We defined a carousel and also defined a modal, wherein a dialog box comes into the picture on clicking on the **Sign in** option.

Now, let's look at the next steps wherein we create the remaining three pages, that is, account.html, category.html, and product.html, so that you have a real-time scenario.

Designing the account.html web page

The account.html web page looks like this:



Similar to the procedure followed in the ecommerce.html page, we define the page content. Note that you need to add the page content by inserting the code snippet after the modal code and before the element with the .container class that encloses the footer of the web page. For code clarity and readability, we add the <!-- Page Content --> comment and then define a element with the .container class and #content as the ID, and add a element for it. Now our entire code needs to be inserted between these specific elements.

Take a look at the following code snippet to understand it better:

```
<!-- Page Content -->
<div id="content" class="container">
 </div><!-- div element with the content as id and a .container class
-->
```

We then create a element with the .row class, wherein we will include the account menu. Let's create a **Manage Order** section and enclose it within three columns for a medium-sized device. Then, we define a list group of items and define the different options, namely, All Order, Manage Feedback, My Coupons, and My Shipping Address:

```
<div class="row">
      <div class="col-md-3">
        <h3 class="">Manage Orders</h3>
        <div class="list-group">
          <a href="account.html" class="list-group-item">All Order</a>
          <a href="account.html" class="list-group-item">Manage
Feedback</a>
          <a href="account.html" class="list-group-item">My Coupons</
a>
          <a href="account.html" class="list-group-item">My Shipping
Address</a>
        </div>
      </div>
  </div>
```

The output of the code upon execution will be:

Let's define a element with the .thumbnail class and create another element within it with a .row class. Thereafter, we add the product names, the number of orders, prices, the track buttons, and the messages for the latest status of the products.
We will use the following code snippet to understand this better:

```
<div class="thumbnail">
          <div class="row">
            <div class="col-sm-1">
              <img class="img-responsive" src="http://placehold.
it/750x600" alt="" />
            </div>
            <div class="col-sm-4">
              <a href="product.html">Product Name</a>
            </div>
            <div class="col-sm-1">
              1
        </div>
        <div class="col-sm-2">
          $49.99
        </div>
        <div class="col-sm-2">
          <button class="btn btn-sm btn-default">Track</button>
        </div>
        <div class="col-sm-2">
          <a href="#">1 Message</a>
        </div>
      </div>
    </div>
</div>
```

On executing the code, you can see the following output:

Now that we have a single product defined and the order status for that product, let's replicate the products five times. We customize the status of the products by specifying different amounts of messages, prices, and the latest updates of some of the products so as to determine whether the status is **Cancelled** or **Completed** (or a **Track** button to determine whether it is on its way).

Thus, to create a wide array of order status' for different products, we use the following code snippet:

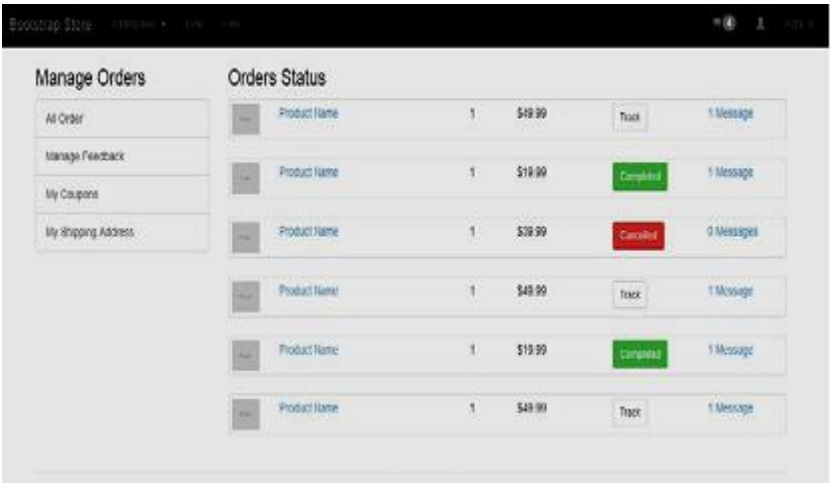
```
<div class="thumbnail">
          <div class="row">
            <div class="col-sm-1">
              <img class="img-responsive" src="http://placehold.
it/750x600" alt="" />
            </div>
            <div class="col-sm-4">
              <a href="product.html">Product Name</a>
            </div>
            <div class="col-sm-1">
              1
            </div>
            <div class="col-sm-2">
              $19.99
            </div>
            <div class="col-sm-2">
              <button class="btn btn-sm btn-success">Completed</
button>
            </div>
```

```
          <div class="col-sm-2">
            <a href="#">1 Message</a>
          </div>
        </div>
      </div>

  <div class="thumbnail">
        <div class="row">
          <div class="col-sm-1">
            <img class="img-responsive" src="http://placehold.
it/750x600" alt="" />
          </div>
          <div class="col-sm-4">
            <a href="product.html">Product Name</a>
          </div>
      <div class="col-sm-1">
        1
      </div>
      <div class="col-sm-2">
        $39.99
      </div>
      <div class="col-sm-2">
        <button class="btn btn-sm btn-danger">Cancelled</button>
      </div>
          <div class="col-sm-2">
            <a href="#">0 Messages</a>
          </div>
        </div>
      </div>

  <div class="thumbnail">
        <div class="row">
          <div class="col-sm-1">
            <img class="img-responsive" src="http://placehold.
```

```
                <img class="img-responsive" src="http://placehold.
it/750x600" alt="" />
              </div>
              <div class="col-sm-4">
                <a href="product.html">Product Name</a>
              </div>
              <div class="col-sm-1">
                1
              </div>
              <div class="col-sm-2">
      $49.99
    </div>
    <div class="col-sm-2">
      <button class="btn btn-sm btn-default">Track</button>
    </div>
    <div class="col-sm-2">
      <a href="#">1 Message</a>


            </div>
          </div>
        </div>

      <div class="thumbnail">
          <div class="row">
            <div class="col-sm-1">
              <img class="img-responsive" src="http://placehold.
 it/750x600" alt="" />
    </div>
    <div class="col-sm-4">
      <a href="product.html">Product Name</a>
    </div>
    <div class="col-sm-1">
      1
    </div>
    <div class="col-sm-2">
      $19.99
```

```
            </div>
            <div class="col-sm-2">
              <button class="btn btn-sm btn-success">Completed</
  button>
            </div>
            <div class="col-sm-2">
              <a href="#">1 Message</a>
            </div>
          </div>
        </div>

    <div class="thumbnail">
          <div class="row">
            <div class="col-sm-1">
              <img class="img-responsive" src="http://placehold.
it/750x600" alt="" />
            </div>
            <div class="col-sm-4">
              <a href="product.html">Product Name</a>
            </div>

    <div class="col-sm-1">
      1
    </div>
    <div class="col-sm-2">
      $49.99
    </div>
    <div class="col-sm-2">
      <button class="btn btn-sm btn-default">Track</button>
    </div>
    <div class="col-sm-2">
              <a href="#">1 Message</a>
            </div>
          </div>
      </div>
```

The output of the code upon execution will be as follows:

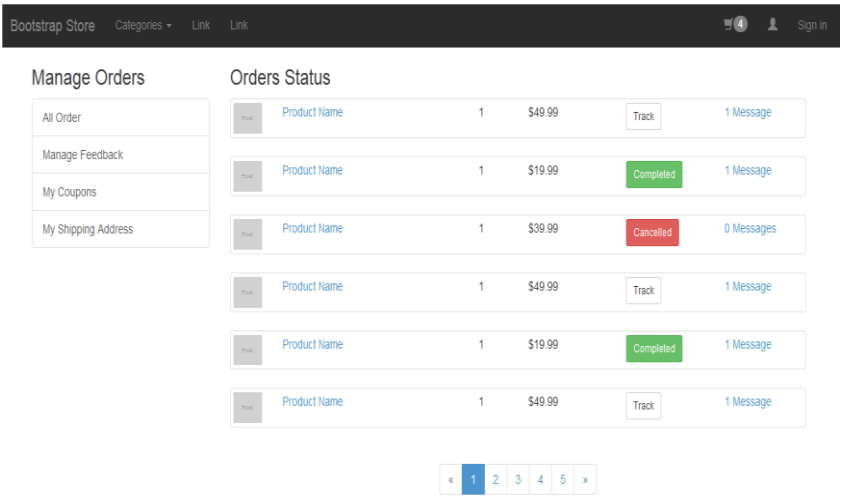


Let's further enhance the web page by adding a .pagination class so that your website users can scroll for different products on different web pages.

Take a look at the following code snippet to understand it better:

```
<div class="col-sm-12 center">
          <ul class="pagination">
            <li class="disabled"><a href="#">&laquo;</a></li>
            <li class="active"><a href="#">1</a></li>
            <li><a href="#">2</a></li>
            <li><a href="#">3</a></li>
            <li><a href="#">4</a></li>
            <li><a href="#">5</a></li>
            <li><a href="#">&raquo;</a></li>
          </ul>
        </div>
```

The output of the executed code will have pagination at the bottom of the page displayed as follows:

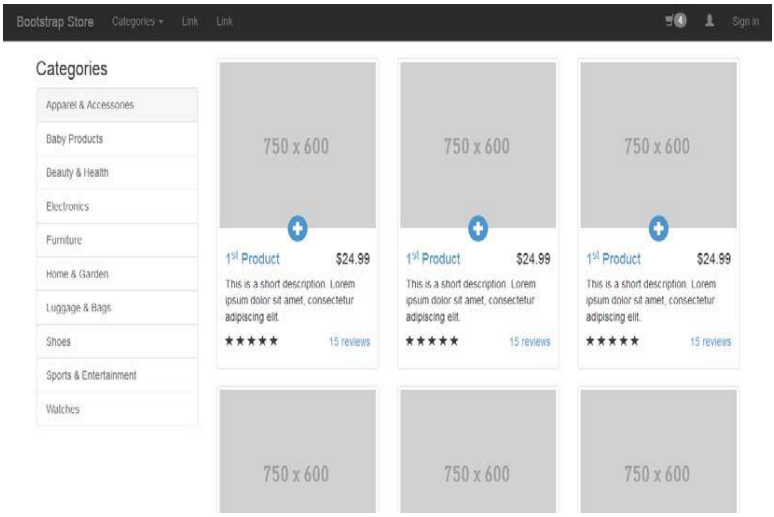


Designing the category.html web page

Just like the account.html web page, the category.html web page looks like the following screenshot due to the common code containing the navigation links, the modal, and the various Glyphicons:

Similar to the e-commerce web page, we then create a page content section after the modal. We add a element with the .container class and #content as the ID and close it with its corresponding element. Within these specific elements, we create a category menu using the list-group items attribute. It is the same code snippet used to create a category menu on the left-hand side of the web page. We do not use a carousel here as is the case with the ecommerce.html web page. Then, we add the products in the same way we defined the products in the ecommerce.html web page.

The category.html web page will look like this:

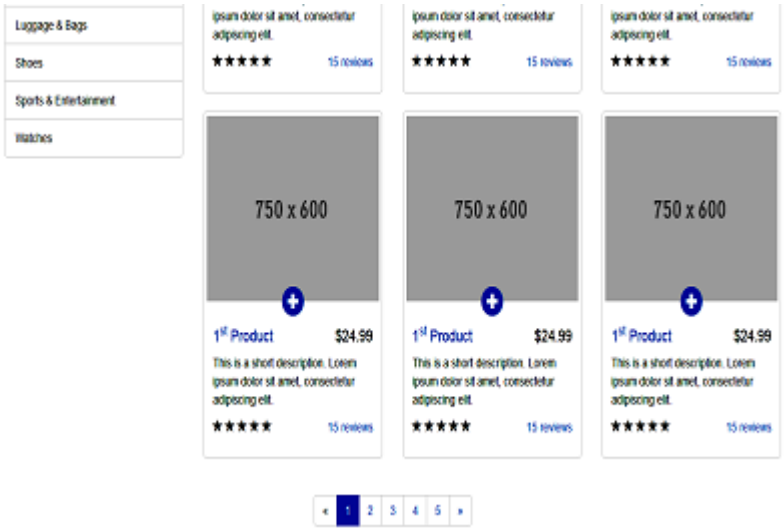


Add the following .pagination class to the code for it to reflect on different pages. It is similar to the code we used for pagination for the account.html web page.

Take a look at the following code to understand it better:

```
<div class="col-sm-12 center">
          <ul class="pagination">
            <li class="disabled"><a href="#">&laquo;</a></li>
            <li class="active"><a href="#">1</a></li>
            <li><a href="#">2</a></li>
            <li><a href="#">3</a></li>
            <li><a href="#">4</a></li>
            <li><a href="#">5</a></li>
            <li><a href="#">&raquo;</a></li>
          </ul>
        </div>
```

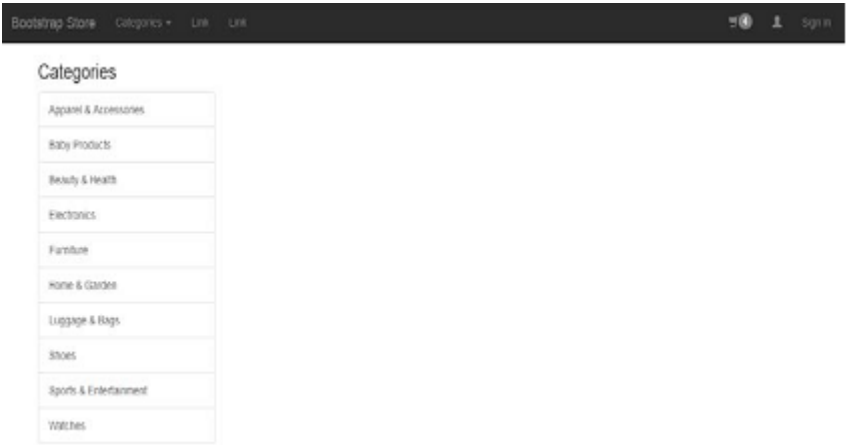
Now, the lower half of your category.html page will look like this:



We have defined the account.html and category.html pages so far. Next, we proceed to the final part, the product.html web page.

Designing the product.html web page

Similar to category.html, we create a **Categories** menu on the left-hand side of the screen below the navbar.

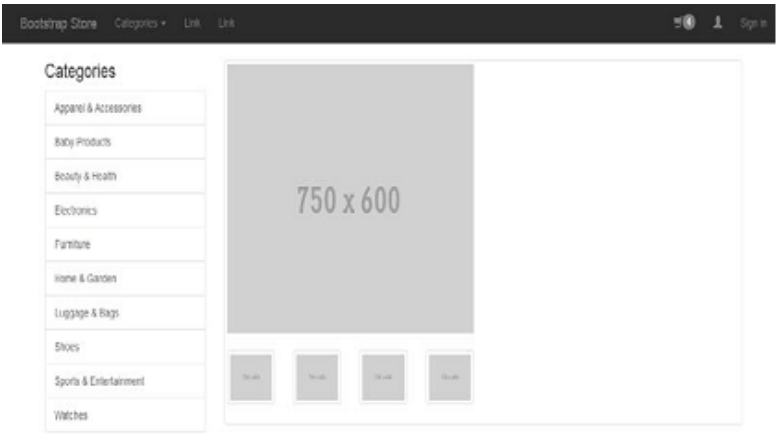


Suppose we want to add a particular product to the web page. Initially, we define the column width for the product. In this case, we have defined the .col-md-9 class as the space to be particularly assigned to the product, meaning that the product details can be viewed on the right-hand side of the **Categories** menu. Then, we define the graphic

image for the product by defining the .thumbnail class. Next, we nest four small different images in the same .thumbnail class.
Take a look at the following code to understand it better:

```
<div class="col-md-9">

        <div class="thumbnail">
          <div class="row">
            <div class="col-sm-6">
              <img class="img-responsive" src="http://placehold.
it/750x600" alt="" />
              <div class="thumbnails row">
                <div class="col-xs-3">
                  <a href="#"><img src="http://placehold.it/750x600"
alt="" class="img-thumbnail img-responsive" /></a>
</div>
<div class="col-xs-3">
                </div>
                <div class="col-xs-3">
<a href="#"><img src="http://placehold.it/750x600" alt="" class="img-
thumbnail img-responsive" /></a>
                </div>
              </div>
            </div>
        </div>
```

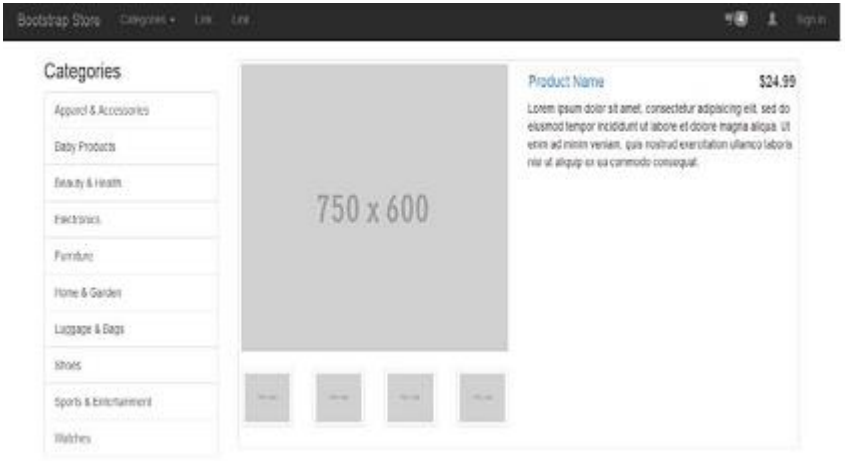
The output of the code upon execution will be as follows:



Next, we define the product name and a short description that will be displayed to the right-hand side of the main image. In the code, we have defined the .col-sm-6 class to determine the space assigned for the product name and the short description.
Therefore, we add the following code for this purpose:

```
<div class="col-sm-6">
              <h4 class="pull-right">$24.99</h4>
              <h4><a href="product.html">Product Name</a></h4>
              <p>Lorem ipsum dolor sit amet, consectetur adipisicing
elit, sed do eiusmod tempor incididunt ut labore et dolore magna
aliqua. Ut enim ad minim veniam, quis nostrud exercitation ullamco
laboris nisi ut aliquip ex ea commodo consequat.</p>
</div>
```

The output of the code will be as follows:

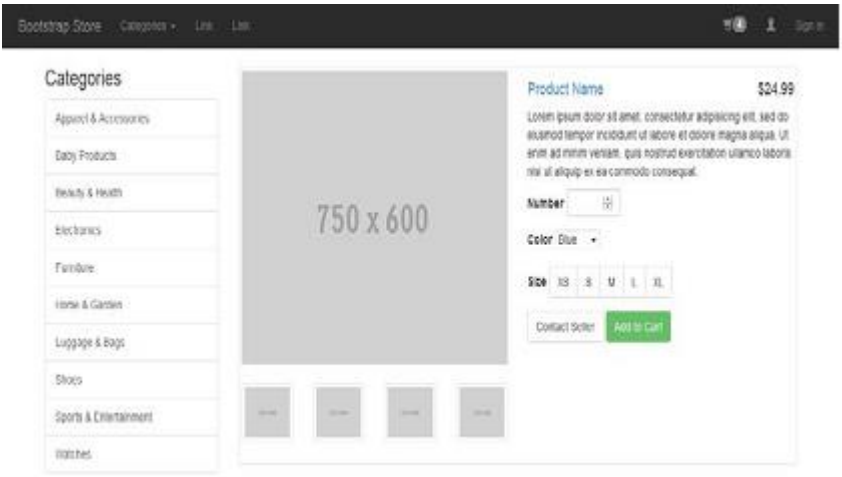


Now, we write the code for a form, wherein we specify the color, number of products to be ordered, and size, while also adding the **Contact Seller** and **Add to Cart** buttons.
Take a look at the following code to understand it better:

```
<form role="form">
                <div class="number form-group">
                  <label class="control-label" for="number">Number</
label>
                  <input type="number" class="form-control input-sm"
id="number" />
                </div>

                <div class="form-group">
                  <label>Color</label>
```

```
            <select id="color">
              <option name="color">Blue</option>
              <option name="color">Green</option>
              <option name="color">Red</option>
              <option name="color">Yellow</option>
            </select>
          </div>

              <div class="form-group">
                <label>Size</label>

              <div class="btn-group">
                <button type="button" class="btn btn-default">XS</
button>
                <button type="button" class="btn btn-default">S</
button>
                <button type="button" class="btn btn-default">M</
button>
                <button type="button" class="btn btn-default">L</
button>
                <button type="button" class="btn btn-default">XL</
button>
              </div>
            </div>

            <div class="form-group">
              <button type="submit" class="btn btn-
default">Contact Seller</button>
              <button type="submit" class="btn btn-success">Add to
Cart</button>
                                  </div>
                                </form>
```

The output of the code on execution will be as follows:



Let's also add the product's detailed description while also defining the reviews section. We specify the number of ratings and use Glyphicons to create the stars to rate the review. Using a .wells class, we create a shaded space to house the **Leave a Review** text.

Take a look at the following code snippet to understand this:

```
<div class="description">
   <p>
Excepteur sint occaecat cupidatat non proident, sunt in culpa qui
officia deserunt mollit anim id est laborum. Lorem ipsum dolor sit
amet, consectetur adipiscing elit. Cras interdum cursus est, facilisis
imperdiet diam fringilla vel. Proin ut sagittis nibh, vehicula euismod
sapien. Cras commodo pellentesque aliquet. Nullam interdum urna et
nibh dictum, id feugiat risus volutpat. Nulla ac velit fringilla,
efficitur arcu a, ultrices erat. Vestibulum elementum metus suscipit
purus vehicula faucibus. Ut lobortis hendrerit magna. In ac urna non
est malesuada maximus in ut nulla.
   </p>

        </div>
        <div class="ratings">
          <p class="pull-right">3 reviews</p>
          <p>
            <span class="glyphicon glyphicon-star"></span>
            <span class="glyphicon glyphicon-star"></span>
```

```
            <span class="glyphicon glyphicon-star"></span>
            <span class="glyphicon glyphicon-star"></span>
            <span class="glyphicon glyphicon-star-empty"></span>
            4.0 stars
          </p>
        </div>
      </div>
  <div class="well">

    <div class="text-right">
      <a class="btn btn-success">Leave a Review</a>
    </div>
```

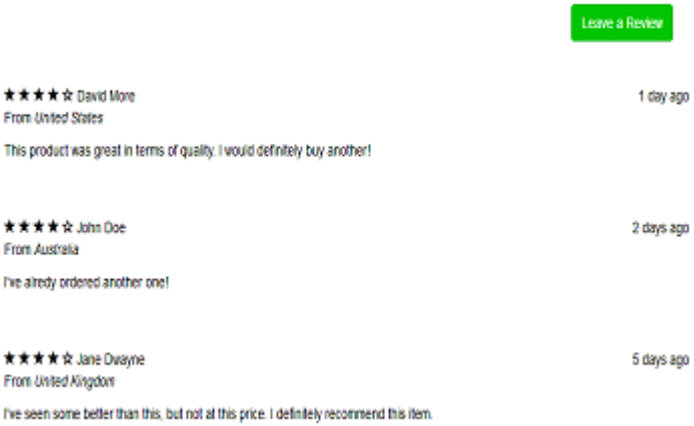
The lower half of the product.html web page will be as follows:

Now, let's just add some reviews to enhance the layout and make it look like an authentic web page. Do this by adding the following code snippet:

```
<div class="row">
            <div class="col-md-12">
              <span class="glyphicon glyphicon-star"></span>
              <span class="glyphicon glyphicon-star"></span>
              <span class="glyphicon glyphicon-star"></span>
              <span class="glyphicon glyphicon-star"></span>
              <span class="glyphicon glyphicon-star-empty"></span>
```

```
            David More
            <span class="pull-right">1 day ago</span>
            <p>From <em>United States</em></p>
            <p>This product was great in terms of quality. I would
definitely buy another!</p>
      </div>
    </div>

    <hr>

    <div class="row">
      <div class="col-md-12">
        <span class="glyphicon glyphicon-star"></span>
        <span class="glyphicon glyphicon-star"></span>
        <span class="glyphicon glyphicon-star"></span>
     <span class="glyphicon glyphicon-star"></span>
     <span class="glyphicon glyphicon-star-empty"></span>
     John Doe
     <span class="pull-right">2 days ago</span>
     <p>From <em>Australia</em></p>
     <p>I've already ordered another one!</p>
      </div>
   </div>

   <hr>

   <div class="row">
     <div class="col-md-12">
       <span class="glyphicon glyphicon-star"></span>
       <span class="glyphicon glyphicon-star"></span>
       <span class="glyphicon glyphicon-star"></span>
       <span class="glyphicon glyphicon-star"></span>
            <span class="glyphicon glyphicon-star-empty"></span>
            Jane Dwayne
            <span class="pull-right">5 days ago</span>
            <p>From <em>United Kingdom</em></p>
            <p>I've seen some better than this, but not at this
price. I definitely recommend this item.</p>
          </div>
        </div>
```

The addition of reviews results in the following output that is displayed on the lower-half of the page as we scroll down the screen:



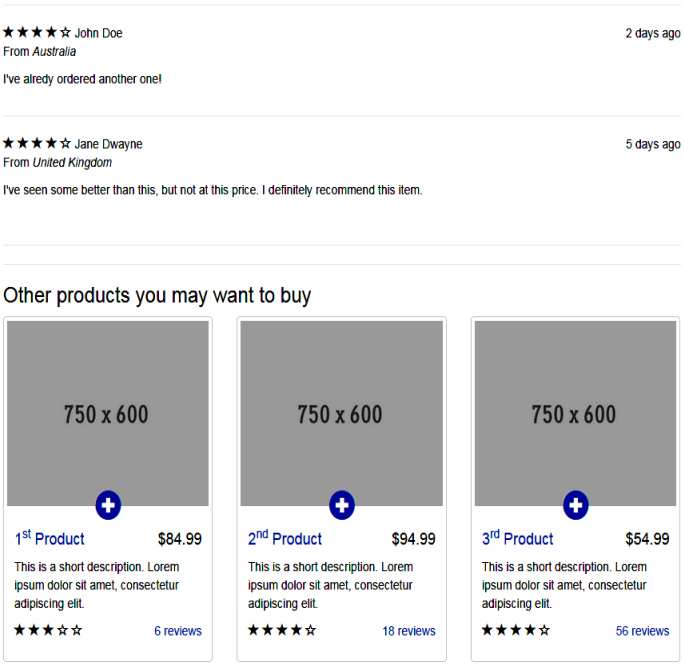
In real-time scenarios, on an e-commerce website, we can sometimes view the **Similar Products** section or **Other things to Buy** suggestions, which are kind of related to the showcased product. So in our following code snippet, we will take a look at the procedure to showcase similar products so that you can implement it in your web designing projects. Take a look at the following code to understand it better:

```
<div class="row">
      <div class="col-sm-12">
        <h3> Other products you may want to buy </h3>
      </div>

      <div class="col-sm-6 col-md-4">
        <div class="thumbnail">
          <img src="http://placehold.it/750x600" alt="">
          <div class="add-to-cart">
            <a href="#" class="glyphicon glyphicon-plus-sign"
data-toggle="tooltip" data-placement="top" title="Add to cart"></a>
          </div>
          <div class="caption">
            <h4 class="pull-right">$84.99</h4>
            <h4><a href="product.html">1<sup>st</sup> Product</a>
            </h4>
```

```
                <p>This is a short description. Lorem ipsum dolor sit
   amet, consectetur adipiscing elit.</p>
                <div class="ratings">
                  <p class="pull-right"><a href="product.
   html#comments">6 reviews</a></p>
                  <p>
                   <span class="glyphicon glyphicon-star"></span>
                   <span class="glyphicon glyphicon-star"></span>
                   <span class="glyphicon glyphicon-star"></span>
                   <span class="glyphicon glyphicon-star-empty"></
  span>
                   <span class="glyphicon glyphicon-star-empty"></
  span>
                                        </p>
                                      </div>
                                    </div>
                                  </div>

            <div class="col-sm-6 col-md-4">
              <div class="thumbnail">
                <img src="http://placehold.it/750x600" alt="">
                <div class="add-to-cart">
                  <a href="#" class="glyphicon glyphicon-plus-sign"
   data-toggle="tooltip" data-placement="top" title="Add to cart"></a>
                </div>
                <div class="caption">
                  <h4 class="pull-right">$94.99</h4>
                  <h4><a href="product.html">2<sup>nd</sup> Product</a>
                  </h4>
                  <p>This is a short description. Lorem ipsum dolor sit
   amet, consectetur adipiscing elit.</p>
                  <div class="ratings">
                    <p class="pull-right"><a href="product.
html#comments">18 reviews</a></p>
                  <p>
                    <span class="glyphicon glyphicon-star"></span>
                    <span class="glyphicon glyphicon-star"></span>
                    <span class="glyphicon glyphicon-star"></span>

                         <span class="glyphicon glyphicon-star"></span>
                         <span class="glyphicon glyphicon-star-empty"></
       span>
                       </p>
                      </div>
                    </div>
```

```
                </div>
              </div>

    <div class="col-sm-6 col-md-4">
                <div class="thumbnail">
                  <img src="http://placehold.it/750x600" alt="">
                  <div class="add-to-cart">
                    <a href="#" class="glyphicon glyphicon-plus-sign"
data-toggle="tooltip" data-placement="top" title="Add to cart"></a>
                </div>
                <div class="caption">
                  <h4 class="pull-right">$54.99</h4>
                  <h4><a href="product.html">3<sup>rd</sup> Product</a>
                  </h4>
                  <p>This is a short description. Lorem ipsum dolor sit
amet, consectetur adipiscing elit.</p>
                  <div class="ratings">
                    <p class="pull-right"><a href="product.
 html#comments">56 reviews</a></p>
                     <p>
                       <span class="glyphicon glyphicon-star"></span>
                       <span class="glyphicon glyphicon-star"></span>
                       <span class="glyphicon glyphicon-star"></span>
                       <span class="glyphicon glyphicon-star"></span>
                       <span class="glyphicon glyphicon-star-empty"></
                     span>
                                          </p>
                                        </div>
                                      </div>
                                    </div>
                                  </div>
```

On execution of the code, the following page is displayed when you scroll down the web page:



From the preceding code and the output, you can see that we have defined the suggested products and assigned the same space for all of them. We defined the .col-sm-6 col-md-4 class for 1st Product, 2nd

Product, and 3rd Product respectively, due to which you see the products displayed when you scroll down the product page.
In the preceding code example, we have linked all the products to the product.html page. Thus, by clicking on any product in the web page, you will be directed to the same product.html page. In real-time scenarios, you can link the products to their respective web pages to see more information and order the products if need be.

WHAT's NEXT?

APPENDIX C
BASIC COMPUTER GRAPHICS, PHOTOSHOP AND FLASH ANIMATION

Computer Graphics

Graphics Systems

Introduction

Computer graphics is not a machine. It is not a computer, nor a group of computer programs. It is not the know-how of a graphic designer, a programmer, a writer, a motion picture specialist, or a reproduction specialist. ***Computer graphics*** is all these –a consciously managed and documented technology directed toward ***communicating*** information accurately and descriptively.

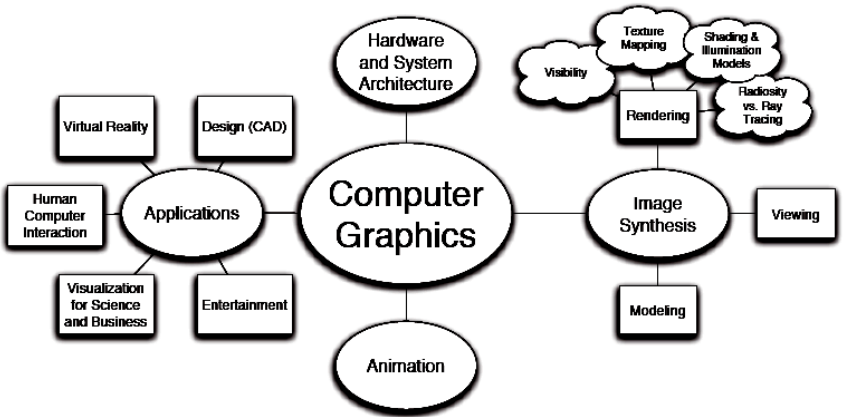


Figure C1: Computer Graphics Elements

- *Using a computer as a rendering tool for the generation (from models) and manipulation of images is called computer graphics.*
- *More precisely: image synthesis.*
- *Display those images on screens or hardcopy devices*
- *Image processing*
- *Others: GUI, Haptics (of the sense of touch), Displays (VR)...*

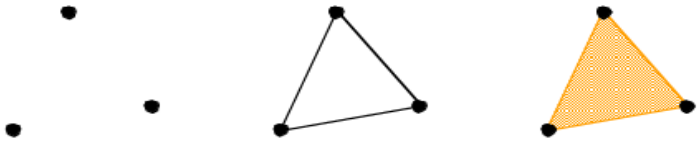


Figure C2: Image Processing

Goals of Computer Graphics

- *Generate synthetic images.*
- *Do it in a practical way and scientifically sound.*
- *In real time.*
- *And make it look easy.*

History of Computer Graphics

The Age of Sutherland

In the early 1960's IBM, Sperry-Rand, Burroughs and a few other computer companies existed. The computers of the day had a few kilobytes of memory, no operating systems to speak of and no graphical display monitors. The peripherals were Hollerith punch cards, line printers, and roll-paper plotters. The only programming languages supported were assembler, FORTRAN, and Algol. Function graphs and "Snoopy" calendars were about the only graphics done.

In 1963 Ivan Sutherland presented his paper Sketchpad at the Summer Joint Computer Conference. Sketchpad allowed interactive design on a vector graphics display monitor with a light pen input device. Most people mark this event as the origins of computer graphics.

The Middle to Late '60's

Software and Algorithms

Jack Bresenham taught us how to draw lines on a raster device (CRTs, TV, monitors). He later extended this to circles. Anti-aliased lines and curve drawing is a major topic in computer graphics. Larry Roberts pointed out the usefulness of homogeneous coordinates, matrices and hidden line detection algorithms. Steve Coons introduced parametric surfaces and developed early computer aided geometric design concepts. The earlier work of Pierre Bézier on parametric curves and surfaces also became public.

Author Appel at IBM developed hidden surface and shadow algorithms that were pre-cursors to ray tracing. The fast Fourier transform was discovered by Cooley and Tukey. This algorithm allows us to better understand signals and is fundamental for developing antialiasing techniques. It is also a precursor to wavelets.

Hardware and Technology

Doug Englebart invented the mouse at Xerox PARC. The Evans & Sutherland Corporation and General Electric started building flight simulators with real-time raster graphics. The floppy disk was invented at IBM and the microprocessor was invented at Intel. The concept of a research network, the ARPANET, was developed.

The Early '70's

The state of the art in computing was an IBM 360 computer with about 64 KB of memory, a Tektronix 4014 storage tube, or a vector display with a light pen (but these were very expensive).

Software and Algorithms

Rendering (shading) were discovered by Gouraud and Phong at the University of Utah. Phong also introduced a reflection model that included specular highlights. Keyframe based animation for 3-D graphics was demonstrated. Xerox PARC developed a ``paint'' program. Ed Catmull introduced parametric patch rendering, the z-buffer algorithm, and texture mapping. BASIC, C, and Unix were developed at Dartmouth and Bell Labs.

Hardware and Technology

An Evans & Sutherland Picture System was the high-end graphics computer. It was a vector display with hardware support for clipping and perspective. Xerox PARC introduced the Altos personal computer, and an 8 bit computer was invented at Intel.

The Middle to Late '70's

Software and Algorithms

Turned Whitted developed recursive ray tracing and it became the standard for photorealism, living in a pristine world. Pascal was the programming language everyone learned.

Hardware and Technology

The Apple I and II computers became the first commercial successes for personal computing. The DEC VAX computer was the mainframe (mini) computer of choice. Arcade games such as Pong and Pac Mac became popular. Laser printers were invented at Xerox PARC.

The Early '80's

Hardware and Technology

The IBM PC was marketed in 1981 The Apple MacIntosh started production in 1984, and microprocessors began to take off, with the Intel x86 chipset, but these were still toys. Computers with a mouse, bitmapped (raster) display, and Ethernet became the standard in academic and science and engineering settings.

The Middle to Late '80's

Software and Algorithms

Jim Blinn introduces blobby (primitive shapes i.e. cylinder) models and texture mapping concepts. Binary space partitioning (BSP) trees were introduced as a data structure, but not many realized how useful they

would become. Loren Carpenter starting exploring fractals in computer graphics. Postscript was developed by John Warnock and Adobe was formed. Steve Cook introduced stochastic sampling to ray tracing. Paul Heckbert taught us to ray trace Jello (this is a joke;) Character animation became the goal for animators. Radiosity was introduced by the Greenberg and folks at Cornell. Photoshop was marketed by Adobe. Video arcade games took off, many people/organizations started publishing on the desktop. Unix and X windows were the platforms of choice with programming in C and C++, but MS-DOS was starting to rise.

Hardware and Technology

Sun workstations, with the Motorola 680x0 chipset became popular as advanced workstation a in the mid 80's. The Video Graphics Array (VGA) card was invented at IBM. Silicon Graphics (SGI) workstations that supported real-time raster line drawing and later polygons became the computer graphicists desired. The data glove, a precursor to virtual reality, was invented at NASA. VLSI for special purpose graphics processors and parallel processing became hot research areas.

The Early '90's

The computers we have then was an SGI workstation with at least 16 MB of memory, at 24-bit raster display with hardware support for Gouraud shading and z-buffering for hidden surface removal. Laser printers and single frame video recorders were standard. Unix, X and Silicon Graphics GL were the operating systems, window system and application programming interface (API) that graphicist used. Shaded raster graphics were starting to be introduced in motion pictures. PCs started to get decent, but still they could not support 3-D graphics, so most programmer's wrote software for scan conversion (rasterization) used the painter's algorithm for hidden surface removal, and developed “tricks”" for real-time animation.

Software and Algorithms

Mosaic, the first graphical Internet browser was written by xxx at the University of Illinois, National Center for Scientific Applications (NCSA). MPEG standards for compressed video began to be promulgated. Dynamical systems (physically based modeling) that allowed animation with collisions, gravity, friction, and cause and effects were introduced. In 1992 OpenGL became the standard for graphics APIs In 1993, the World Wide Web took off. Surface subdivision algorithms were rediscovered. Wavelets begin to be used in computer graphics.

Hardware and Technology

Hand-held computers were invented at Hewlett-Packard about 1991. Zip drives were invented at Iomega. The Intel 486 chipset allowed PC to get reasonable floating point performance. In 1994, Silicon Graphics produced the Reality Engine: It had hardware for real-time texture mapping. The Ninetendo 64 game console hit the market providing Reality Engine-like graphics for the masses of games players. Scanners were introduced.

The Middle to Late '90's

The PC market erupts and supercomputers begin to wane. Microsoft grows, Apple collapses, but begins to come back, SGI collapses, and lots of new startups enter the graphics field.

Software and Algorithms

Image based rendering became the area for research in photo-realistic graphics. Linux and open source software become popular.

Hardware and Technology

PC graphics cards, for example 3dfx and Nvidia, were introduced.

Laptops were introduced to the market. The Pentium chipset makes PCs almost as powerful as workstations. Motion capture, begun with the data glove, becomes a primary method for generating animation sequences.

3-D video games become very popular: DOOM (which uses BSP trees), Quake, Mario Brothers, etc. Graphics effects in movies becomes pervasive: Terminator 2, Jurassic Park, Toy Story, Titanic, Star Wars I. Virtual reality and the Virtual Reality Meta (Markup) Language (VRML) become hot areas for research. PDA's, the Palm Pilot, and flat panel displays hit the market.

The '00's

Today most graphicist want an Intel PC with at least 256 MB of memory and a 10 GB hard drive. Their display should have graphics board that supports real-time texture mapping. A flatbed scanner, color laser printer, digital video camera, DVD, and MPEG encoder/decoder are the peripherals one wants. The environment for program development is most likely Windows and Linux, with Direct 3D and OpenGL, but Java 3D might become more important. Programs would typically be written in C++ or Java.

What will happen in the near future --difficult to say, but high definition TV (HDTV) is poised to take off (after years of hype). Ubiquitous, untethered (not tied down), wireless computing should become

widespread, and audio and gestural input devices should replace some of the functionality of the keyboard and mouse.
You should expect 3-D modeling and video editing for the masses, computer vision for robotic devices and capture facial expressions, and realistic rendering of difficult things like a human face, hair, and water.
With any luck C++ will fall out of favor.

Application of Computer Graphics

- **Movie Industry**
  - *Leaders in quality and artistry*
  - *Not slaves to conceptual purity*
  - *Big budgets and tight schedules*
  - *Reminder that there is more to CG than technology*
- **Game Industry**
  - The newest driving force in CG
    - Why? Volume and Profit
    - This is why we have commodity GPUs (Graphics Processing Unit)
  - Focus on interactivity
  - Cost effective solutions
  - Games drive the baseline
- **Medical Imaging and Scientific Visualization**
  - Tools for teaching and diagnosis
    - No cheating or tricks allowed
  - New data representations and modalities
  - Drive issues of precision and correctness
  - Focus on presentation and interpretation of data
  - Construction of models from acquired data

  Computer graphics makes vast quantities of data accessible. Numerical simulations frequently produce millions of data values. Similarly, satellite-based sensors amass data at rates beyond our abilities to interpret them by any other means than visually. Mathematicians use computer graphics to explore abstract and high-dimensional functions and spaces. Physicists can use computer graphics to transcend the limits of scale. With it they can explore both microscopic and macroscopic world.
- **Computer Aided Design**
  - Mechanical, Electronic, Architecture…etc.
  - Drives the high end of the hardware market
  - Integration of computing and display resources
  - Reduced design cyles == faster systems, sooner

Computer graphics has had a dramatic impact on the design process. Today, most mechanical and electronic designs are executed entirely on computer. Increasingly, architectural and product designs are also migrating to the computer. Automated tools are also available that verify tolerances and design constraints directly from CAD designs. CAD designs also play a key role in a wide range of processes from the design of tooling fixtures to manufacturing.

- **Graphical User Interfaces (GUIs)**

Computer graphics is an integral part of everyday computing. Nowhere is this fact more evident than the modern computer interface design.

Graphical elements such as windows, cursors, menus, and icons are so common place it is difficult to imagine computing without them. Once graphics programming was considered a speciality. Today, nearly all professional programmers must have an understanding of graphics in order to accept input and present output to users.

- **Entertainmen**t

If you can imagine it, it can be done with computer graphics. Obviously, Hollywood has caught on to this. Each summer, we are amazed by state-of- the-art special effects. Computer graphics is now as much a part of the entertainment industry as stunt men and makeup. The entertainment industry plays many other important roles in the field of computer graphics.

Interactive Computer Graphics

User controls contents, structure, and appearance of objects and their displayed images via rapid visual feedback.

Basic components of an interactive graphics system input (e.g., mouse, tablet and stylus, force feedback device, scanner, live video streams…), processing (and storage), display/output (e.g., screen, paper-based printer, video recorder, non-linear editor.

Computer Graphics Requirements

In computer graphics we work with points and vectors defined in terms of some coordinate frame (a positioned coordinate system). We also need to change coordinate representation of points and vectors, hence to transform between different coordinate frames. Hence a mathematical background of geometry and algebra is very essential and also a knowledge of simple programming in C language or its likes.

Graphics Rendering Pipeline

- What is an image?
  – Distribution of light energy on 2D "film".
- How do we represent and store images?
  – Sampled array of "pixels": p[x, y].
- How do we generate images from scenes?
  – Input: 3D description of scene, camera etc
  – Project to camera's viewpoint
  – Illumination

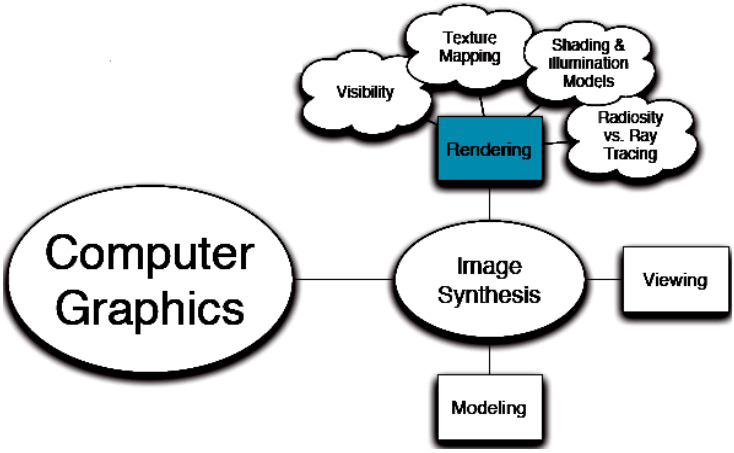


Figure C3: Image Synthesis

Rendering is the conversion of a scene into an image:

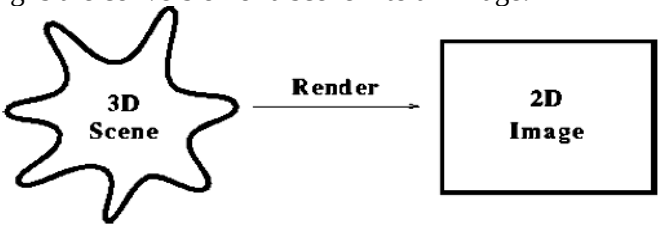


Figure C4: Rendering

The scene composed of models in three space. Models composed of primitives supported by the rendering system. Models are entered by hand or created by a program.

Today, models are already generated. The images drawn on monitor, printed on laser printer, or written to a raster in memory or a file. These different possibilities require us to consider device independence.

Classically, "*model*" to "*scene*"' to "*image*" conversion are broken into finer steps, called the ***graphics pipeline*** commonly implemented in graphics hardware to get interactive speeds. At a high level, the graphics pipeline usually looks like;

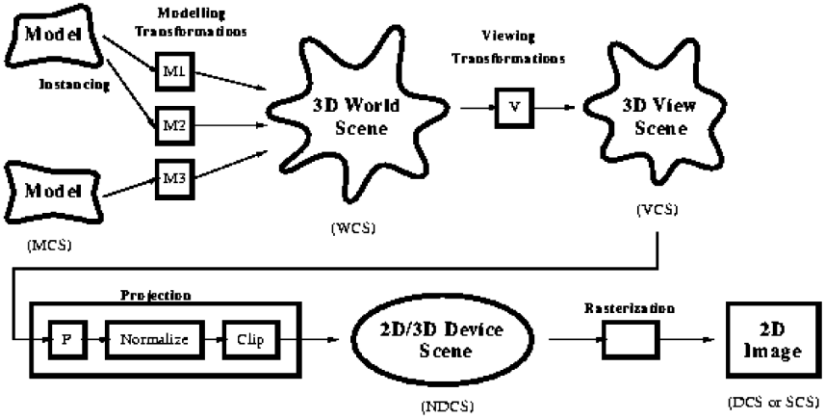


Figure C5: Graphics Pipeline

Each stage refines the scene, converting primitives in modelling space to primitives in device space, where they are converted to pixels (rasterized).

A number of coordinate systems are used:

MCS: Modelling Coordinate System.
WCS: World Coordinate System.
VCS: Viewer Coordinate System.
NDCS: Normalized Device Coordinate System.
DCS or SCS: Device Coordinate System or equivalently the Screen Coordinate System.

Keeping these straight is the key to understanding a rendering system.
Transformation between two coordinate systems represented with matrix. Derived information may be added (lighting and shading) and primitives may be removed (hidden surface removal) or modified (clipping).

Quiz

1. Identify the Ethical issues of computer graphics.

Hardware, Software and Display Devices

Computer graphics don't work in isolation. They require Hardware- the physical component that houses the software. The software tools to create graphics applications are also needed and display devices for

effective output are not left. These and their functionalities, we shall discuss in this section.

Types of Input Devices

Devices can be described either by

- Physical properties
  · Mouse
  · Keyboard
  · Trackball
- Logical Properties
  · What is returned to program via API

A position

An object identifier

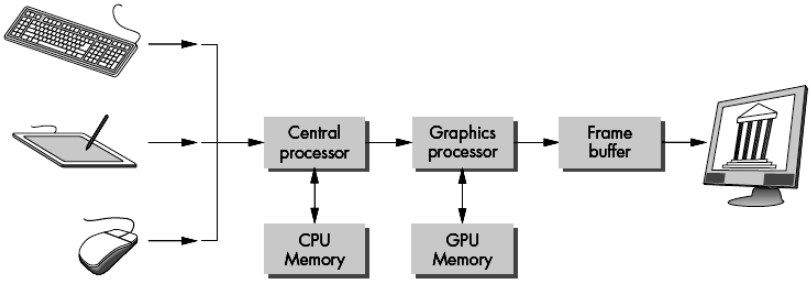


Figure C6: A Graphics system

Input Devices are also categorized as follows:

String: produces string of characters. (e.g keyboard)

Valuator: generates real number between 0 and 1.0 (e.g.knob)

Locator: User points to position on display (e.g. mouse)

Pick: User selects location on screen (e.g. touch screen in restaurant, ATM)

Graphics Software

Graphics software (that is, the software tool needed to create graphics applications) has taken the form of subprogram libraries. The libraries contain functions to do things like: draw points, lines, polygons apply transformations fill areas with color handle user interactions. An important goal has been the development of standard hardware independent libraries such as:

CORE GKS (Graphical Kernel Standard)

PHIGS (Programmer's Hierarchical Interactive Graphics System)

X Windows OpenGL

Hardware vendors may implement some of the OpenGL primitives in hardware for speed.

OpenGL

Every computer has special graphics hardware that controls what you see on the screen. OpenGL tells this hardware what to do.

The Open Graphics Library is one of the oldest, most popular graphics libraries graphic creators have. It was developed in 1992 by Silicon Graphics Inc. (SGI) but only really got interesting for graphic creators when it was used for GLQuake in1997. The GameCube, Wii, PlayStation 3, and the iPhone all base their graphics libraries on OpenGL. The alternative to OpenGL is Microsoft's DirectX. DirectX encompasses a larger number of libraries, including sound and input. It is more accurate to compare OpenGL to the Direct3D library in DirectX.

gl: basic graphics operations
glu: utility package containing some higher-level modelling capabilities (curves, splines)
glut: toolkit. adds platform-independent functions for window management, mouse and keyboard interaction, pull-down menus
glui: adds support for GUI tools like buttons, sliders, etc.

Open Inventor. An object-oriented API built on top of OpenGL.
VRML. Virtual Reality Modeling Language. Allows creation of a model which can then be rendered by a browser plug-in.
Java3d. Has hierarchical modeling features similar to VRML.
POVray. A ray-tracing renderer

Hardware

Modern graphics hardware is very good at processing vast sums of vertices, making polygons from them and rendering them to the screen. This process of going from vertex to screen is called the pipeline. The pipeline is responsible for positioning and lighting the vertices, as well as the projection transformation.

This takes the 3D data and transforms it to 2D data so that it can be displayed on your screen. A projection transformation may sound a little complicated, but the world's painters and artists have been doing these transformations for centuries, painting and drawing the world around them on to a flat canvas.

"Vector graphics" Early graphic devices were line-oriented. For example, a "pen plotter" from H-P. Primitive operation is line drawing.

"Raster graphics" Today's standard. A raster is a 2-dimensional grid of pixels (picture elements). Each pixel may be addressed and illuminated independently. So the primitive operation is to draw a point; that is,

assign a color to a pixel. Everything else is built upon that. There are a variety of raster devices, both hardcopy and display.

Hardcopy:

Laserprinter
Ink-jet printer
Film recorder
Electrostatic printer
Pen plotter

Display Hardware

An important component is the "refresh buffer" or "frame buffer" which is a random-access memory containing one or more values per pixel, used to drive the display. The video controller translates the contents of the frame buffer into signals used by the CRT to illuminate the screen. It works as follows:

i. The display screen is coated with "phosphors" which emit light when excited by an electron beam. (There are three types of phosphor, emitting red, green, and blue light.) They are arranged in rows, with three phosphor dots (R, G, and B) for each pixel.

ii. The energy exciting the phosphors dissipates quickly, so the entire screen must be refreshed 60 times per second.

iii. An electron gun scans the screen, line by line, mapping out a scan pattern. On each scan of the screen, each pixel is passed over once.

Using the contents of the frame buffer, the controller controls the intensity of the beam hitting each pixel, producing a certain color.

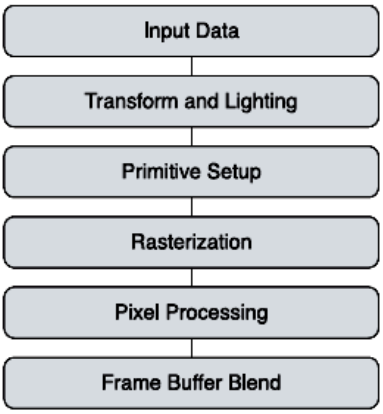


Figure C7: Pipelining

Cathode Ray Tube (CRT) and others

Until recently, the dominant type of display (or **monitor**) was the **cathode-ray tube** (**CRT**). A simplified picture of a CRT is shown in Figure C8. When electrons strike the phosphor coating on the tube, light is emitted. The direction of the beam is controlled by two pairs of deflection plates. The output of the computer is converted, by digital-to-analog converters, to voltages across the *x* and *y* deflection plates. Light appears on the surface of the CRT when a sufficiently intense beam of electrons is directed at the phosphor.

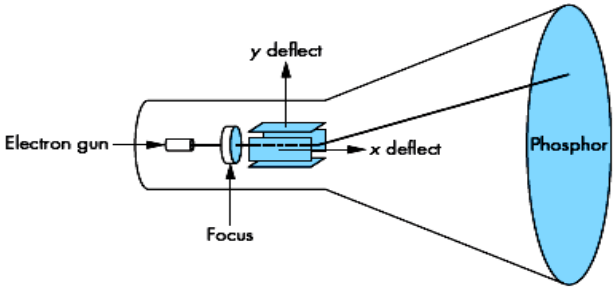


Figure C8: The cathode-ray tube (CRT)

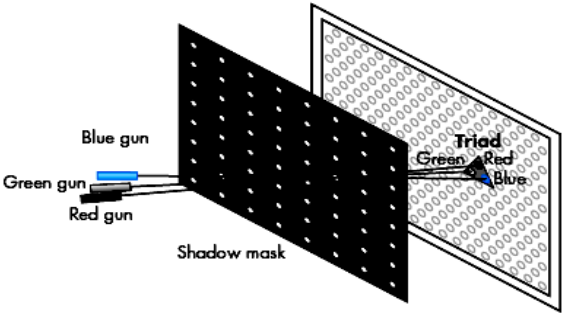


Figure C9: Shadow-mask CRT

Although CRTs are still common display devices, they are rapidly being replaced by flat-screen technologies. Flat-panel monitors are inherently raster based. Although there are multiple technologies available, including light-emitting diodes (LEDs), liquid-crystal displays (LCDs), and plasma panels, all use a two-dimensional grid to address individual light-emitting elements. Figure C10 shows a generic flat-panel monitor. The two outside plates each contain parallel grids of wires that are oriented perpendicular to each other. By sending electrical signals to the proper wire in each grid, the electrical field at a location, determined by

the intersection of two wires, can be made strong enough to control the corresponding element in the middle plate.

The middle plate in an LED panel contains light-emitting diodes that can be turned on and off by the electrical signals sent to the grid. In an LCD display, the electrical field controls the polarization of the liquid crystals in the middle panel, thus turning on and off the light passing through the panel. A plasma panel uses the voltages on the grids to energize gases embedded between the glass panels holding the grids. The energized gas becomes glowing plasma.

Most projection systems are also raster devices. These systems use a variety of technologies, including CRTs and digital light projection (DLP). From a user perspective, they act as standard monitors with similar resolutions and precisions. Hard-copy devices, such as printers and plotters, are also raster based but cannot be refreshed.

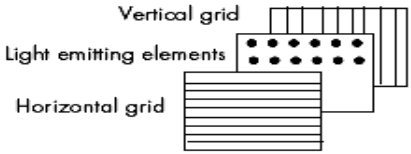
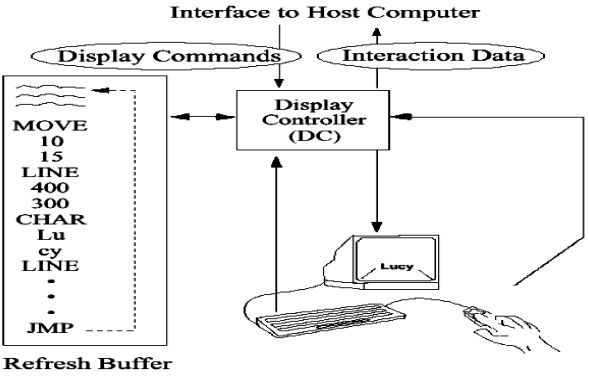


Figure C10: Generic flat-panel display

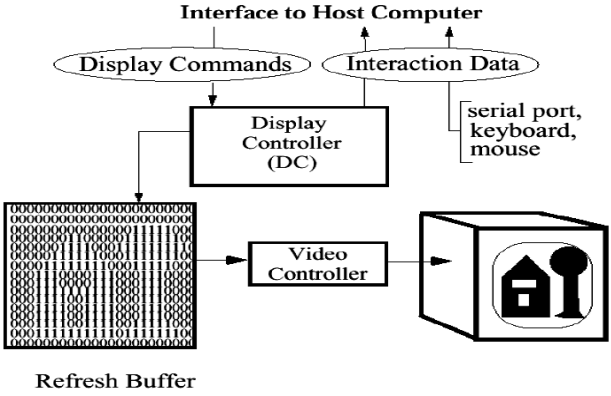
Vector Displays

Oscilloscopes were some of the 1st computer displays.Used by both analog and digital computers. Computation results used to drive the vertical and horizontal axis (X-Y). Intensity could also be controlled (Z-axis). Used mostly for line drawings. Called vector, calligraphic or affectionately stroker displays. Display list had to be constantly updated (except for storage tubes).

Vector Architecture



Raster Architecture



Raster display stores bitmap/pixmap in refresh buffer, also known as bitmap, frame buffer; be in separate hardware (VRAM) or in CPU's main memory (DRAM) Video controller draws all scan-lines at Consistent >60 Hz; separates update rate of the frame buffer and refresh rate of the CRT.

Interfacing between the CPU and the Display

In a simple system, there may be only one processor, the **central processing unit** (**CPU**) of the system, which must do both the normal processing and the graphical processing. The main graphical function of the processor is to take specifications of graphical primitives (such as lines, circles, and polygons) generated by application programs and to assign values to the pixels in the frame buffer that best represent these entities. For example, a triangle is specified by its three vertices, but to display its outline by the three line segments connecting the vertices, the graphics system must generate a set of pixels that appear as line segments to the viewer. The conversion of geometric entities to pixel colors and locations in the frame buffer is known as **rasterization**, or **scan conversion**.

A typical video interface card contains a display processor, a frame buffer, and a video controller. The frame buffer is a random access memory containing some memory (at least one bit) for each pixel, indicating how the pixel is supposed to be illuminated. The depth of the frame buffer measures the number of bits per pixel. A video controller then reads from the frame buffer and sends control signals to the monitor, driving the scan and refresh process. The display processor processes software instructions to load the frame buffer with data.

(Note: In early PCs, there was no display processor. The frame buffer was part of the physical address space addressable by the CPU. The CPU was responsible for all display functions.)

Some Typical Examples of Frame Buffer Structures:

i. For a simple monochrome monitor, just use one bit per pixel.

ii. A gray-scale monitor displays only one color, but allows for a range of intensity levels at each pixel. A typical example would be to use 6-8 bits per pixel, giving 64-256 intensity levels. For a color monitor, we need a range of intensity levels for each of red, green, and blue. There are two ways to arrange this.

iii. A color monitor may use a color lookup table (LUT). For example, we could have a LUT with 256 entries. Each entry contains a color represented by red, green, and blue values. We then could use a frame buffer with depth of 8. For each pixel, the frame buffer contains an index into the LUT, thus choosing one of the 256 possible colors. This approach saves memory, but limits the number of colors visible at any one time.

iv. A frame buffer with a depth of 24 has 8 bits for each color, thus 256 intensity levels for each color. 224 colors may be displayed. Any pixel can have any color at any time. For a 1024x1024 monitor we would need 3 megabytes of memory for this type of frame buffer. The display processor can handle some medium-level functions like scan conversion (drawing lines, filling polygons), not just turn pixels on and off. Other functions: bit block transfer, display list storage.

Use of the display processor reduces CPU involvement and bus traffic resulting in a faster processor. Graphics processors have been increasing in power faster than CPUs, a new generation every 6-9 months.

Example: 10 3E. NVIDIA GeForce FX

- *125 million transistors (GeForce4: 63 million)*
- *128MB RAM*
- *128-bit floating point pipeline*

One of the advantages of a hardware-independent API like OpenGL is that it can be used with a wide range of CPU-display combinations, from software-only to hardware-only. It also means that a fast video card may run slowly if it does not have a good implementation of OpenGL.

Data Structures for Graphics

Requirements for this section in the appendix are good programming skills in C++, Java, C or its likes; an understanding of basic data

structures (linked lists, trees); and a simple knowledge of linear algebra and trigonometry. The mathematical backgrounds of computer science students, whether undergraduates or graduates, vary considerably.
Models, including the geometric objects, lights, cameras, and material properties, are placed in a data structure called a ***scene graph*** that is passed to a renderer or game engine.

The representation structures used for an object model may be either *declarative* or *procedural*. In a ***declarative*** representation, the model is explicitly embedded in a standard computational data structure. In a ***procedural*** scheme, the model is embedded into any convenient computational procedure, such as a formula, implicit equation, or arbitrary code. In the former, data is retrieved by search, indexing, or pointer chasing; in the latter, data is obtained by invoking a procedure with passed parameters or sending a message to an object which then executes a response.

A Cube

We could now describe a cube through a set of vertex specifications. For example, we could use a two-dimensional array of positions

```
point3 faces[6][4];
```

or we could use a single array of 24 vertices

```
point3 cube_vertices[24];
```

where `cube_vertices[i]` contains the *x*, *y*, *z* coordinates of the ith vertex in the list. Both of these methods work, but they both fail to capture the essence of the cube's **topology**, as opposed to the cube's **geometry**. If we think of the cube as a polyhedron, we have an object—the cube—that is composed of six faces. The faces are each quadrilaterals that meet at vertices; each vertex is shared by three faces. In addition, pairs of vertices define edges of the quadrilaterals; each edge is shared by two faces. These statements describe the topology of a six-sided polyhedron. All are true, regardless of the location of the vertices—that is, regardless of the geometry of the object.

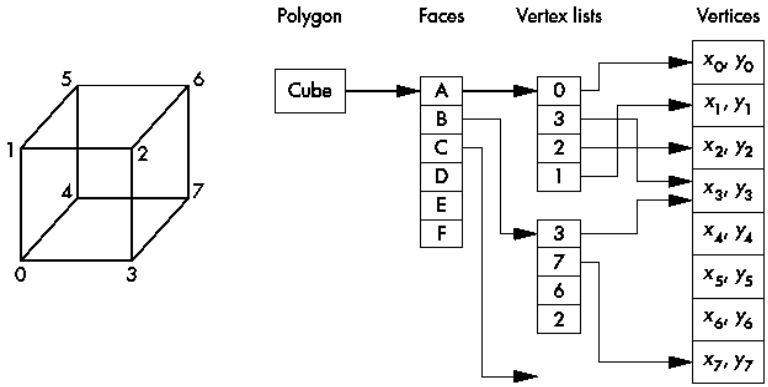


Figure C11: Vertex-list representation of a cube

Throughout this section, we will see that there are numerous advantages to building for our objects data structures that separate the topology from the geometry. In this example, we use a structure, the vertex list that is both simple and useful.

The data specifying the location of the vertices contain the geometry and can be stored as a simple list or array, such as in **`vertices[8]`**—the **vertex list**. The top level entity is a cube; we regard it as being composed of six faces. Each face consists of four ordered vertices. Each vertex can be specified indirectly through its index. This data structure is shown in Figure C11. One of the advantages of this structure is that each geometric location appears only once, instead of being repeated each time it is used for a facet. If, in an interactive application, the location of a vertex is changed, the application needs to change that location only once, rather than searching for multiple occurrences of the vertex.

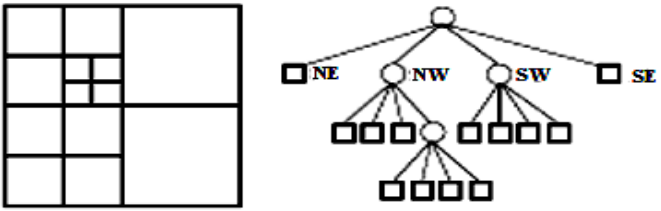
Octrees

Octrees are one of the data structures used for volumetric models that tessellate a given 3D space. The original volume, say a cube, is partitioned into 8 cubes if it is non-empty. Recursively, each sub-cube is partitioned whenever non-empty, until some minimum size element is reached. Since empty cubes are not sub-divided, the storage space efficiency is increased. The major use of octrees appears to be an indexing scheme for access efficiency in a large 3D array.

Quadtrees

A partition of the plane into four quadrants, each of which may be recursively subdivided into quadrants to the desired level of object

detail or pixel size.
There are a variety of data structures possible. Groups of polygons are stored in lists, tables, or linked structures to facilitate traversal of connected faces, the edge network, etc.
A *quadtree* is a rooted tree so that every internal node has four children. Every node in the tree corresponds to a square. If a node *v* has children, their corresponding squares are the four quadrants, as shown



Quadtrees can store many kinds of data. The quadtree consists of a root node *v*, *Q* is stored at *v*. In the following, let *Q*(*v*) denote the square stored at *v*. Furthermore *v* has four children:
The X-child is the root of the quadtree of the set *PX*, where *X* is an element of the set *{NE, NW, SW, SE}*.

Uses
Quadtrees are used to partition 2-D space, while octrees are for 3-D.
The two concepts are nearly identical, and i think it is unfortunate that they are given different names.

Handling Observer-Object Interactions:
Subdivide the quadtree/octree until each leaf's region intersects only a small number of objects.
Each leaf holds a list of pointers to objects that intersect its region.
Find out which leaf the observer is in. We only need to test for interactions with the objects pointed to by that leaf.

- Inside/Outside Tests for Odd Shapes

The root node represents a square containing the shape.

If a node's region lies entirely inside or entirely outside the shape, do not subdivide it.
Otherwise, do subdivide (unless a predefined depth limit has been exceeded).
Then the quadtree or octree contains information allowing us to check quickly whether a given point is inside the shape.

- Sparse Arrays of Spatially-Organized Data

Store array data in the quadtree or octree.

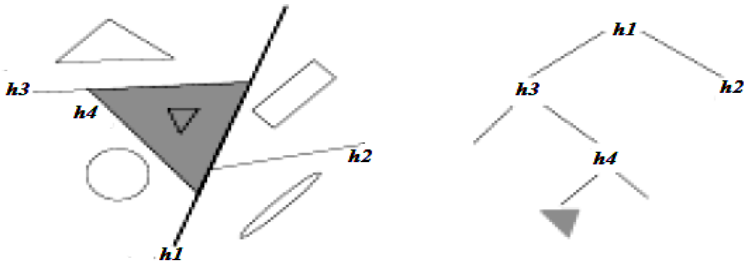
Only subdivide if that region of space contains interesting data.
This is how an octree is used in the BLUI sculpt (virtual reality of shaping figures) program.

K-d-Trees

The *k-d*-tree is a natural generalization of the one dimensional search tree.

BSP Trees

BSP trees (short for binary space partitioning trees) can be viewed as a generalization of *k*-d trees. Like *k*-d trees, BSP trees are binary trees, but now the orientation and position of a splitting plane can be chosen arbitrarily. The figure below depicts the feeling of a BSP tree.



A *Binary Space Partition* tree (BSP tree) is a very different way to represent a scene, Nodes hold facets, the structure of the tree encodes spatial information about the scene. It is useful for HSR (Hidden Surface Removal) and related applications.

Characteristics of BSP Tree

A BSP tree is a binary tree.
Nodes can have 0, 1, or two children.
Order of child nodes matters, and if a node has just 1 child, it matters whether this is its left or right child.

• Each node holds a facet.
  This may be only part of a facet from the original scene.
  When constructing a BSP tree, we may need to split facets.
• Organization:
  Each facet lies in a unique plane. In 2-D, a unique line.
  For each facet, we choose one side of its plane to be the “outside”. (The other direction is “inside”.)
  This can be the side the normal vector points toward.
  Rule: For each node,

• Its left descendant subtree holds only facets "inside" it.
• Its right descendant subtree holds only facets "outside" it.

Construction

• To construct a BSP tree, we need:
A list of facets (with vertices).
An "outside" direction for each.

• Procedure:
Begin with an empty tree. Iterate through the facets, adding a new node to the tree for each new facet. The first facet goes in the root node.
For each subsequent facet, descend through the tree, going left or right depending on whether the facet lies inside or outside the facet stored in the relevant node.
• If a facet lies partially inside & partially outside, split it along the plane [line] of the facet.
• The facet becomes two "partial" facets. Each inherits its "outside" direction from the original facet.
• Continue descending through the tree with each partial facet separately.
Finally, the (partial) facet is added to the current tree as a leaf.

Bounding Volume Hierarchies

Like the previous hierarchical data structures, bounding volume hierarchies (BVHs) are mostly used to prevent performing an operation exhaustively on all objects. Like with previously discussed hierarchical data structures, one can improve a huge range of applications and queries using BVHs, such as ray shooting, point location queries, nearest neighbor search, view frustum and occlusion culling, geographical data bases, and collision detection (the latter will be discussed in more detail below). Often times, bounding volume (BV) hierarchies are described as the opposite of spatial partitioning schemes, such as quadtrees or BSP trees: instead of partitioning space, the idea is to partition the set of objects recursively until some leaf criterion is met. Here, objects can be anything from points to complete graphical objects.

With BV hierarchies, almost all queries, which can be implemented with space partitioning schemes, can also be answered, too. Example queries and operations are ray shooting, frustum culling, occlusion culling, point location, nearest neighbor, collision detection.

Construction of BV Hierarchies

Essentially, there are 3 strategies to build BV trees:

- *bottom-up,*
- *top-down,*
- *insertion*

From a theoretical point of view, one could pursue a simple top-down strategy, which just splits the set of objects into two equally sized parts, where the objects are assigned randomly to either subset.

Asymptotically, this usually yields the same query time as any other strategy. However, in practice, the query times offered by such a BV hierarchy are by a large factor worse.

During construction of a BV hierarchy, it is convenient to forget about the graphical objects or primitives, and instead deal with their BVs and consider those as the atoms.

Color

Color is one of the most interesting aspects of both human perception and computer graphics. Full exploitation of the capabilities of the human visual system using computer graphics requires a far deeper understanding of the human anatomy, physiology, and psychophysics.

A visible color can be characterized by a function $C(\lambda)$ that occupies wavelengths from about 350 to 780 nm, as shown in Figure C12. The value for a given wavelength λ in the visible spectrum gives the intensity of that wavelength in the color. Although this characterization is accurate in terms of a physical color whose properties we can measure, it does not take into account how we *perceive* color.

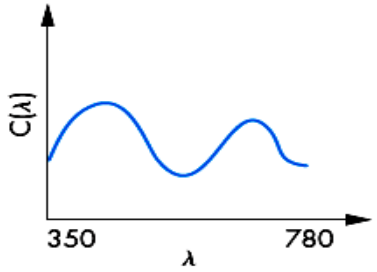


Figure C12: A color distribution

Colour Theory

To get color, people have come up with some strange practices and overcoming difficulties. Our eyes work by focusing light through an elastic lens, onto a patch at the back of our eye called the ***retina***. The retina contains light sensitive rod and cone cells that are sensitive to

light, and send electrical impulses to our brain that we interpret as a visual stimulus. Given this biological apparatus, we can simulate the presence of many colours by shining Red, Green and Blue light into the human eye with carefully chosen intensities. This is the basis on which all colour display technologies (CRTs, LCDs, TFTs (*Thin-film Transistor used in flatscreen displays*), Plasma, Data projectors) operate. Inside our machine (TV, Computer, Projector) we represent pixel colours using values for Red, Green and Blue (RGB triples) and the video hardware uses these values to generate the appropriate amount of Red, Green and Blue light.

Color space

A color model is an abstract mathematical model describing the way colors can be represented as ***tuples*** of numbers, typically as three or four values or color components (e.g. RGB and CMYK are color models).

Adding a certain mapping function between the color model and a certain reference color space results in a definite "footprint" within the reference color space. This "footprint" is known as a gamut, and, in combination with the color model, defines a new color space. For example, Adobe RGB and SRGB are two different absolute color spaces, both based on the RGB model.

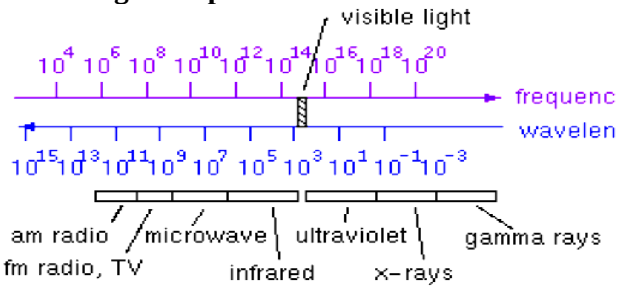
Light

From a physical perspective, a surface can either emit light by self-emission, as a light bulb does, or reflect light from other surfaces that illuminate it. Some surfaces may both reflect light and emit light from internal physical processes. When we look at a point on an object, the color that we see is determined by multiple interactions among light sources and reflective surfaces. These interactions can be viewed as a recursive process.

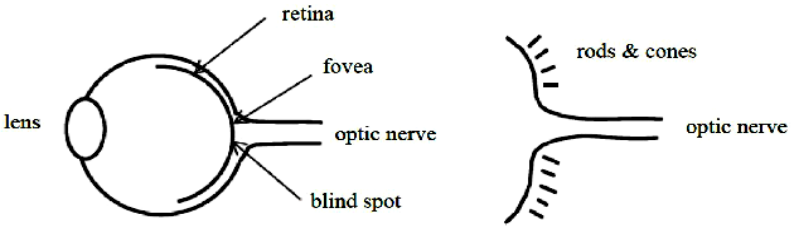
Light as we perceive it is electromagnetic radiation from a narrow band of the complete spectrum of electromagnetic radiation called the visible spectrum. The physical nature of light has elements that are like particle (when we discuss photons) and as a wave. Recall that wave can be described either in terms of its frequency, measured say in cycles per second, or the inverse quantity of wavelength. The electro-magnetic spectrum ranges from very low frequency (high wavelength) radio waves (greater than 10 centimeter in wavelength) to microwaves, infrared, visible light, ultraviolet and x-rays and high frequency (low wavelength) gamma rays (less than 0.01 nm in wavelength). Visible

light lies in the range of wavelengths from around 400 to 700 nm, where nm denotes a nanometer, or 10−9 of a meter.
Physically, the light energy that we perceive as color can be described in terms of a function of wavelength λ, called the spectral distribution function or simply spectral function, f(λ). As we walk along the wavelength axis (from long to short wavelengths), the associated colors that we perceive varying along the colors of the rainbow red, orange, yellow, green, blue, indigo, violet. (Remember the "Roy G. Biv" mnemonic.) Of course, these color names are human interpretations, and not physical divisions.

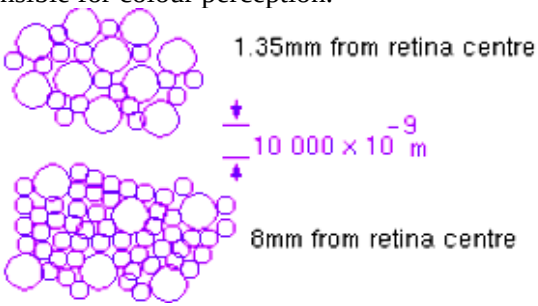
The Electromagnetic spectrum



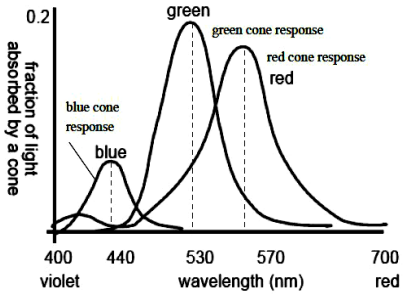
The Retina

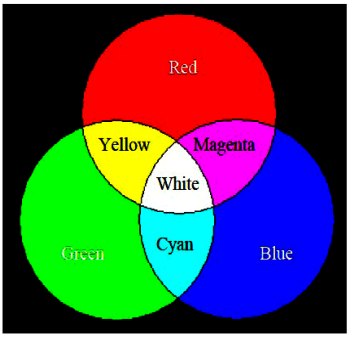


The retina has both rods and cones, as shown below. It is the cones which are responsible for colour perception.



There are three types of cones, referred to either as B, G, and R, or equivalently as S, M, and L, respectively. Their peak sensitivities are located at approximately 430nm, 560nm, and 610nm for the "average" observer. Animals exist with both fewer and more types of cones. The photo-pigments in rods and cones are stimulated by absorbed light, yielding a change in the cell membrane potential. The different types of cells have different spectral sensitivities:





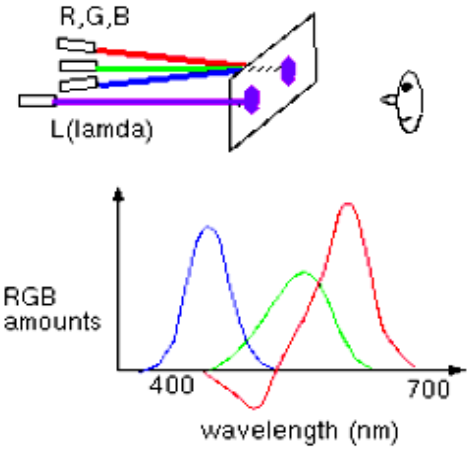
Mapping from Reality to Perception

Different spectra can be perceptually identical to the eye. Such spectra are called ***metamers***. Our perception of colour is related only to the stimulation of three types of cones. If two different spectra stimulate the three cone types in the same way, they will be perceptually indistinguishable.

Colour Matching

In a typical system, there might be a 1280 × 1024 array of pixels, and each pixel might consist of 24 bits (3 bytes): 1 byte for each of red, green, and blue. With present commodity graphics cards having up to 12GB of memory, there is no longer a problem of storing and displaying the contents of the frame buffer at video rates.

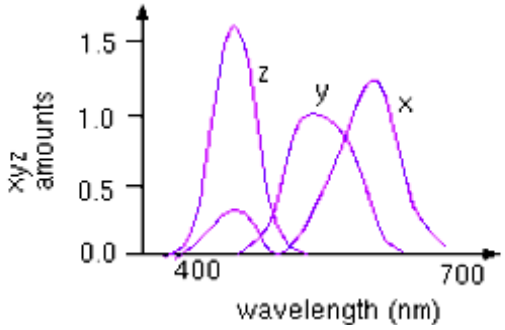
In order to define the perceptual 3D space in a "standard" way, a set of experiments can (and have been) carried by having observers try and match colour of a given wavelength, lambda, by mixing three other pure wavelengths, such as R=700nm, G=546nm, and B=436nm in the following example. Note that the phosphors of colour TVs and other CRTs do not emit pure red, green, or blue light of a single wavelength, as is the case for this experiment.



The above scheme can tell us what mix of R,G,B is needed to reproduce the perceptual equivalent of any wavelength. A problem exists, however, because sometimes the red light needs to be added to the target before a match can be achieved. This is shown on the graph by having its intensity, R, take on a negative value.

In order to achieve a representation which uses only positive mixing coefficients, the CIE ("Commission Internationale d'Eclairage") defined three new hypothetical light sources, x, y, and z, which yields positive matching curves:

If we are given a spectrum and wish to find the corresponding X, Y, and Z quantities, we can do so by integrating the product of the spectral power and each of the three matching curves over all wavelengths. The weights X,Y, Z form the three-dimensional CIE XYZ space, as shown below.



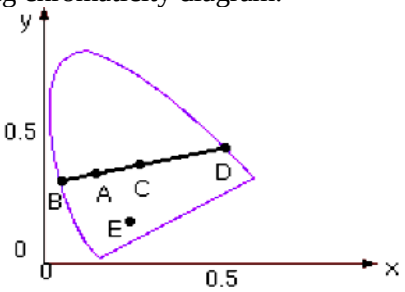
Often it is convenient to work in a 2D colour space. This is commonly done by projecting the 3D colour space onto the plane X+Y+Z=1,

yielding a CIE chromaticity diagram. The projection is defined as given below, and produces the following chromaticity diagram:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

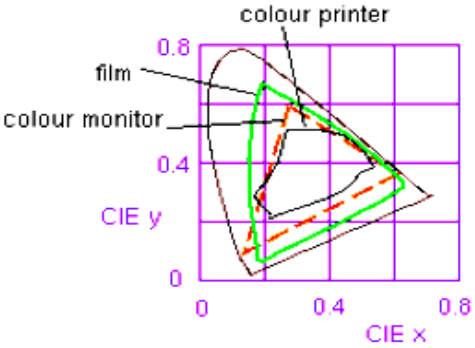
or

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = M \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$



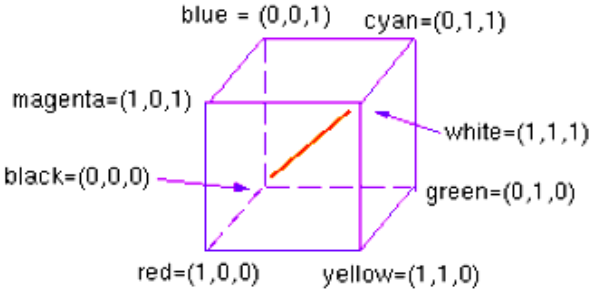
Colour Gamuts

The chromaticity diagram can be used to compare the "gamuts" of various possible output devices (i.e., monitors and printers). Note that a colour printer cannot reproduce all the colours visible on a colour monitor.

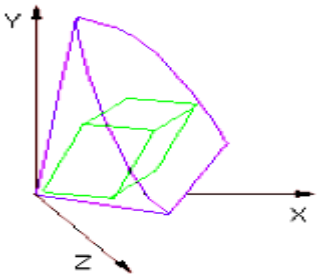


RGB Colour Cube

The additive colour model used for computer graphics is represented by the RGB colour cube, where R, G, and B represent the colours produced by red, green and blue phosphours, respectively.



The colour cube sits within the CIE XYZ colour space as follows.



Colour Printing

Green paper is green because it reflects green and absorbs other wavelengths. The following table summarizes the properties of the four primary types of printing ink.

dye colour	absorbs	reflects
Cyan	red	blue and green
Magenta	green	blue and red
yellow	blue	red and green
Black	all	none

To produce blue, one would mix cyan and magenta inks, as they both reflect blue while each absorbing one of green and red. Black ink is used to ensure that a high quality black can always be printed, and is often referred to as to K. Printers thus use a CMYK colour model.

Colour Conversion

Monitors are not all manufactured with identical phosphors. To convert from one colour gamut to another is a relatively simple procedure (with the exception of a few complicating factors!). Each phosphor colour can be represented by a combination of the CIE XYZ primaries, yielding the following transformation from RGB to CIE XYZ:

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = \begin{bmatrix} X_r & X_g & X_b \\ Y_r & Y_g & Y_b \\ Z_r & Z_g & Z_b \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

or

$$\begin{bmatrix} X \\ Y \\ Z \end{bmatrix} = M \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

The transformation $C_2 = M_2^{-1} M_1 C_1$ yields the colour on monitor 2 which is equivalent to a given colour on monitor 1. Quality conversion to-and-from printer gamuts is difficult. A first approximation is shown on the

left. A fourth colour, K, can be used to replace equal amounts of CMY, as shown on the right.

$K = \min(C,M,Y)$

$C = 1 - R$ $\quad\quad$ $C' = C - K$

$M = 1 - G$ $\quad\quad$ $M' = M - K$

$Y = 1 - B$ $\quad\quad$ $Y' = Y - K$

Other Colour Systems

Several other colour models also exist. Models such as HSV (hue, saturation, value) and HLS (hue, luminosity, saturation) are designed for intuitive understanding. Using these colour models, the user of a paint program would quickly be able to select a desired colour.

Geometry for Computer Graphics

Introduction

In computer graphics, we work with sets of geometric objects, such as lines, polygons, and polyhedra. Such objects exist in a three-dimensional world and have properties that can be described using concepts such as length and angles. Our fundamental geometric object is a point. In a three-dimensional geometric system, a **point** is a location in space. The only property that a point possesses is that point's location; a mathematical point has neither a size nor a shape. Real numbers—and complex numbers, which we will use occasionally—are examples of **scalars**. We need one additional type—the **vector**—to allow us to work with directions.

Physicists and mathematicians use the term *vector* for any quantity with direction and magnitude. Physical quantities, such as velocity and force, are vectors. A vector does not, however, have a fixed location in space.

In computer graphics, we often connect points with directed line segments, as shown in Figure C13a. A directed line segment has both magnitude—its length— and direction—its orientation—and thus is a vector. Because vectors have no fixed position, the directed line segments shown in Figure C13b are identical because they have the same direction and magnitude. We will often use the terms *vector* and *directed line segment* synonymously.

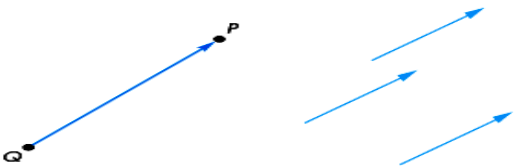


Figure C13:* a. *Directed line segment* *b. Vectors that connects points

Coordinate Geometry

Points exist in space regardless of any reference or coordinate system. Thus, we do not need a coordinate system to specify a point or a vector. This fact may seem counter to your experiences, but it is crucial to understanding geometry and how to build graphics systems. Consider the two-dimensional example shown in Figure C14. Here we see a coordinate system defined by two axes, an origin, and a simple geometric object, a square. We can refer to the point at the lower-left corner of the square as having coordinates *(*1, 1*)* and note that the sides of the square are orthogonal to each other and that the point at *(*3, 1*)* is 2 units from the point at *(*1, 1*)*. Now suppose that we remove the axes as shown in Figure C15. We can no longer specify where the points are. But those locations were relative to an arbitrary location of the origin and the orientation of the axes. What is more important is that the fundamental geometric relationships are preserved. The square is still a square, orthogonal (perpendicular) lines are still orthogonal, and distances between points remain the same.

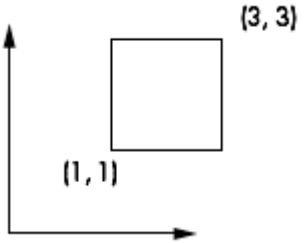


Figure C14: Object and coordinate system

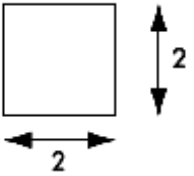


Figure C15: Object without coordinate system

Mathematical View: Vector and Affine Spaces

If we view scalars, points, and vectors as members of mathematical sets, then we can look at a variety of abstract spaces for representing and manipulating these sets of objects. Mathematicians have explored a variety of such spaces for applied problems, ranging from the solution of differential equations to the approximation of mathematical functions. The formal definitions of the spaces of interest to us— vector spaces, affine spaces (linear transformation), and Euclidean spaces (parallel lines are equidistant). We are concerned with only those examples in which the elements are geometric types.

We start with a set of scalars, any pair of which can be combined to form another scalar through two operations, called *addition* and *multiplication*.

If these operations obey the closure, associativity, commutivity, and inverse properties , the elements form a **scalar field**. Familiar examples of scalars include the real numbers, complex numbers, and rational functions.

Perhaps the most important mathematical space is the **(linear) vector space**. A vector space contains two distinct types of entities: vectors and scalars. In addition to the rules for combining scalars, within a vector space, we can combine scalars and vectors to form new vectors through **scalar–vector multiplication** and vectors with vectors through **vector–vector addition**. Examples of mathematical vector spaces include *n*-tuples of real numbers and the geometric operations on our directed line segments.

In a linear vector space, we do not necessarily have a way of measuring a scalar quantity. A **Euclidean space** is an extension of a vector space that adds a measure of size or distance and allows us to define such things as the length of a line segment.

An **affine space** is an extension of the vector space that includes an additional type of object: the point. Although there are no operations between two points or between a point and a scalar that yield points, there is an operation of *vector–point addition* that produces a new point. Alternately, we can say there is an operation called *point–point subtraction* that produces a vector from two points. Examples of affine spaces include the geometric operations on points and directed line segments.

In these abstract spaces, objects can be defined independently of any particular representation; they are simply members of various sets. One of the major vector space concepts is that of representing a vector in terms of one or more sets of basis vectors. Representation provides the tie between abstract objects and their implementation. Conversion between representations leads us to geometric transformations.

Computer Science View

Although the mathematician may prefer to think of scalars, points, and vectors as members of sets that can be combined according to certain axioms, the computer scientist prefers to see them as **abstract data types** (**ADTs**). An ADT is a set of operations on data; the operations are defined independently of how the data are represented internally or of how the operations are implemented. The notion of *data abstraction* is fundamental to modern computer science. For example, the operation of adding an element to a list or of multiplying two polynomials can be defined independently of how the list is stored or of how real numbers

are represented on a particular computer. People familiar with this concept should have no trouble distinguishing between objects (and operations on objects) and objects' representations (or implementations) in a particular system. From a computational point of view, we should be able to declare geometric objects through code such as

```
vector u,v;
point p,q;
scalar a,b;
```

regardless of the internal representation or implementation of the objects on a particular system. In object-oriented languages, such as C++, we can use language features, such as classes and overloading of operators, so we can write lines of code, such as

```
q = p+a*v;
```

using our geometric data types. Of course, first we must define functions that perform the necessary operations; so that we can write them, we must look at the mathematical functions that we wish to implement. First, we will define our objects. Then we will look to certain abstract mathematical spaces to help us with the operations among them.

Geometric ADTs

The three views of scalars, points, and vectors leave us with a mathematical and computational framework for working with our geometric entities. In summary, for computer graphics our scalars are the real numbers using ordinary addition and multiplication. Our geometric points are locations in space, and our vectors are directed line segments. These objects obey the rules of an affine space. We can also create the corresponding ADTs in a program.

Our next step is to show how we can use our types to form geometrical objects and to perform geometric operations among them. We will use the following notation:

- Greek letters $\alpha, \beta, \gamma, \ldots$ denote scalars;
- uppercase letters $P, Q, R, \ldots$ denote points;
- lowercase letters $u, v, w, \ldots$ denote vectors.

We have not yet introduced any reference system, such as a coordinate system; thus, for vectors and points, this notation refers to the abstract objects, rather than to these objects' representations in a particular reference system. We use boldface letters for the latter. The **magnitude** of a vector v is a real number denoted by $|v|$.

The operation of vector–scalar multiplication has the property that

$|\alpha v| = |\alpha||v|$,

and the direction of αv is the same as the direction of v if α is positive and the opposite direction if α is negative.

We have two equivalent operations that relate points and vectors. First, there is the subtraction of two points, P and Q—an operation that yields a vector v denoted by

$v = P - Q$.

As a consequence of this operation, given any point Q and vector v, there is a unique point, P, that satisfies the preceding relationship. We can express this statement as follows: Given a point Q and a vector v, there is a point P such that

$P = Q + v$.

Thus, P is formed by a point–vector addition operation. Figure C16 shows a visual interpretation of this operation. The head-to-tail rule gives us a convenient way of visualizing vector–vector addition. We obtain the sum $u + v$ as shown in Figure C17 (a) by drawing the sum vector as connecting the tail of u to the head of v. However, we can also use this visualization, as demonstrated in Figure C17 (b), to show that for any three points P, Q, and R,

$(P - Q) + (Q - R) = P - R$.

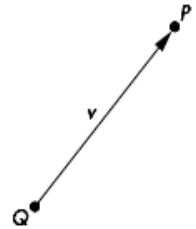


Figure C16: Point–point Subtraction

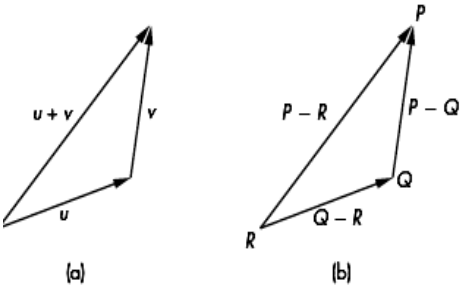


Figure C17: Use of the head-to-tail rule (a) For vectors. (b) For points

Frames in OPENGL

OpenGL is based on a pipeline model, the first part of which is a sequence of operations on vertices, many of which are geometric. We can characterize such operations by a sequence of transformations or, equivalently, as a sequence of changes of frames for the objects specified by an application program.

In versions of OpenGL with a fixed-function pipeline and immediate-mode rendering, six frames were specified in the pipeline. With programmable shaders, we have a great deal of flexibility to add

additional frames or avoid some traditional frames. Although as we demonstrated in our first examples, we could use some knowledge of how the pipeline functions to avoid using all these frames, which would not be the best way to build applications.

Some may not be visible to the application. In each of these frames, a vertex has different coordinates. The following is the usual order in which the frames occur in the pipeline:

i. *Object (or model) coordinates*
ii. *World coordinates*
iii. *Eye (or camera) coordinates*
iv. *Clip coordinates*
v. *Normalized device coordinates*
vi. *Window (or screen) coordinates*

Let's consider what happens when an application program specifies a vertex. This vertex may be specified directly in the application program or indirectly through an instantiation of some object. In most applications, we tend to specify or use an object with a convenient size, orientation, and location in its own frame called the **model** or **object frame**. For example, a cube would typically have its faces aligned with axes of the frame, its center at the origin, and have a side length of 1 or 2 units. The coordinates in the corresponding function calls are in object or model coordinates.

An individual scene may comprise hundreds or even thousands of individual objects. The application program generally applies a sequence of transformations to each object to size, orient, and position it within a frame that is appropriate for the particular application. For example, if we were using an instance of a square for a window in an architectural application, we would scale it to have the correct proportions and units, which would probably be in feet or meters. The origin of application coordinates might be a location in the center of the bottom floor of the building. This application frame is called the **world frame**, and the values are in **world coordinates**. Note that if we do not model with predefined objects or apply any transformations before we specify our geometry, object and world coordinates are the same.

Object and world coordinates are the natural frames for the application program.

However, the image that is produced depends on what the camera or viewer sees. Virtually all graphics systems use a frame whose origin is the center of the camera's lens3 and whose axes are aligned with the sides of the camera. This frame is called the **camera frame** or **eye frame**. Because there is an affine transformation that corresponds to

each change of frame, there are 4 × 4 matrices that represent the transformation from model coordinates to world coordinates and from world coordinates to eye coordinates. These transformations usually are concatenated together into the **model-view transformation**, which is specified by the model-view matrix. Usually, the use of the model-view matrix instead of the individual matrices should not pose any problems for the application programmer.

The last three representations are used primarily in the implementation of the pipeline, but, for completeness, we introduce them here. Once objects are in eye coordinates, OpenGL must check whether they lie within the view volume. If an object does not, it is clipped from the scene prior to rasterization. OpenGL can carry out this process most efficiently if it first carries out a projection transformation that brings all potentially visible objects into a cube centered at the origin in **clip coordinates**. The division by the *w* component, called **perspective division**, yields three-dimensional representations in **normalized device coordinates**. The final transformation takes a position in normalized device coordinates and, taking into account the viewport, creates a three-dimensional representation in **window coordinates**. Window coordinates are measured in units of pixels on the display but retain depth information. If we remove the depth coordinate, we are working with two-dimensional **screen coordinates**.

The application programmer usually works with two frames: the eye frame and the object frame. By concatenating them together to form the model-view matrix, we have a transformation that positions the object frame relative to the eye frame.

Thus, the model-view matrix converts the homogeneous-coordinate representations of points and vectors to their representations in the application space to their representations in the eye frame.

Although an application does not require us to use the model-view matrix, the model-view matrix is so important to most applications.

Let's assume that we allocate a model-view matrix in our applications and initialize it to an identity matrix. Now the object frame and eye frame are identical. Thus, if we do not change the model-view matrix, we are working in eye coordinates.

The camera is at the origin of its frame, as shown in Figure C18. The three basis vectors in eye space correspond to (1) the up direction of the camera, the *y* direction; (2) the direction the camera is pointing, the negative *z* direction; and (3) a third orthogonal direction, *x*, placed so that the *x, y, z* directions form a right-handed coordinate system. We

obtain other frames in which to place objects by performing homogeneous coordinate transformations that specify new frames relative to the camera frame.

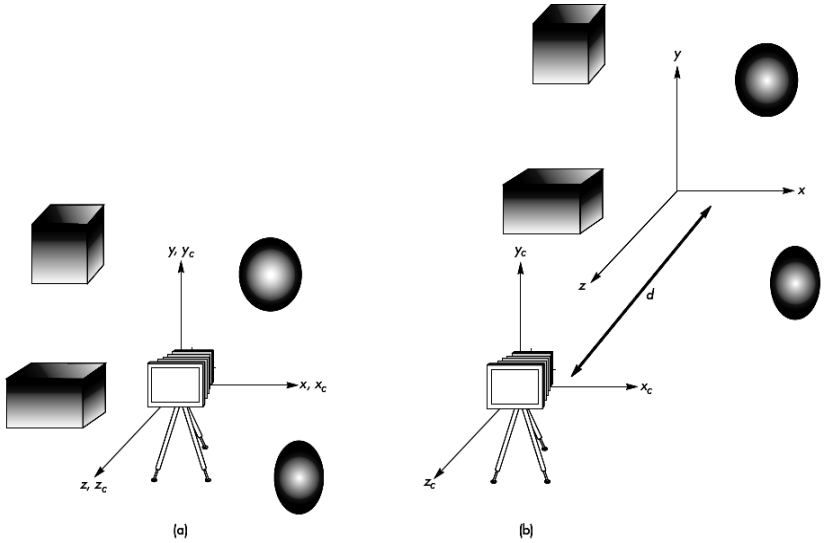


Figure C18: Camera and object frames
(a) In default positions (b) After applying model-view matrix

Because frame changes are represented by model-view matrices that can be stored, we can save frames and move between frames by changing the current model view matrix.

When first working with multiple frames, there can be some confusion about which frames are fixed and which are varying. Because the model-view matrix positions the camera *relative* to the objects, it is usually a matter of convenience as to which frame we regard as fixed. Most of the time, we will regard the camera as fixed and the other frames as moving relative to the camera, but you may prefer to adopt a different view.

Before beginning a detailed discussion of transformations and how we use them in OpenGL, we present two simple examples. In the default settings shown in Figure C18(a), the camera and object frames coincide with the camera pointing in the negative *z*-direction. In many applications, it is natural to specify objects near the origin, such as a square centered at the origin or perhaps a group of objects whose center of mass is at the origin. It is also natural to set up our viewing

conditions so that the camera sees only those objects that are in front of it. Consequently, to form images that contain all these objects, we must either move the camera away from the objects or move the objects away from the camera. Equivalently, we move the camera frame relative to the object frame. If we regard the camera frame as fixed and the model-view matrix as positioning the object frame relative to the camera frame, then the model-view matrix,

$$\mathbf{A} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & -d \\ 0 & 0 & 0 & 1 \end{bmatrix},$$

moves a point *(x, y, z)* in the object frame to the point *(x, y, z − d)* in the camera frame. Thus, by making *d* a suitably large positive number, we "move" the objects in front of the camera by moving the world frame relative to the camera frame, as shown in Figure C18(b). Note that, as far as the user—who is working in world coordinates—is concerned, she is positioning objects as before. The model-view matrix takes care of the relative positioning of the object and eye frames. This strategy is almost always better than attempting to alter the positions of the objects by changing their vertex positions to place them in front of the camera.

Translation, Rotation and Scaling

Translation

Translation is an operation that displaces points by a fixed distance in a given direction, as shown in Figure C19. To specify a translation, we need only to specify a displacement vector *d*, because the transformed points are given by

$P^{\mathrm{l}} = P + d$

for all points *P* on the object. Note that this definition of translation makes no reference to a frame or representation. Translation has three degrees of freedom because we can specify the three components of the displacement vector arbitrarily.

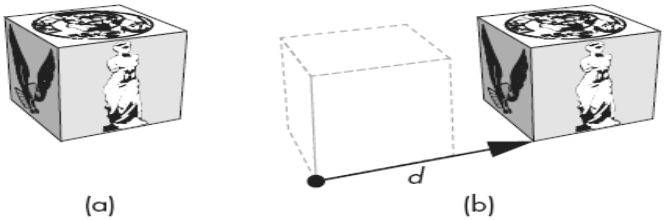


Figure C19: Translation
(a) Object in original position (b) Object translated

Rotation

Rotation is more difficult to specify than translation because we must specify more parameters. We start with the simple example of rotating a point about the origin in a two-dimensional plane, as shown in Figure C20. Having specified a particular point—the origin—we are in a particular frame. A two-dimensional point at *(x, y)* in this frame is rotated about the origin by an angle θ to the position (x^{l}, y^{l}). We can obtain the standard equations describing this rotation by representing *(x, y)* and (x^{l}, y^{l}) in polar form:

$x = \rho \cos \varphi$,
$y = \rho \sin \varphi$,
$x^{l} = \rho \cos(\theta + \varphi)$,
$y^{l} = \rho \sin(\theta + \varphi)$.

Expanding these terms using the trigonometric identities for the sine and cosine of the sum of two angles, we find

$x^{l} = \rho \cos \varphi \cos \theta - \rho \sin \varphi \sin \theta = x \cos \theta - y \sin \theta$,
$y^{l} = \rho \cos \varphi \sin \theta + \rho \sin \varphi \cos \theta = x \sin \theta + y \cos \theta$.

These equations can be written in matrix form as

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}.$$

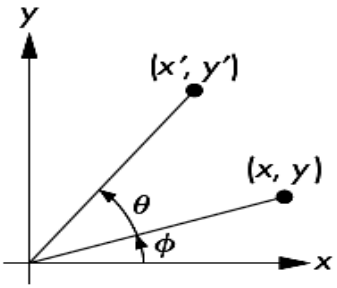


Figure C20: Two-dimensional rotation

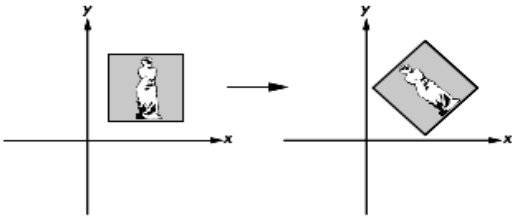


Figure C21: Rotation about a fixed point

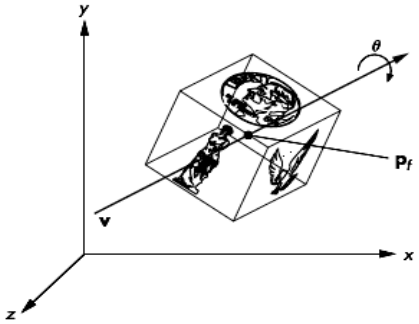


Figure C22: Three-dimensional rotation

Scaling

Scaling is an affine non–rigid-body transformation by which we can make an object bigger or smaller. Figure C23 illustrates both uniform scaling in all directions and scaling in a single direction. We need non-uniform scaling to build up the full set of affine transformations that we use in modeling and viewing by combining a properly chosen sequence of scalings, translations, and rotations.

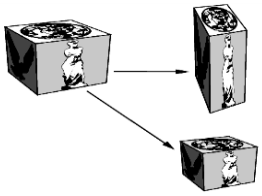


Figure C23: Uniform and non-uniform scaling

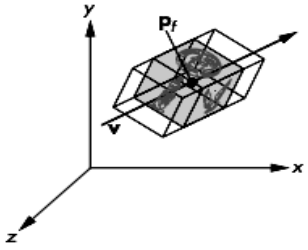


Figure C24: Effect of scale factor

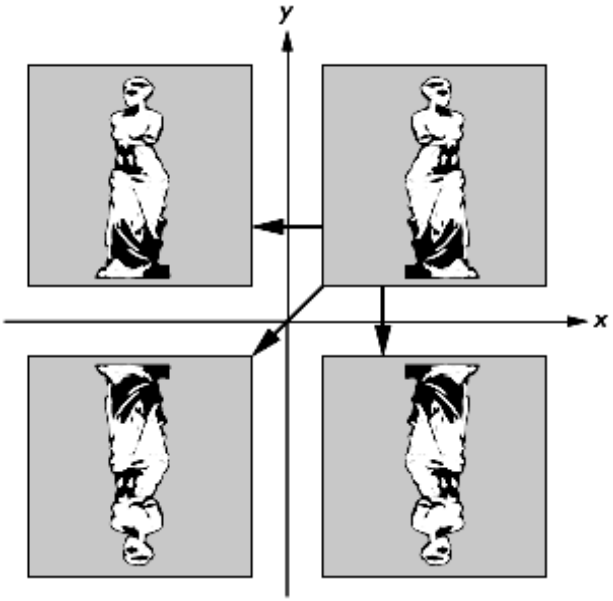


Figure C25: Reflection

Scaling transformations have a fixed point, as we can see from Figure C24. Hence, to specify a scaling, we can specify the fixed point, a direction in which we wish to scale, and a scale factor (α). For $\alpha > 1$, the object gets longer in the specified direction; for $0 \leq \alpha < 1$, the object gets smaller in that direction. Negative values of α give us **reflection** (Figure C25) about the fixed point, in the scaling direction. Scaling has six degrees of freedom because we can specify an arbitrary fixed point and three independent scaling factors.

Interfaces to Three-Dimensional Applications

We can use a three-button mouse to control the direction of rotation of a cube. This interface is limited. Rather than use all three mouse buttons to control rotation, we might want to use mouse buttons to control functions, such as pulling down a menu that we would have had to assign to keys. There were many ways to obtain a given orientation.

Rather than do rotations about the *x*-, *y*-, and *z*-axes in that order, we could do a rotation about the *x*-axis, followed by a rotation about the *y*-axis, and finish with another rotation about the *x*-axis. If we do our orientation this way, we can obtain our desired orientation using only two mouse buttons. However, there is still a problem:

Our rotations are in a single direction. It would be easier to orient an object if we could rotate either forward or backward about an axis and could stop the rotation once we reached a desired orientation.
GLUT (OpenGL Utility Toolkit) allows us to use the keyboard in combination with the mouse. We could, for example, use the left mouse button for a forward rotation about the *x*-axis and the Control key in combination with the left mouse button for a backward rotation about the *x*-axis.
However, neither of these options provides a good user interface, which should be more intuitive and less awkward.

Animation

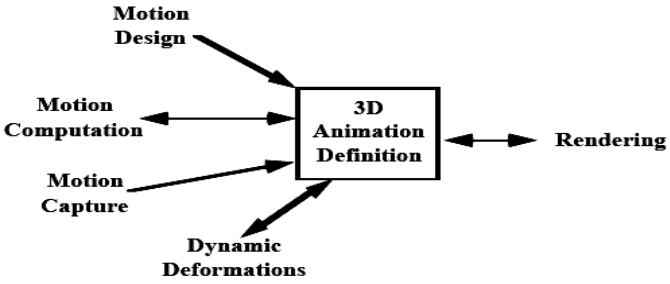
Introduction

Animation is the process of creating and recording images which change over time. Though often interpreted as implying only two-dimensional image changes, it may be applied in general to any model or scene changes in three dimensions as well. In animation, the background and stationary objects are rendered once, while changing or moving objects are rendered on a per frame basis and combined with the static part of the image.

- Articulation
  ~Getting at the part you want
  ~ Getting it to move correctly

-Physics of motion
  Animation by the specification of forces and torques applied to masses, rather than their positions, velocities, and accelerations.

-Constraints
  Constraints are Values, relationships or conditions which are to be maintained while other changes are being made. For example, a constraint can hold a line horizontal, keep two objects a constant distance from one another as they moving, or can cause one object to be attracted to another.



Animating is giving variations on a model over time. Different *keys* given by the animator are interpolated to give *in-betweens.*
Interpolation is the process of filling in information from a set of values. Interpolation can be used to generate plausible surface points for sparse data or for curves or surfaces defined with a small set of control points. It is also a way to generate positions for objects between key frames in an animation.

-Film: 24 frames/second
~30 minutes = 43,200 frames
~ More for TV!
-Motion shape

Concepts
Animation objects may be two-, three-, or even multi-dimensional while their form may be represented in numerous different ways in the computer program. Images are usually two-dimensional, but may have color, depth, and compositing information associated with each pixel. A sequence of images comprises an animation, whether recorded on film or video, or simply viewed interactively. The control of an animation is based on the manipulation of parameters of the image or object. Parameters may describe image background, compositing operations, and layout; or object shape, color, texture, position, orientation, motion paths, and so on. As the parameters change over time, the corresponding attributes of the image or object change to produce the animation.

Levels
-Low level: Individual frames.
- Medium level: Sequences and scenes.
- High level: Story and Message.
Computer helps all three levels

2D Animation
-Hand-drawn *cels*
- Stacks of cels over background
- Only redraw cels that change
Limited animation
Animation produced by stacking multiple 2D drawings (called **cels**), each containing a fragment of a scene or character on a transparent background. In a physical environment the cels are lit and photographed to create one frame of animation. Cels are basically a labor-saving device.

If a character is standing still and talking, then a cel of the character without a mouth might be placed over a background. Then for each frame a cel containing the appropriate mouth image is placed on top of the stack, avoiding the necessity to re-draw the entire character for every frame. Often there are several independent planes of images which may be superimposed in a so-called "two and a half-dimensional" animation.
What gets interpolated?
Strokes, Outlines and Colors.
-3D relationships hard to judge
-Timing based on exposure sheets
- High-quality compositing

3D Animation
-Shoot individual frames with camera
What gets interpolated?
Shape geometry, Shape appearance, Light source information, Cameras and *Anything.*
-Model transformed then rendered

Animation Techniques
The principal task of the animator is to select and control the "proper" parameters to produce the desired effect. Certain applications may dictate a more image-based control approach, while others may require more explicit object models.
Animation means giving life to any object in computer graphics. It has the power of injecting energy and emotions into the most seemingly inanimate objects. Computer-assisted animation and computer-generated animation are two categories of computer animation. It can be presented via film or video.
Animation can make a series of dead images come alive. Animation can be used in many areas like entertainment, computer aided-design, scientific visualization, training, education, e-commerce, and computer art.

Animators have invented and used a variety of different animation techniques and some of them are discussed here.

Constraints
The use of end effector goals and inverse kinematics leads naturally to a desire to specify other types of goals for objects and articulated figure parts.

Since specifying several goals may easily lead to over-constrained systems (more constraints than degrees of freedom), a solution is usually obtained by some "non-procedural" minimization or iterative process. The constraints may take several forms, including relationships, boundary conditions, formulas, potential functions, or spring forces.

Scripting Systems

A scripting system is essentially a programming system or language in which arbitrary changes to program variables can be invoked. The time dimension may be provided explicitly by the order of execution of the program, but more frequently the parameter changes are given times or temporal relationships and then posted in an event list for simulation-style execution. Alternatively, the object behaviors may be encapsulated in an object-oriented programming paradigm. Scripts may be either language or graphics-based and offer an animation interface apparently rather like a series of director commands.

Traditional Animation (frame by frame)

Traditionally most of the animation was done by hand. All the frames in an animation had to be drawn by hand. Since each second of animation requires 24 frames (film), the amount of efforts required to create even the shortest of movies can be tremendous.

Parametric Interpolation

In parametric interpolation, desired attributes of the object model are parameterized so that they may be altered over time. As noted above, parameters may include object geometry, topology, color, surface attributes, textures, position, and orientation. Light sources and their attributes, camera shots, and other image features may also be parametrically specified and changed. The parameters are given key values at various time points and linear or smoothly varying curves are fit to the data points and interpolated to determine in-between values. Motion of an object along a given path is a good example of parametric interpolation.

Parameter values may be determined by direct measurement of some real motion, by algorithm, or by user determined key values.

Key framing

In this technique, a storyboard is laid out and then the artists draw the major frames of the animation. Major frames are the ones in which prominent changes take place. They are the key points of animation. Key framing requires that the animator specifies critical or key

positions for the objects. The computer then automatically fills in the missing frames by smoothly interpolating between those positions.

Procedural

In a procedural animation, the objects are animated by a procedure - a set of rules - not by key framing. The animator specifies rules and initial conditions and runs simulation. Rules are often based on physical rules of the real world expressed by mathematical equations.

Behavioral

In behavioral animation, an autonomous character determines its own actions, at least to a certain extent. This gives the character some ability to improvise, and frees the animator from the need to specify each detail of every character's motion.

Performance Based (Motion Capture)

Another technique is Motion Capture, in which magnetic or vision-based sensors record the actions of a human or animal object in three dimensions. A computer then uses these data to animate the object.

This technology has enabled a number of famous athletes to supply the actions for characters in sports video games. Motion capture is pretty popular with the animators mainly because some of the commonplace human actions can be captured with relative ease. However, there can be serious discrepancies between the shapes or dimensions of the subject and the graphical character and this may lead to problems of exact execution.

Physically Based (Dynamics)

Unlike key framing and motion picture, simulation uses the laws of physics to generate motion of pictures and other objects. Simulations can be easily used to produce slightly different sequences while maintaining physical realism. Secondly, real-time simulations allow a higher degree of interactivity where the real person can maneuver the actions of the simulated character.

In contrast the applications based on key-framing and motion select and modify motions form a pre-computed library of motions. One drawback that simulation suffers from is the expertise and time required to handcraft the appropriate controls systems.

Key Frame

A key frame is a frame where we define changes in animation. Every frame is a key frame when we create frame by frame animation. When someone creates a 3D animation on a computer, they usually don’t

specify the exact position of any given object on every single frame. They create key frames.
Key frames are important frames during which an object changes its size, direction, shape or other properties. The computer then figures out all the in-between frames and saves an extreme amount of time for the animator.

Image Interpolation and Morphing

In image interpolation the two-dimensional drawing of an object's boundary and features is transformed over time by moving (interpolating) points of the drawing between specified positions of those points in a sequence of keyframes.
The transformation of object shapes from one form to another form is called morphing. It is one of the most complicated transformations. A morph looks as if two images melt into each other with a very fluid motion. In technical terms, two images are distorted and a fade occurs between them.

Artificial Intelligence control

Since there is no clear advantage in all situations to any of the above methods, there may be a need to organize several techniques to accomplish particular animation tasks. For example, human animation may require kinetics, dynamics, constraints, and a process model for the task being performed. While such systems are still evolving, several advantages are already apparent: general world model information, object interactions, rule-based action description, motion planning, and simulatable models. This approach may provide the only effective way to deal with the vast numbers of parameters involved to complex animations.

Virtual Reality

Introduction

Computer graphics, engineers have strived to develop more realistic, responsive, and immersive means for the human to interact with the computer this is often called *virtual reality* (or simply *VR*).
There is no one definition of VR everyone will agree on; there is at least one common stand. VR goes beyond the flat monitor that you simply look at, and tries to immerse you in a three dimensional visual world. The things you see appear to be in the room with you, instead of stuck on a flat area like a monitor screen. As you might imagine, there are a number of techniques for achieving this, each with its own trade-off

between degree of immersion, senses involved beyond sight, computational requirements, physical constraints, cost, and others.
VRML (Virtual Reality Modelling Language) was similar to HTML but allowed users to create 3D worlds; it had some academic popularity but never gained attraction with general users.

VR Systems

Stereo Viewing

An important aspect of VR is that the things you look at are presented in *three dimensions*.
Humans see the world around them in 3D using many different techniques. Probably the most powerful technique for nearby (a few tens of meters) objects is called *stereoscopic*.
In stereoscopic 3D perception, we use the fact that we have two eyes that are set some distance apart from each other. When we look at a nearby object we measure the difference in angle from each eye to the object. Knowing this angle, the distance between the eyes, and a bit of trigonometry, we compute the distance to the object. Fortunately this is done for us sub-consciously by an extremely sophisticated image processor in our brain. We simply perceive the end result as objects at specific locations in the space around us.
So, one way to make a 3D computer graphics display is to render two images from slightly different eye points, then present them separately to each eye. Once again there are several ways of achieving this.

Shutter Glasses

Probably the simplest way of displaying stereo computer images is by using the existing monitor. Suppose the display alternates rapidly between the left and right eye images. We could then make sure each eye only saw the image intended for it by opening a shutter in front of the eye when its image is being displayed. The shutters would have to be synchronized to the display.
This is exactly what shutter glasses are. They typically use electronic shutters made with liquid crystals. Another variation places the shutter over the monitor screen.
Instead of blocking or unblocking the light, this shutter changes the light polarization between the left and right eye images. You can now wear passive polarizing glasses where the polarizer for each eye only lets thru the image that was polarized for that eye.

Figure C26: Shutter Glasses and Controller

Head Mounted Display

Another way to present a separate image to each eye is to use a separate monitor for each eye. This can be done by mounting small monitors in some sort of head gear.

With the right optics, the monitors can appear large and at a comfortable viewing distance. This setup is usually referred to as a *head mounted display*, or HMD for short.

Figure 27: Head Mounted Display in Use

Head Tracking

What if the computer could sense the position and orientation of your head in real time?

Assuming we have a sufficiently fast computer and a head mounted display (that's where the position/orientation sensor is hidden), we could re-render the image for each eye in real time also, taking into account the exact eye position. Objects could then be defined in a fixed space. As you moved your head around, the eye images would update to present the illusion of a 3D object at a fixed location in the room, just like real objects.

Figure C28: Position and Orientation Sensors

Hand Tracking

We can use more motion sensors and track the position and orientation of other objects, like your fingers, for example. Just like a mouse or joystick can be used to control a program, your finger actions could be used to control a program.

This might take the form of pushing virtual menu buttons, or maybe grabbing an object and moving it around with your hand. The possible interactions are virtually boundless, limited mostly by the software designer's imagination. Hand and finger position and orientation is typically achieved by wearing a special glove that has a position sensor on it and can sense the angles of your finger joints. This can be taken a step further by wearing a whole suite with imbedded joint angle sensors.

Figure C29: Glove that senses joint angles

Force Feedback

One can be in the same space with the objects we're viewing and interact with them thru hand motions, but we can't feel them. That's where force feedback comes in. This is also referred to as *haptic feedback*. Suppose the special glove (or body suit) you were wearing

could not only sense joint angles, but also had actuators that could push back at you.

With some clever software and fast computers, the actuators could present the illusion of hard objects at particular locations. You can now not only see it in 3D, walk around it, control it, but also bump into it.

Note that force feedback is currently limited to "pushing back" to simulate the existence of a object. It does not provide other parts of what we call tactile feel, like texture, temperature, etc.

Figure C30: Haptic Feedback Glove

Applications

Entertainment

This is definitely the biggest market, and is the main force for driving down prices on VR hardware. You can be in a computer game with computer generated players and/or other real players.

Augmented Reality

Imagine a VR head mounted display as we've discussed, but the display doesn't block out the regular view, it's just superimposed on it. Imagine walking around a building and "seeing" inside the walls to the wiring, plumbing, and structure. Or, seeing the tumor inside a patient's head as you hack away at it.

Training

VR is already being used in to teach people how to use expensive equipment, or when the cost of a mistake in Real Reality is very high. For example, use of VR is getting more common in aircraft simulators used as part of an overall program to train pilots. The benefits are also substantial for military applications for obvious reasons.

Remote Robotics

This is another "real" application that is gaining much attention. Suppose you had a robot that had arms and hands modelled after those

of humans. It could have two video cameras where we have eyes. You could be wearing a head mounted display and see what the robot sees in real time. If your head, arm, and hand motions are sensed and replicated in the robot, for many applications you could be where the robot is without really being there.
This could be useful and worth all the trouble in situations where you can't physically go, or you wouldn't be able to survive. Examples might be in hostile environments like high radiation areas in nuclear power plants, deep undersea, or in outer space. The robot also doesn't need to model a human exactly.

Distributed collaboration
VR is being employed to allow geographically distributed people to do more together than simply hear and see each other as allowed by telephone or videoconferencing.
For example, the military is using VR to create virtual battles. Each soldier participates from his own point of view in an overall simulation that may involve thousands of individuals. All participants need not be in physical proximity, only connected on a network.
This technology is still in its infancy, but evolving rapidly. I expect commercial applications of distributed collaboration to slowly gain momentum over the next several years.

Visualization
Scientists at NASA/Ames and other places have been experimenting with VR as a visualization research tool. Imagine being able to walk around a new aircraft design as it's in a simulated mach 5 wind tunnel (speed of tunnel). VR can be used to "see" things humans can't normally see, like air flow, temperature, pressure, strain, etc.

Problems
The current state of VR is far from everything we could imagine or want. A few of the open issues are:

Cost
This stuff is just too expensive for everyone to have one, and it's likely to stay that way for quite a while.

Importance
I listed some application areas above, but note that none of them solve common everyday problems. While VR certainly has its application niches - and the number is steadily growing- it's hard to imagine how it can help the average secretary type a letter on a word processor.

Display Resolution

Head mounted displays need to be small and light else you get a sore neck real fast.

Unfortunately, the display resolution is therefore limited. Most displays are only about 640 pixels across, which is a tiny fraction of what a normal human can see over the same angle of view.

Update Speed

Most VR displays are updated at 30 Herz (30 times per second). This requires a large amount of computation, just to maintain what looks and feels like "nothing is happening".

The amount of computation required also depends on the scene complexity. VR is therefore limited to relatively "simple" scenes that can be rendered in 1/30 second or faster. This currently precludes any of the rendering methods that can provide shadows, reflections, transparency, and other realistic lighting effects. As a result, VR scenes look very "computer-ish".

Photoshop –*Practical Session*

Introduction

In the Middle to Late '80's, Postscript was developed by John Warnock and Adobe was formed. Photoshop was marketed by Adobe.

Adobe Photoshop is a vast program and it will take an enormous time and pages to treat the content of every function. This appendix provides a tour of the Photoshop user interface and covers some of the basics function of Adobe Photoshop, in a general overview. This will provide a good foundation that once can build on.

In Adobe Photoshop, there are often multiple ways of accomplishing the same task. It is not necessary for you to learn the 5 or 6 different ways there are in the Photoshop for creating a new layer. You only need to remember the one method that fits with the way you prefer to work. This may be a keyboard shortcut, right clicking, menus or icons, its completely up to you how to utilize the tools that are available to you.

Photoshop Panels and Tools

Photoshop is modular in it is layout and it is infinitely customizable. In the default configuration, the panels are located on the right hand side,

the tools are on the left and the options bar which displays the most useful parameters for a selected tool is across the top of the work area. There is also a standard menu bar at the very top of the application.

Workspaces

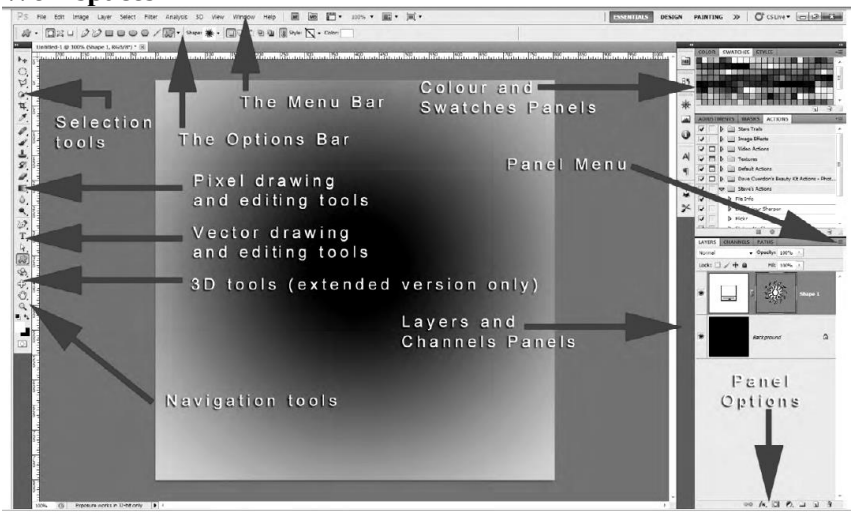


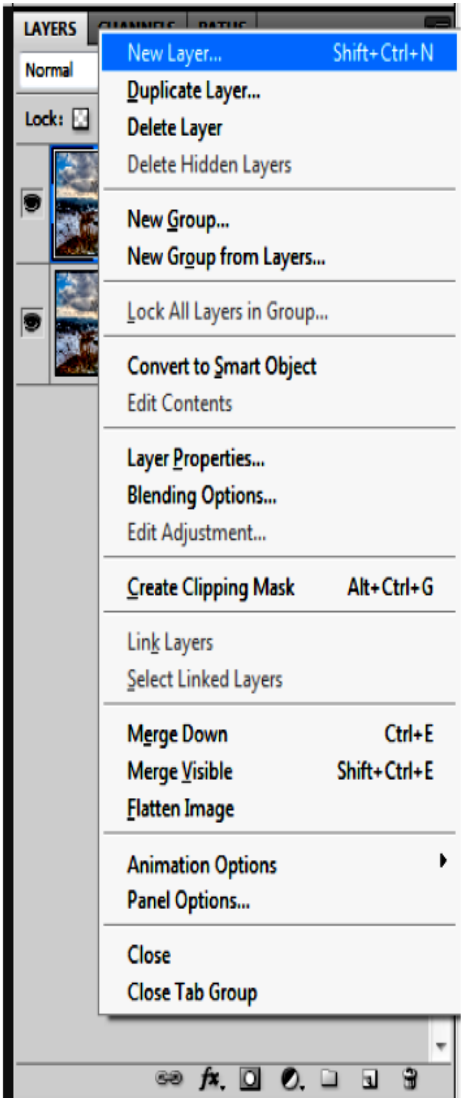
Figure CA.1: the default Photoshop workspace layout

All of the program elements, with the exception of the menu bar, can be dragged around and docked with other panels to create your own custom work area called a workspace. There a number of pre-set workspaces that can be selected from the right hand side of the menu bar, the default workspaces is "***Essential***" and can be recall at any time by clicking on the button. This is very useful because beginners will often accidently close or collapse panels and cannot remember how to restore them.

Figure CA.2: a close up of the workspaces available from the top right corner of the menu bar

If you cannot see all of the workspace options shown in ***Figure CA.2*** they can be accessed by clicking on the >> icons. Alternatively, they can be accessed from the ***Window menu > Workspace***.

Each open panel has its own fly-out menu that contains commonly used functions. The menu can be accessed by clicking on  the icon at the top right of the respective panel.



Some panels are often represented by a strip of icons docked on the left hand side of the open panels. The full panel can be revealed by simple clicking on the icon. However, it is not always clear to beginners what these icons mean. The solution to this is to left-click with your mouse and drag on the left hands side of the icon strip. This will expand the icon strip to reveal the name of the function associated with that icon.

Figure CA.3: an example of a fly-out menu, in the case, accessed from the layers panel

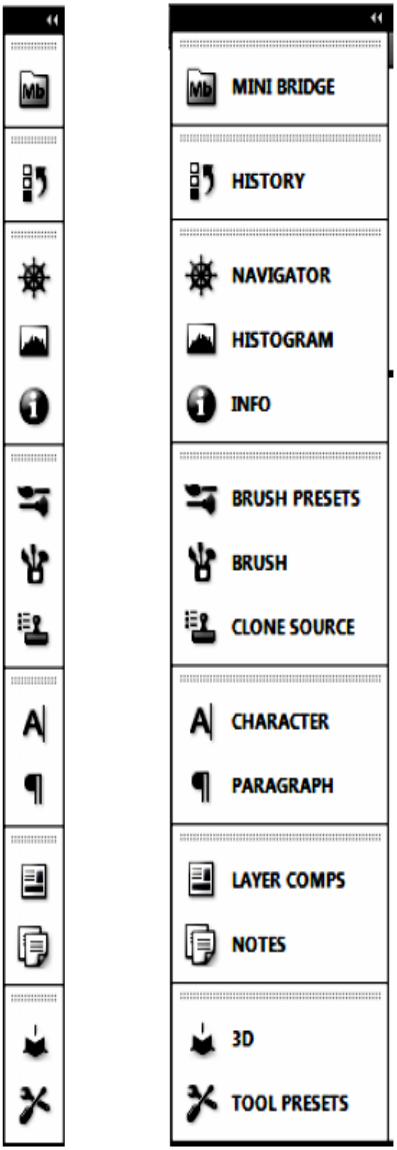


Figure CA.4: Panel icons

Tool Bar

The tool bar (or tool panel) is located by default on the left hand side of the work area. It contain mouse based tool that are used for editing and navigation in Photoshop. Most of the icons have a small black down-pointing arrowhead in the bottom right hand corner. This indicates that

these are more tools that can be accessed by clicking and holding down the mouse button. Once the extended tools have appeared, you may release the mouse button and the tool will remain.

The tools in the tool bar are loosely grouped according to their functionally. You may notice that there are small lines or spacers separating the groups of tools. The first group of tools are used for creating selection, the second group are the pixel editing tools, the third group are the vector editing tools and the final group are the navigation tools. At the bottom of the tool bar there is colour picker and very bottom is an icon that allows you to enter the quick mask mode for creating and editing selections.

Figure CA.5: an example of multiple tools, available from a single tool icon

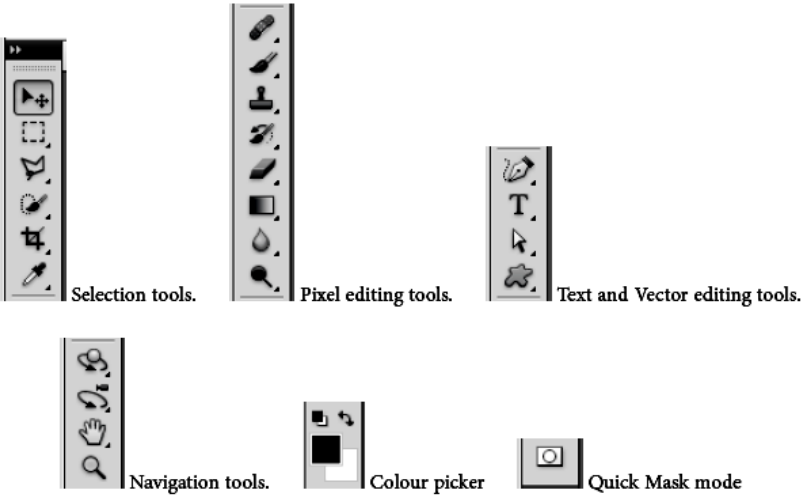


Figure CA.6: tool groupings

Options Bar

The option bar is a context sensitive panels. Which to say, that its changes depending on which tools is selected in the tool bar. It provides access to the most important configuration settings for a particular tool. For example, in fig CA.7 the options bar has changed to provide the attributes associated with the move tool. We have the show transform control and a range of alignment options, which are faded in appearance, because is the case, no layer was highlighted in the layers panel.

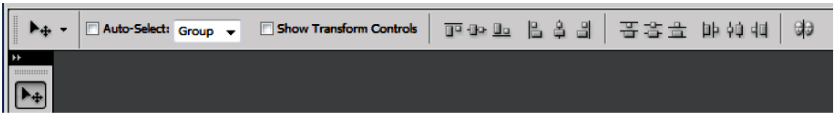
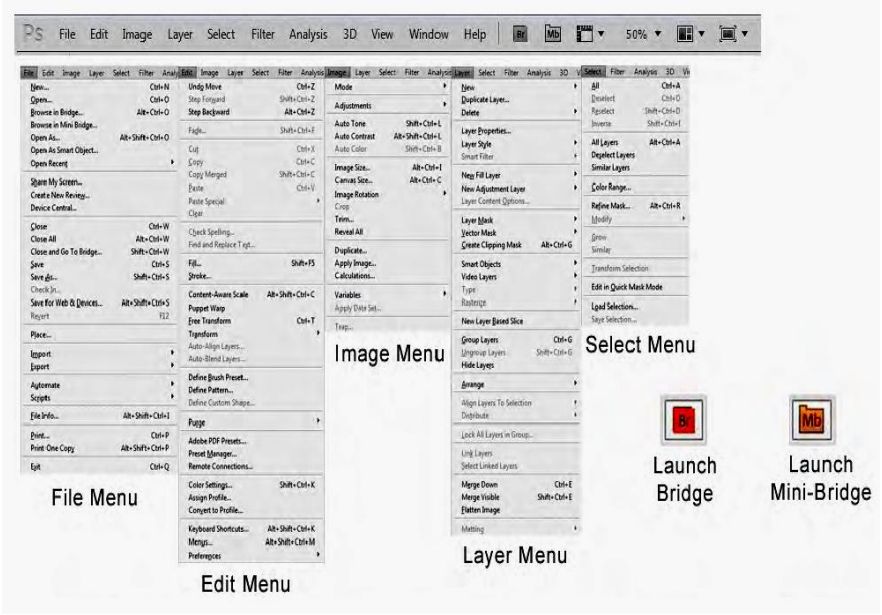


Figure CA.7: the options associated with the move tool

Menu Bar



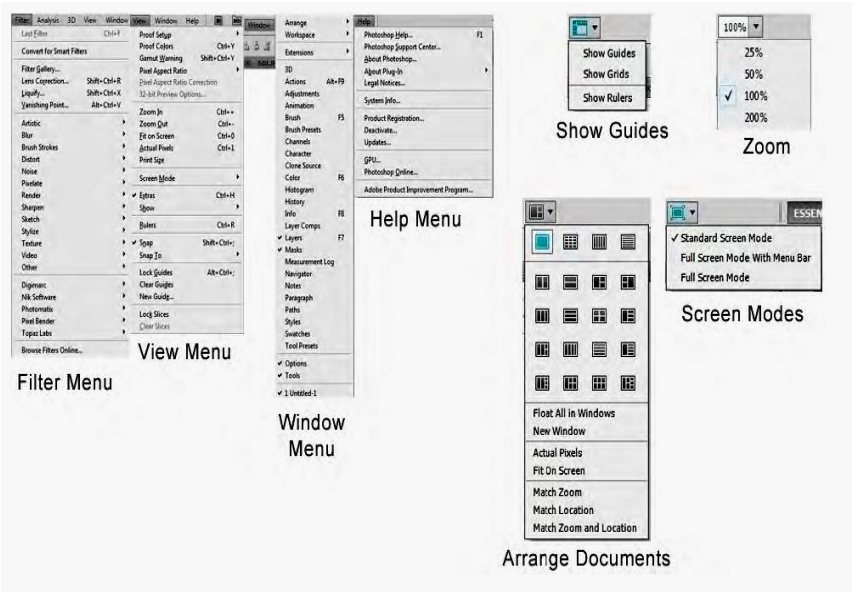


Figure CA.8: all of the menus and items located in the menu bar Photoshop standard edition

The menu bar contains menu items and many sub menu items. The most important functions in the menu bar have assigned keyboard shortcuts which are customisable from the keyboard shortcuts section of the edit menu. Many of the functions in these menus are accessible from the fly-out menus in the various panels. The options to work with keyboard shortcuts, the menu bar or the fly-out menus is one of personal choice and is down to a particular workflow, or may be determined by the type of input device you are using -Mouse, graphics tablet or track pad etc.

Basic Operations

Opening Files

Open

As with most programs, the options to open a file can found in the file menu fig CA.9. This operation also has a keyboard shortcuts ctrl-O/cmd-O (PC/Mac).

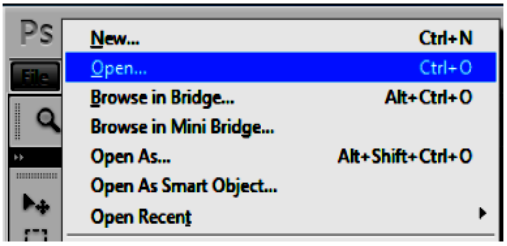


Figure CA.9: Opening a file form the file menu

Once you've selected to open a file, the Open dialogue box will appear. From here you can navigate to your file and images, fig CA.10.

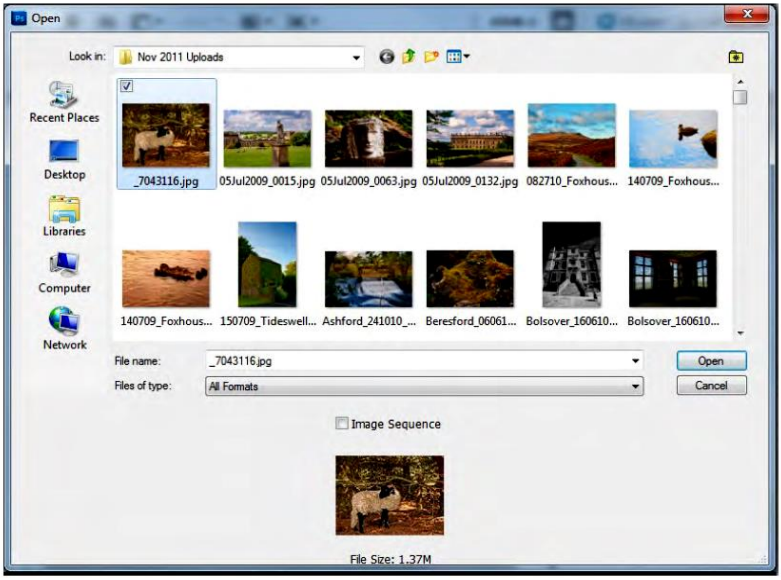


Figure CA.10: open dialogue box

Open As

In addition to the standard open option in the file menu, there is also choice to Open As. The Open As dialogue box is almost identical to the Open dialogue box, except that is allows you open image in one format, fig CA.11.

A useful application of the Open As command is that you can open most single layered file format as a Camera Raw document. This will open the image into Camera Raw plugin (fig CA.12) that is part of Adobe Photoshop. From here you can quickly and easily make simple adjustments to your image without having to get in to complex editing techniques in Photoshop.

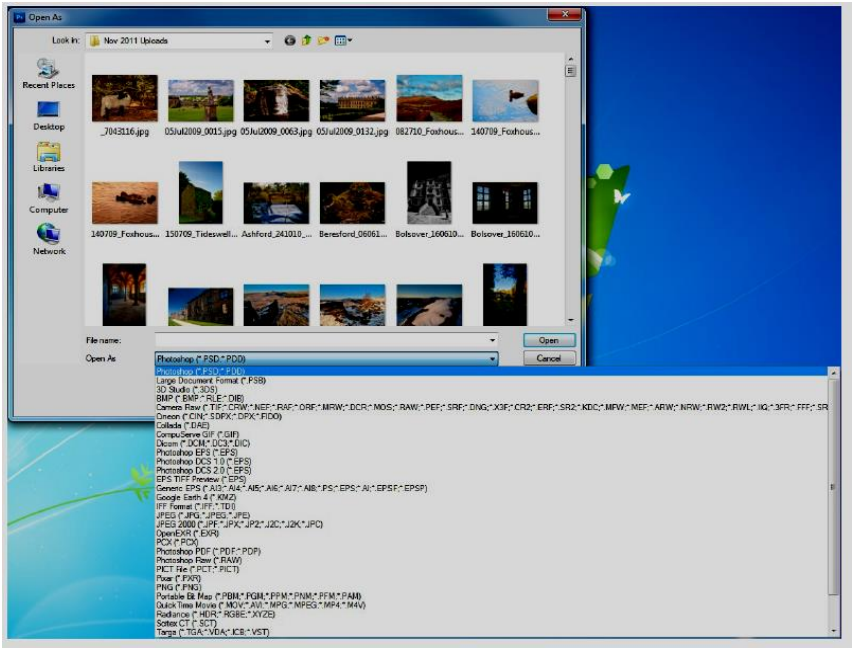


Figure CA.11: Open As dialogue box, showing some of the many available file formats

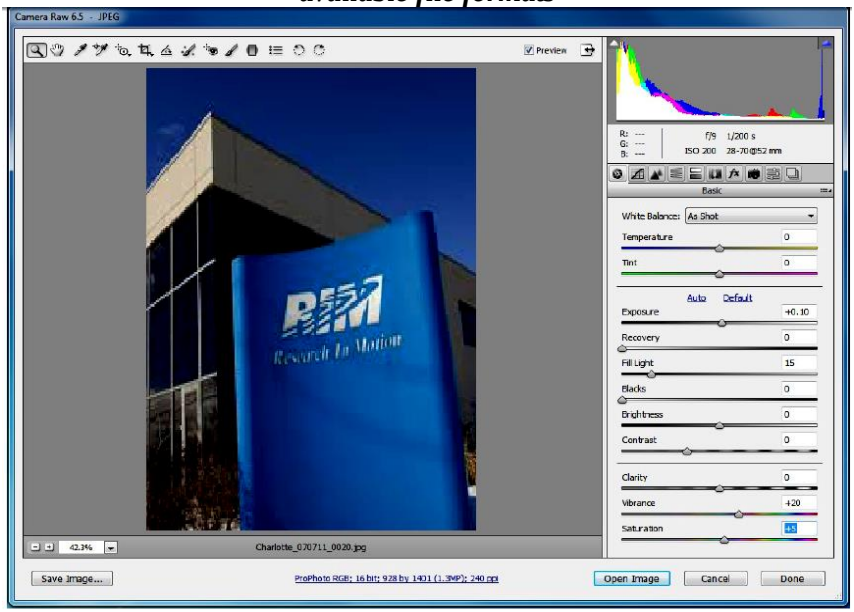


Figure CA.12: Camera Raw

Open As Smart object

The third option is to use Open As Smart Object. A Smart Object allows non-destructive editing of your image and is useful if you are planning on making extreme edits or multiple transformations, such as scaling, to your image. Smart objects appear in the layers and panels with a small icon in the bottom right hand corner and if you apply a filter to them, they will display the filter below the layer fig CA.13. The application of a filter to a smart object is not permanent and the settings can be changed at any time by double clicking on the filter name in Layers Panel.

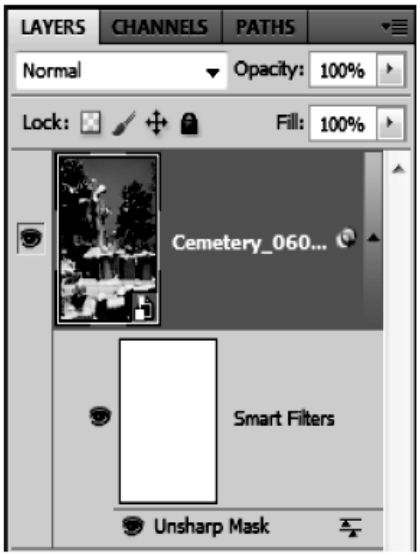


Figure CA.13: a Smart Object

There are several ways of getting an image in to Photoshop, including simple dragging an image on to the work area. You can also use one of the two file browsing options, Bridges and Mini-Bridge.

Saving your work

When you need to save your work you will go to the File menu and choose Save or Save As.

Save: If the file has been saved previously, the file will update. If the document has not been previously saved you will see the same dialogue box As if you had pressed Save As.

Save As: This command brings up a dialogue box (fig CA.14) where you can name the file, choose the properties that you wish to be included in the file and also choose the file format.

File Formats

When you click on the Format drop down menu in the "Save As" dialogue box (fig CA.14) you will notice that Photoshop allows you to save your file in many different formats. Some of these formats are now redundant and are only there to provide backwards compatibility: others are specialist format used in areas such as medical imaging. In reality you only need to know a few of these formats - the number depends on the type of work you are doing.

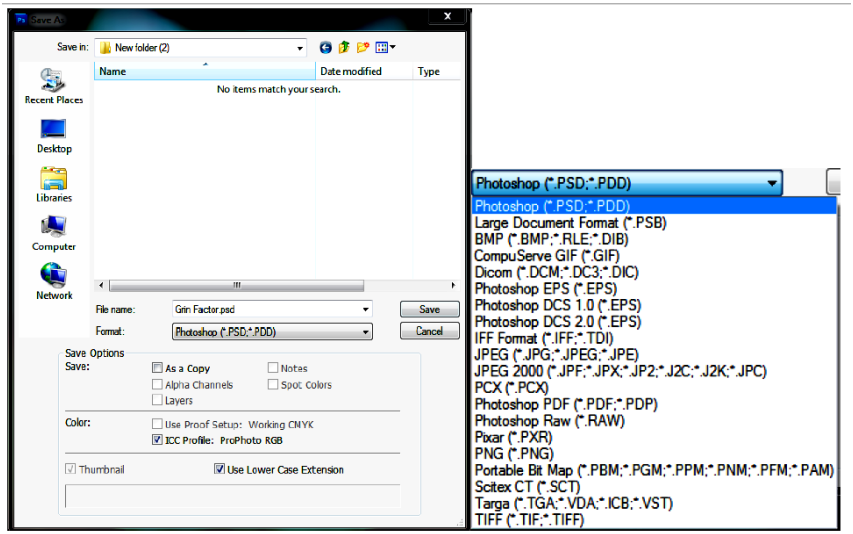


Figure CA.14: the Save As dialogue box and file formats

Popular and Useful File Formats:

PSD: This is Photoshop's native file format and if you only use Photoshop as your Image editor, it is the one that will give you the most flexibility. It will retain all layers, adjustment and effect that you have applied to you image. This format support high bit depths filed Up to 32bits. The file size for high bit depth file can be extremely large so think carefully before you consider saving your image in anything other than 8bits.

TIFF: Tagged-Image file format (TIFF, TIF) has most of the same attributes as the PSD format (when opened in Photoshop). This format is useful for its compatibility with almost all software that will open Image data. It also allowed the use of several different compression methods to reduce the size of your file.

JPEG: Joint Photographic Expert Group (JPEG, JPG) format is mostly used for images that will be displayed on screen or the web. This file format uses "Lossy" compression, which is to say, that data is lost during the compression process resulting in a much smaller file, but may also compromise image quality. If your camera only takes JPEG Images i recommend that you save the image as PSD during the editing process, as repeatedly opening and saving JPEG image causes recompression of the image and can severally degrade the data, resulting in very noticeable compression artefacts.

GIF: Graphics Interchange Format (GIF) is used to display indexed colour mode graphics. This file format may only contain 256 colours so it is not commonly used for photographs. However, it has several characteristics that have made it very popular in web graphics.

. *Small file size*
. *Supports transparency*
. *Supports animation*

PNG: Like the GIF format this format is commonly used for web graphics. The PNG format is a lot more flexible in its support for 24bit photographic images and alternative colour modes than the GIF format. However, it is not widely supported in web browsers.

PDF: The Portable Document Format (PDF) is very useful for displaying file across multiple platform and applications. It has the benefit of supporting compression, 16bit format and common colour modes. Whilst retaining font, vector, raster information and Photoshop editing (if selected).

Creating a New Documents

You can create a new document by selecting New form the File menu or by using the keyboard shortcuts Ctrl-N/Cmd-N (PC/Mac). The new dialogue box allows you set all of the parameters for your new document and have a number of presets to get you started, fig CA.15 and fig CA.16. It is important to set up your document correctly. For example, you need it to be at the correct size and resolution for it's intended purpose. If you select a paper size preset, it will automatically set the resolution for print. Likewise, if a web preset is selected the resolution will be set accordingly.

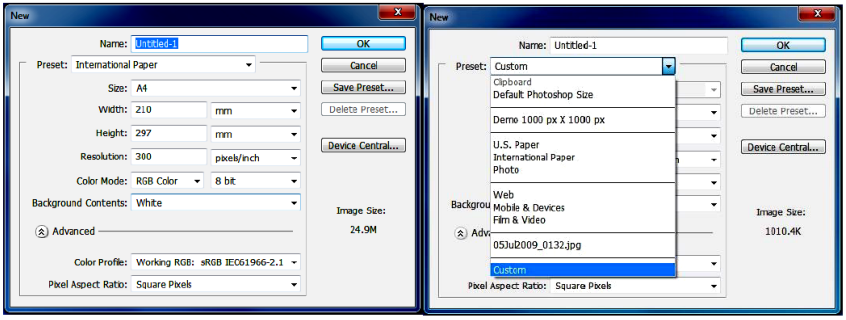


Figure CA.15: the New dialog box and Preset options

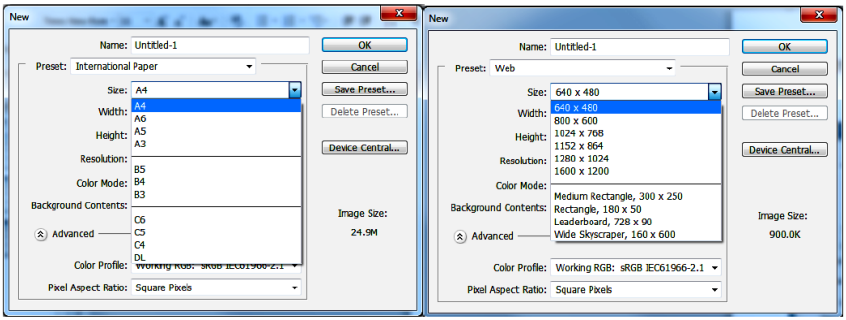


Figure CA.16: International Paper and Web size options

There are other options in the New dialogue box that also need to be determined:

- Colour Mode- For example, RGB for web design and photography and CMYK for commercial print.
- Bit Depths - 8 bit is generally fine unless you are intending to perform some major image editing or have a lot of gradients in a design, in which case it would be better to work in 16 bit mode and then convert to 8 bit once the editing process is completed. Web graphics should be 8 bit.
- Background Contents - This determines the colour of your background layer.

Under the Advanced section of the dialogue box there are two options:

- Colour Profile - you should leave the colour profile set to sRGB except for those wishing to do advanced Photographic work, in which case you may wish to use Adobe RGB or ProPhoto RBG.
- Pixel Aspect Ratio - in almost all cases this should be set to Square Pixels. Rectangular pixels are only generally used to correctly display wide screen content.

When you click Ok, you will have a new empty documents with a single layer.

Navigation and Zooming

Navigator Panel

When you are zoomed in to and image in Photoshop it is easy to lose where you are. B*y quickly* checking the Navigator panel, the red (grey) square (fig CA.17) will indicate the zoomed area. Navigation can also be undertaken by dragging the red (grey) square to a new position.

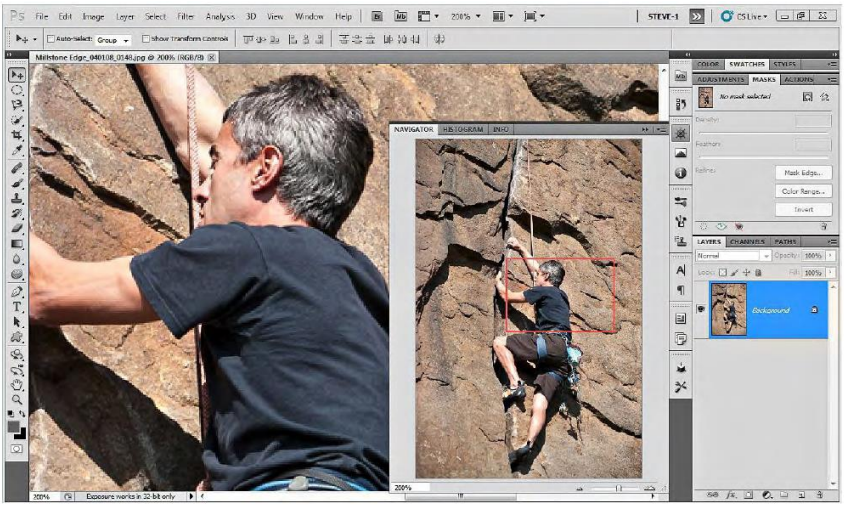


Figure CA.17: Navigator Panel

Hand Tool 

The Hand tool is used for moving the image around zoomed in. In CS5, flick panning is possible and can be enabled or disabled in the general preferences dialogue box which is found in the Edit menu on the PC, or the Photoshop menu on the Mac. If you are performing some close up editing and need to quickly access the Hand tool, holding down the Space Bar will temporarily select the tool, your original too will be reinstated once the Space Bar is released.

Zoom Tool 

When the zoom tool is selected, the default setting is to enable you to zoom in to an image in two ways:

- *Click on the desired portion of the image to zoom in predefined increments.*
- *Click and drag to define a particular area that you wish to zoom in on.*

When you wish to zoom back out, you can either click on the icon with the magnifier glass containing the minus symbol (fig CA.18), or Alt/option (PC/Mac) - Click to temporarily change the tool. The second method is preferable because unless you remember to click on the zoom in icon (the one with the plus), you may zoom out when you want to zoom in the next time that you come to use the tool.

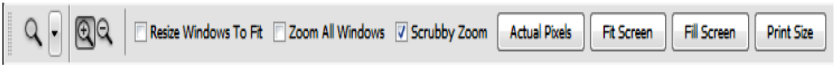


Figure CA.18: the Options associated with the zoom tool

If you have the Scrubby zoom option selected (fig CA.18) you can quickly zoom in to your image by clicking and dragging to the right. You can then zoom back out by clicking and dragging to the left.

There are four buttons in the Zoom options that can help you navigate your image:

- *Actual Pixels-shows your image at 100% magnification*
- *Fit Screen – zoom's your image to fit available work area, without respecting the panels and tool bar*
- *Fill screen -zoom's your image to fill all the available work area without respect for the position of any open panels.*
- *Print size- show the image at the size it will be printed based on the documents resolution setting.*

The above four options are also available from the View menu.

Useful Keyboard Shortcuts

Experienced Photoshop users will often use keyboard shortcuts for navigating and zooming. I recommend that you try to learn the following shortcuts, as it will speed up your workflow and productivity. It will also, ultimately make using Photoshop a more pleasant and rewarding experience.

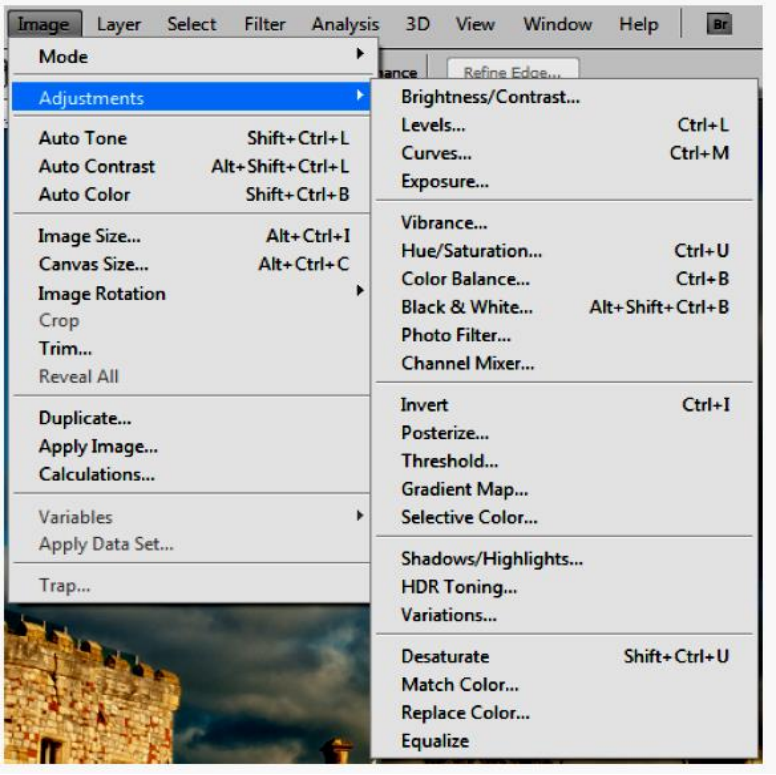
Keyboard/Mouse Shortcut	Action
Ctrl + Alt/Option (PC/Mac) + 0	100% Magnification
Ctrl + 0	Fit Screen
Ctrl + Spacebar	Temporarily select the Zoom tool (Zoom in)
Ctrl + Alt/Option (PC/Mac) + Spacebar	Temporarily select the Zoom tool (Zoom out)
Spacebar	Temporarily select the Hand tool
Tab	Hide the Tool bar and Panels
Shift + Tab	Hide the Panels but keep the Tools
Page Up or Page Down	Scroll Up or Scroll Down the area contained within the screen
Shift + Page Up or Page Down	Scroll Up or Scroll Down in 10 units
Double Click on the Zoom tool icon	100% Magnification
Double Click on the Hand tool icon	Fit Screen

Simple Global Adjustment

In this section we are going to take a look at Global Adjustments. This means adjusting the whole image to correct colour, contrast or exposure

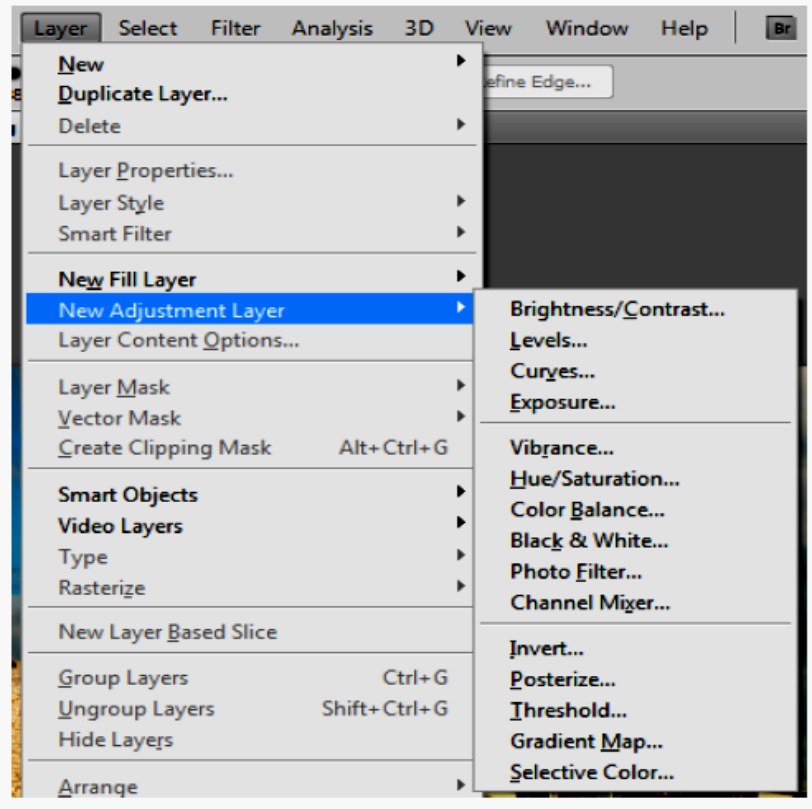
of an image. It can also be used as a special effect to create a specific look.

There are two different ways in which you can apply an adjustment to an image. The first is to go to the menu and choose adjustments:

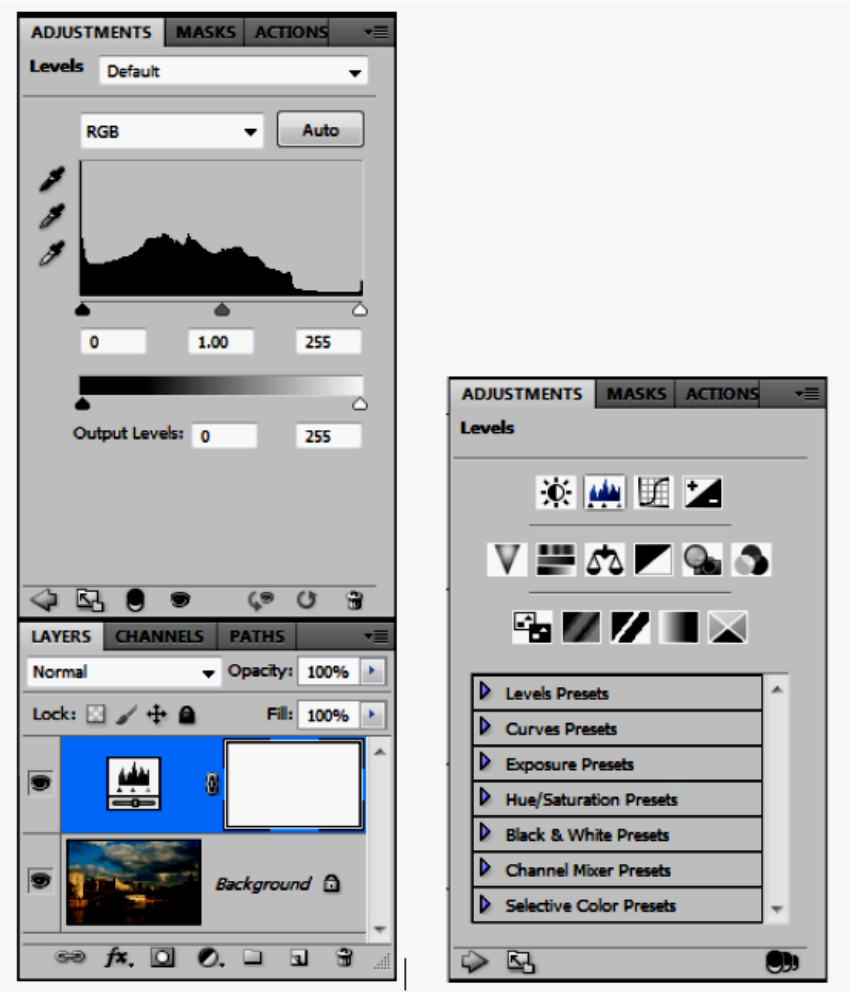


There are 22 different adjustment that you can apply to you image from this menu. However, if you apply an adjustment to your image and save it, the adjustment is permanent and destructive. The second method for applying an adjustment is non-destructive and utilizes a special type of layer called an Adjustment layer.

You can apply an adjustment layer by clicking on the  icon at the bottom of the layers panel (if you can't see your layers panel Press the F7 Key to open). Alternatively, you can create an adjustment layer from the menu by choosing New Adjustment layer. You can also apply an adjustment layer by clicking on the appropriate icon in the appointment list. The Adjustment list is visible in adjustment panel if no adjustment layer is highlighted in the Layers panel.



You will notice that this time there are only 15 adjustments instead of the 22 we saw previously. However, with the exception of Shadows /Highlights, we have all of the important adjustment available to us.



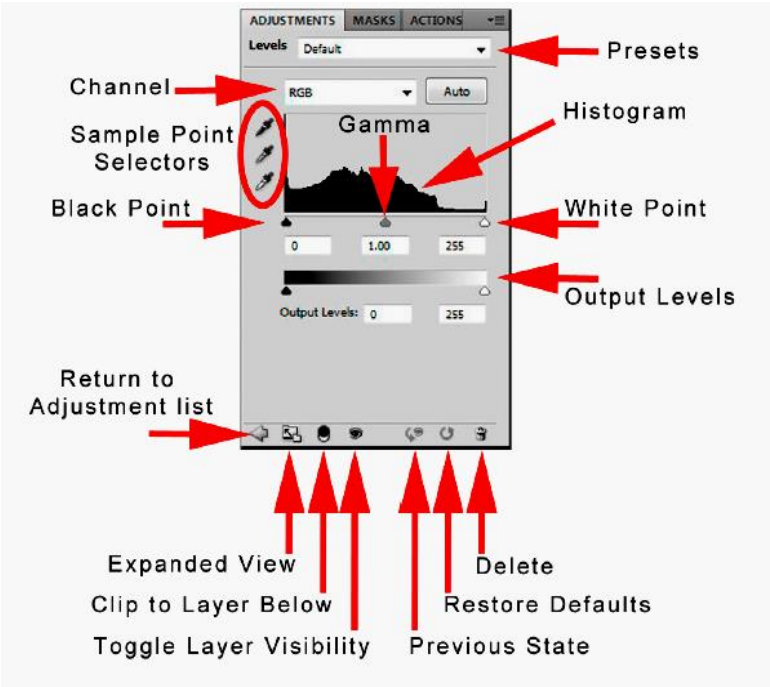
Once you've added the adjustment layer it will be visible as a new layer above your image in the Layers panel. The dialog box for the adjustment will also appear in the Adjustments panel (if you can't see the adjustments panel, select it from the Windows menu).

There are many different adjustments, but the most useful adjustment for beginners to start with are:

- *Levels*
- *Hue/Saturation*

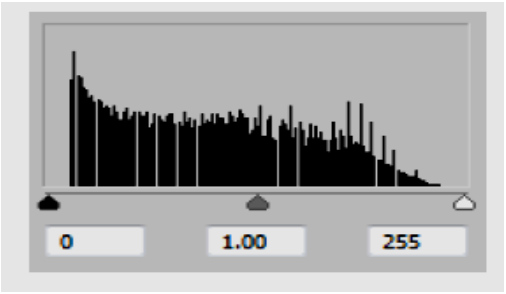
Levels

Levels is an extremely powerful tool for adjusting exposure problems and can also be used to colour an image.

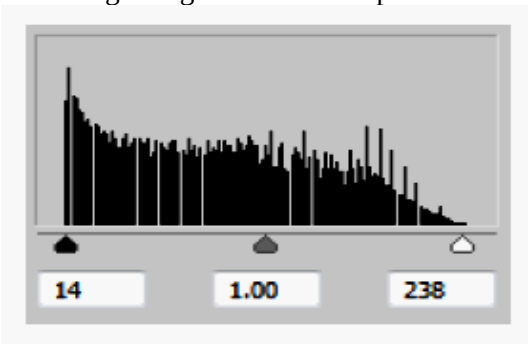


When you apply a Levels correction you will notice that there is a histogram, this is a representation of all the tonal values in the image. The black point slide on the left is set at 0 which is pure black and the value point slider on the right is set at 255 which is pure white. This gives us a complete range of 256 possible values.

The middle Gamma slider works differently to the black and white sliders. If the Gamma is adjusted to the left (towards black end of the scale) the image will get lighter in the mid tones and the opposite will happen if you move it to the right, the image will get darker in the mid tones.



In this Histogram we can see that the dark values don't go all the way to the black point on the left and the white values don't go all the way to the white point on the right. This means that there are no true blacks or whites in the image and this may result in the image lacking in contrast. We can adjust the histogram by moving the black and white points inwards to meet the beginning and end of the pixel values.

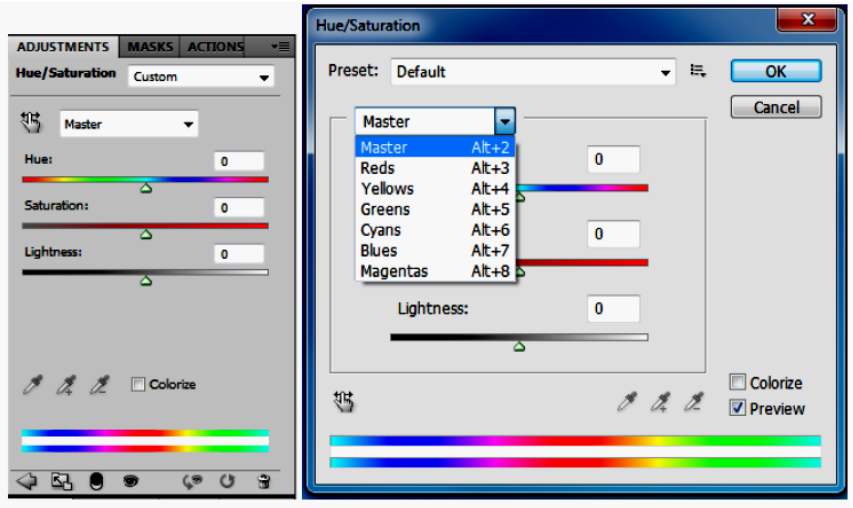


The black and white values have been changed and in the left hand box the input value now reads 14.This means that any pixels value of 14, or less, Will be mapped to 0 (black). Similarly, the right hand input box now reads 238. This means that any pixels value in the image of 238 or greater will be mapped to 255 (white). By moving these sliders inwards to meet where the pixel values begin and end, we now have an image that contains a full range of value from pure black to pure white. This results in an increase in contrast within the image.

If you are confident as to where your black points and white point should be in your image, you can select the appropriate sample point selectors (commonly known as the eye dropper tools, see figure below) in the Levels dialogue box and click directly in your image to set the value. The eyedroppers will also attempt to create neutral values, so they will remove any colour cast that your image may have.

Hue Saturation

The Hue/Saturation adjustment allows us to manipulate in an image, either globally or independently.



As with the Levels adjustment, i recommend that you apply it as an Adjustment layer. In the Hue/Saturation dialogue box we have ***three*** main sliders:

- **Hue:** affects the colour value. When you move it, the spectrum of colour is shifted. This is not useful as a global adjustment but can be very useful when applied to a specific colour.
  Saturation: represent the intensity of the colour. Moving the slider value to the right will make the colours in your image appear stronger. Moving the slider to the left will decrease the intensity of the colours and if you move it all the way to the left, will result in a greyscale image. For the best and most natural looking results you should do this on a per -colour basis.

- **Lightness:** as the name implies, will affect the lightness of image and unless used with care, can result in loss of contrast.

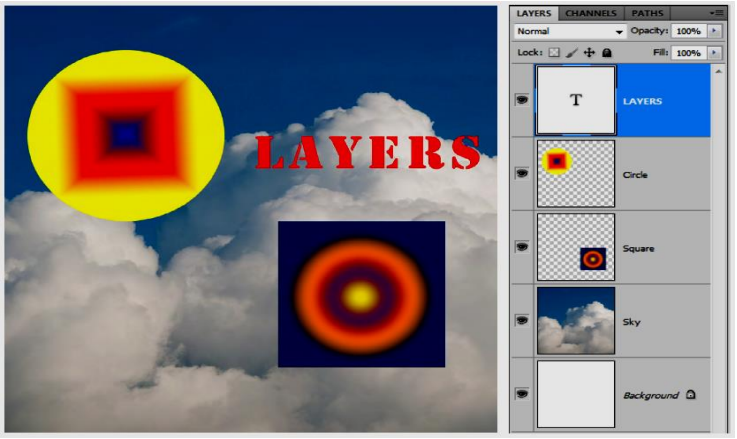
The default settings is to affect all of the colours in the image, as indicated by "Master" in the dropped down menu. However, individual colours can be affected independently, by either selecting the appropriate colours from the dropped down menu - see preceding fig (right). Or by using the small icon that depicts a finger on right pointing arrow, to clicking and dragging to the left will de-saturate the selected colour.

Near the bottom of the panel is a check box called Colorize, preceding fig if you click in this box your entire image will be tinted with one specific colour. The Hue slider will allow you to change the colour and the Saturation slider will adjust the intensity.

Layers

Beginners to Photoshop often have a problem with the concept of layers and how they can be made to interact with one another. They can be partially hidden using layer masks, they can be blended together in several different ways, they can be re-ordered, renamed, linked, grouped, clipped, duplicated and flattened.

So what are layers? Layers are simply stacks of images, objects or text. Layers are very powerful because they will allow us to edit and move individual image components without affecting other elements within the scene. Let's look at an example of a layered document, see the fig below.

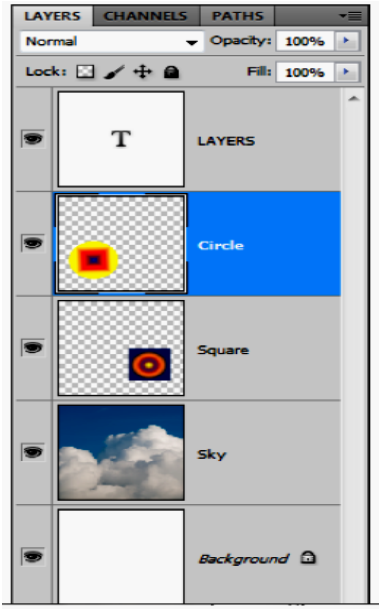


In the image of the Layers panel, in the preceding fig, we can see that there are total of 5 layers, from the bottom upwards, there is a Background layer, an image of the Sky, a layer containing a Square, a layer containing a Circle and a Text layer which contains the word "LAYERS", we can see 4 of the 5 layers, the background layer is completely covered up by the sky layer.
The two shape layers and the Text layer do not completely cover the Sky layer because they are surrounded by transparency which allows the underlying layers to show through. Transparency is represented in the Layer panel as a chequer-board pattern and is apparent in this example, in the square and circle layers. Finally, the Text layer is a special kind of Layer and although text is always surrounded by transparency, it does not show in the Layers panel.

Aligning and Moving Layers

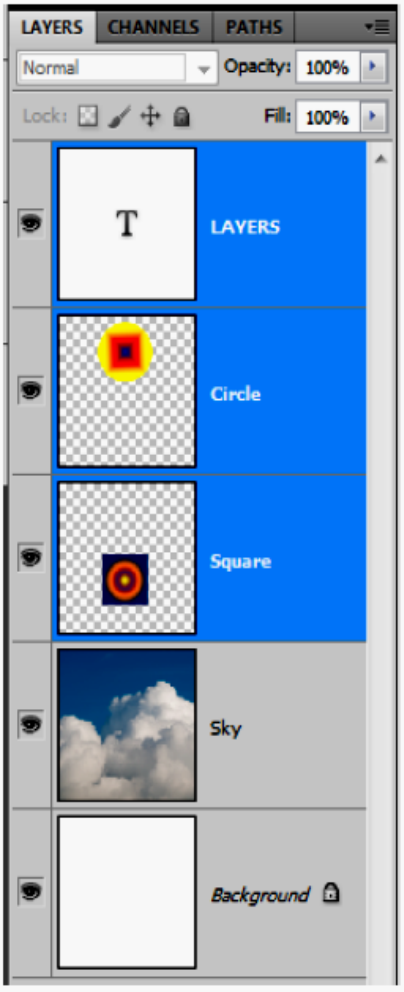
To move the contents of a layer, select the Move tool which is located at the top of the Tools panel, see the Fig below. You must then highlight the layer you wish to move by clicking on it in the Layers panel. Once the layer is selected it will appear blue, fig.5.3. With a layer selected, you can use the move to click and drag the layer into a new position.

Move tool

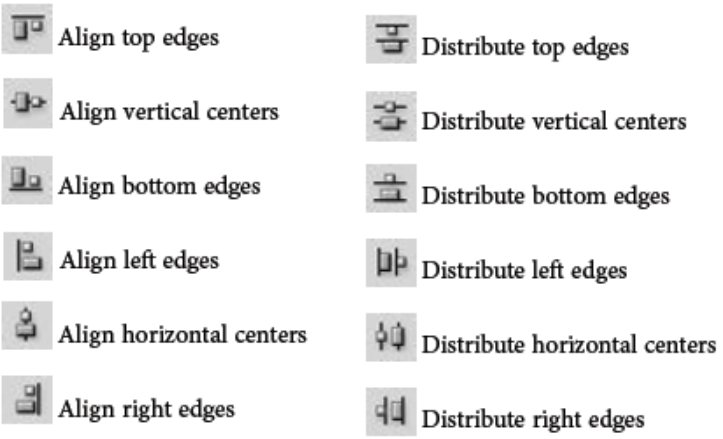


Circle layer is selected, as indicated by the blue highlight

Multiple layers can be selected by choosing a layer and then holding down the Shift Key and clicking on different layer. All of the layers between the two selected layers will be highlighted in blue, see the fig below. Non-adjacent layer can be selected by Ctrl/Cmd-click(Pc/Mac) on the required layers.

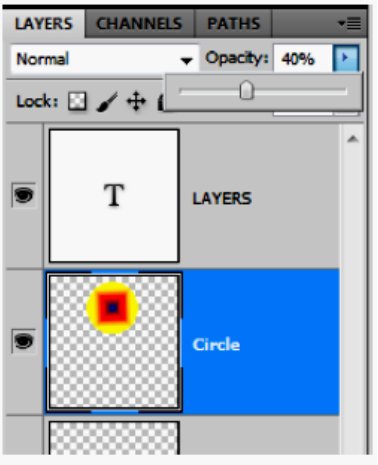


With several layers highlighted, multiple layers can be moved simultaneously, however, they will maintain their relative positions to one another. If you wish to change the position of the layers so that they are aligned with one another, you can use the Alignment tools that are present in the Options bar once the Move tool is selected, see the fig below.

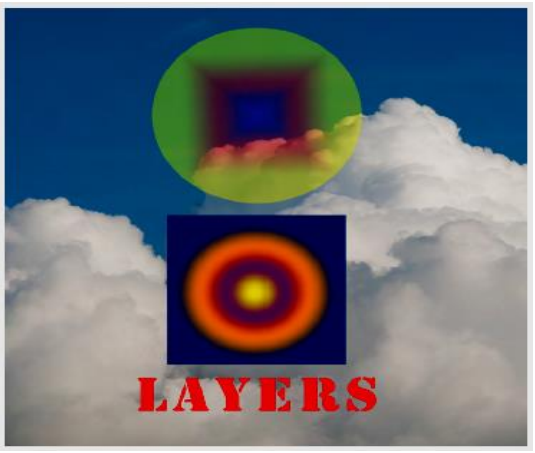


Layers Interactions

The manner in which the individual layers in a document may interact with each other is editable in a number of ways. The most basic interaction is Opacity, by default the Opacity is 100%, but this can be adjusted by moving the slider that appears when you click on the right pointing arrow at the side of the Opacity value, see the fig below. Alternatively, you can just position your cursor over the word "Opacity" and drag to the left to reduce the value, and to the right to increase it.



When we reduce the opacity of a layer it allows any layers that may be below the altered layered, to be partially visible, see the fig below.



Blend Modes

Layer can also interact with one another by using Blend modes. They allow one layer to affect underlying layers in a number of different ways. However, to help you to understand in broad terms what their function is, similar modes are grouped together, see the fig below.

You can alter a layers blend mode by clicking on the down arrow next to the word Normal in the Layers panel.

Naming Layers

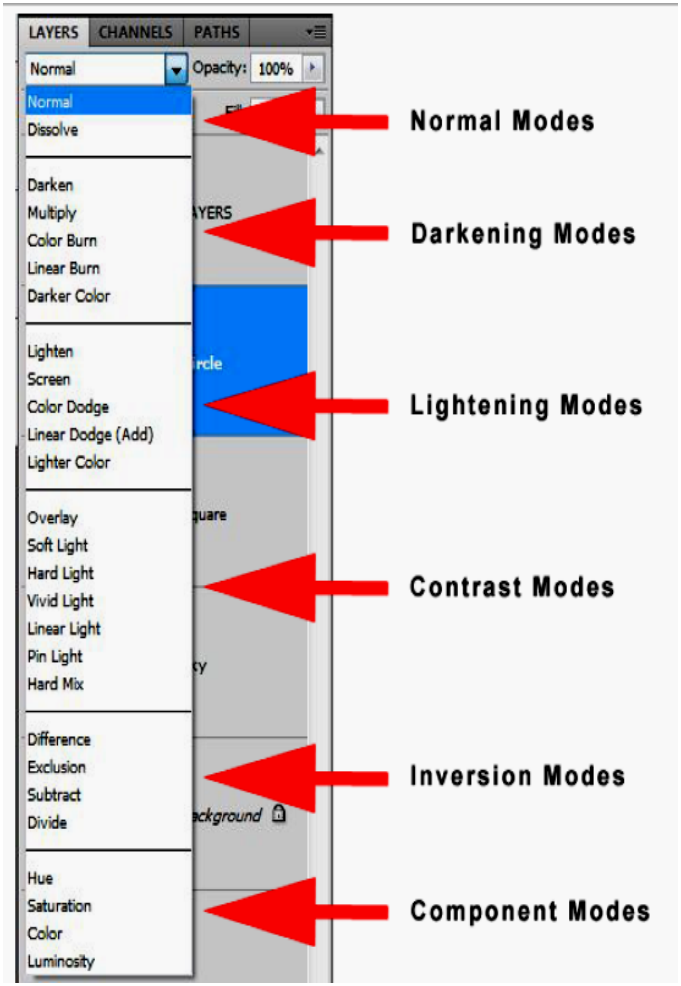
It can be very easy to lose track of which layer is which if you have many layered document. It is a good idea to get into the habit of naming your layers as you create them. To rename a layer simply double click on the name of the layer in the layer's panel and type in the new name.

Text Layers

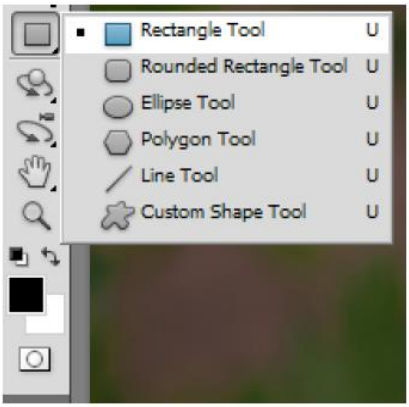
Adding a text to an image is very simple and can be done in two ways.

- Select the Type tool from the Tool panel and click in the image where you would like the type to appear. A flashing cursor will appear and you can commence typing.
- If with to create a column of text for a magazine layout, choose the Type tool, but instead of just clicking, click and drag to create the text box. Now when you type, it will be contained within the box.

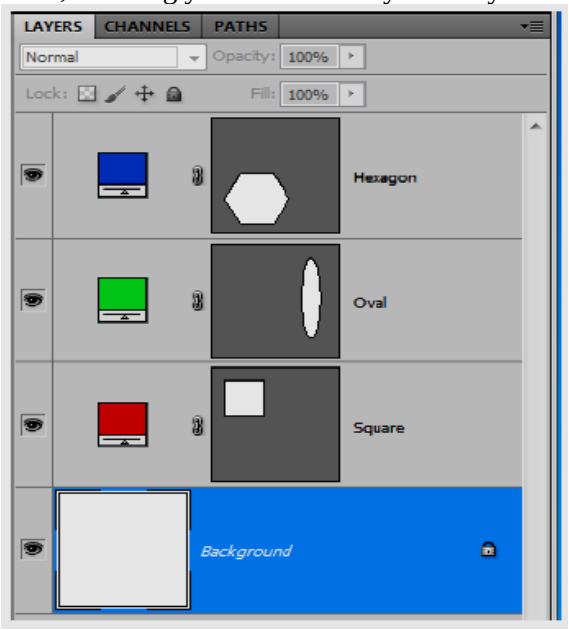
You will notice that when you type a new type layer appears in the Layers panel so there is no need to manually create a new layer when you wish to add text.

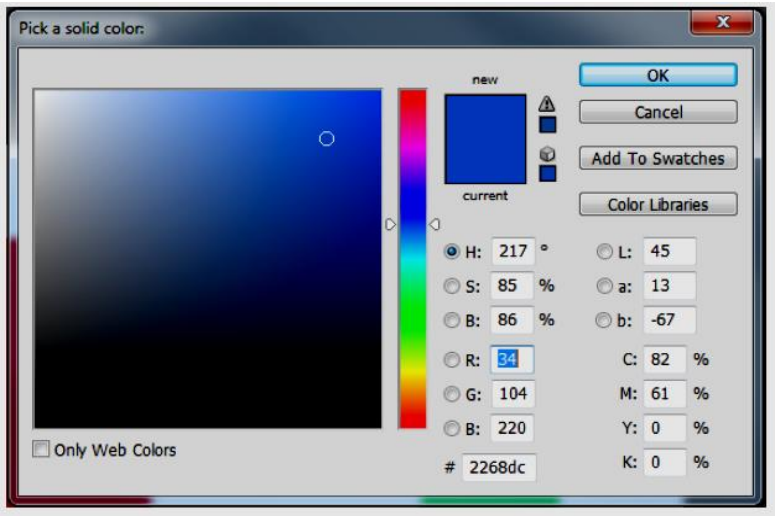


When you choose one of the Shape tools, see the fig below, the default setting in the associated Options bar is to create what is known as a Shape layer, in the same manner as the Type tool, Shapes layers are created automatically when you start to draw with one of the tools, provided that the create shape layer icon is selected in the Option bar- the first selected icon from the left in the second fig below.



Shape layers consist of a solid colour layer with a vector mask, see the fig below. A vector mask is like a stencil, only allowing the colour to be seen where the shape has been drawn. The colour of a shape can be changed at a time by double clicking on the coloured thumbnail icon in the layers panel. The icon will reflect the currently chosen colour. As you double click the Colour picker dialogue box will appear, see the second fig below, allowing you to choose any colour you wish.





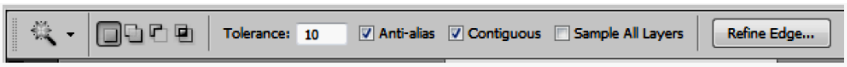
Simple Selections

Selections allow us to isolate a particular portion of the layer. We can then edit the selected region without affecting the rest of the layer. There are so many ways in which we can generate a selection in Photoshop. In this Appendix we will start by taking a look at some of the simple selection tools.

Magic Wand Tool

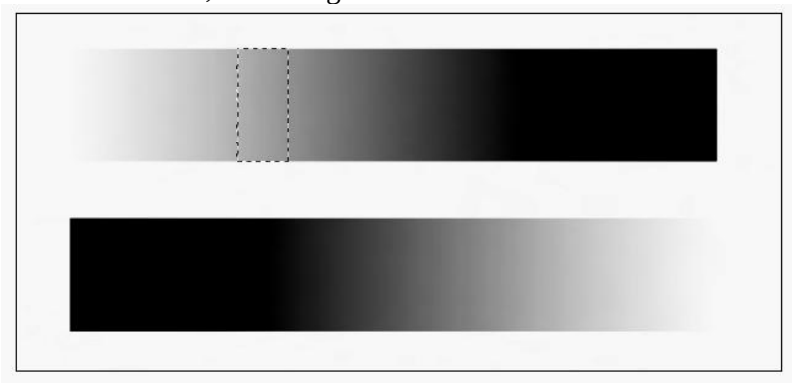
The Magic Wand tool is located fourth from the top in the Tools panels and is stacked with the Quick Selection tool. To access a stacked tool, simply click and hold on a particular tool to see the other tools available in that location. Stacked tools are indicated by the small black triangle in the bottom right hand corner of the icon.

The Options bar functions associated with the Magic Wand tool play an important role in its functionality, see the fig below.



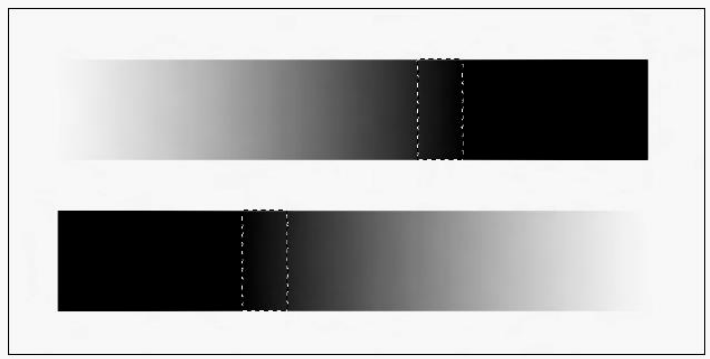
- Tolerance: determines how much of the image is selected when you click on a particular part of your image. A value of 10 indicates that 5 darker values and 5 lighter values will be chosen based on the position where you clicked, see the fig below.
- Anti-alias: make selection edges smoother.

- Contiguous: when this option is selected only neighbouring pixels are selected, see the fig below.

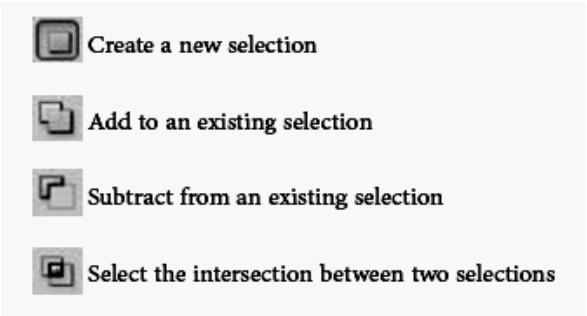


When unchecked all pixels of the selected value will be selected, see the fig below.

- Sample All Layers: if chosen, will take all layers into consideration and not just the highlighted layer.

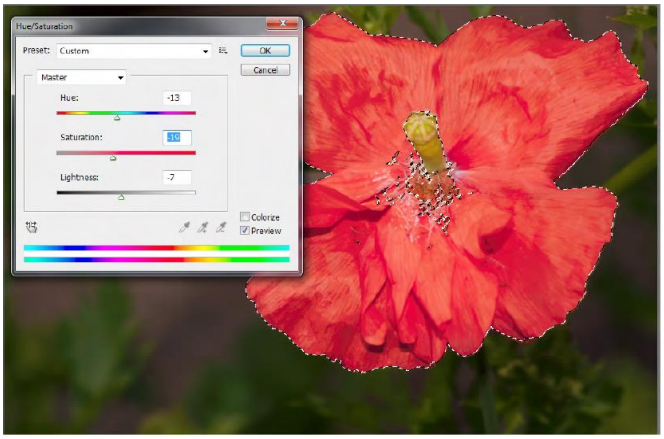


There are other icons in the Magic Wands Options that allow us to make more accurate selections.



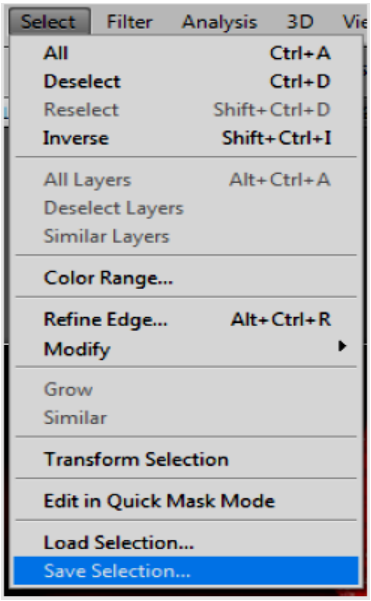
In the following example we will look at making a selective edit to an image. In the fig below, the orange poppy was selected with the magic wand tool so that we can change the colour of the flower without affecting the rest of the image. The poppy contains a surprisingly amount of different tones so it is difficult to make an accurate selection. If you keep increasing the tolerance value, you will get to a point where areas outside of the poppy start to become selected. We don't want this, so a better solution for an image like this, is to keep the tolerance value relatively low (in this example, I used a value of 20) and chose the Add Selection icon in the option bar. You can then keep clicking various parts of the flower until you are happy with the selection. If you manage to have unwanted selection areas appear, click on the Subtract from selection icon to remove those values from the selection (***note***: take care not to leave this icon selected once you have removed the unwanted selection areas).

Once the selection is complete you can go to the Image menu>Adjustments>Hue/Saturation. The values that I chose to use can be seen below.



Not only can you apply Adjustments to selected areas, you can also apply filters from the Filter menu. In the next fig, you can see that the Filters>Artistic>Plastic Wrap filter has been applied to the same selection

Selections can be saved and loaded from the Select menu, fig below. If you have taken the time to make a selection, it is a good idea to save it in case you want to make further edits. Once you have finished with your selection you can remove it by choosing Deselect from the Select menu.

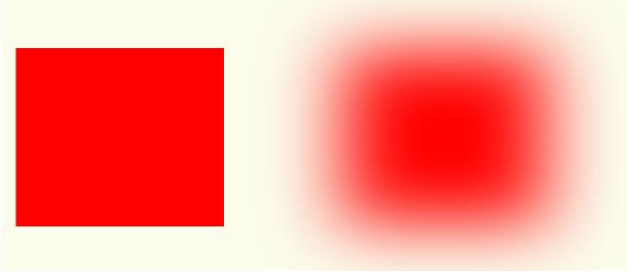


Marquee Tools

The Marquee tools allow us to draw basic shapes with the selections, see fig below.

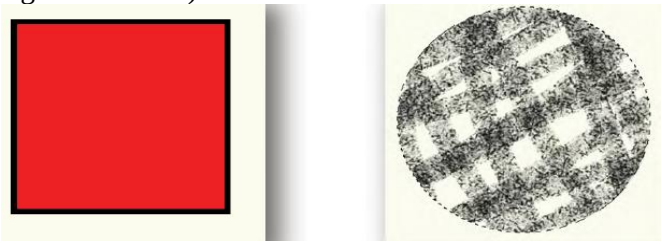
As with the Magic Wand tool, the selection can then be used to edit the image.

The option bar for the Marquee tools is very similar in appearance to that of the Magic Wand tool. However, you will notice that you also have an option called Feather. By default, the Marquee tools will define a hard edged selection. However, by entering a pixel value in to the Feather option we can create a selection that will have a soft edge. Soft feathered selections can be very useful as they allow us to blend edits with underlying layers, see fig below.

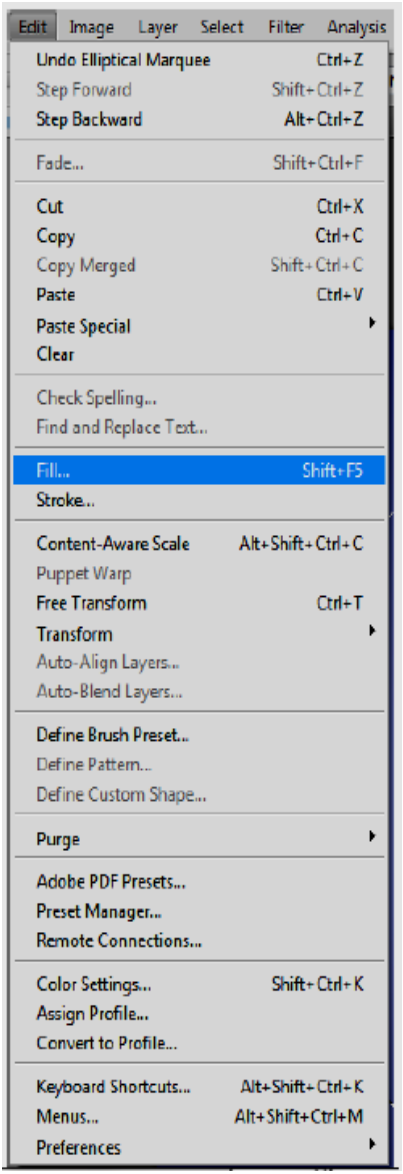


As well as you could able to edit the content of a selection on a layer that contains an image or design, we can also create a new empty layer and use the active selection to:

- Fill with a colour (see fig below)
- Stroke with an outline (see fig below)
- Paint or clone inside (see fig below)
- Apply an adjustment or filter to the selected area (as was saw with magic wand tool.)



The fill and stroke commands can be found in the edit menu, see fig below.



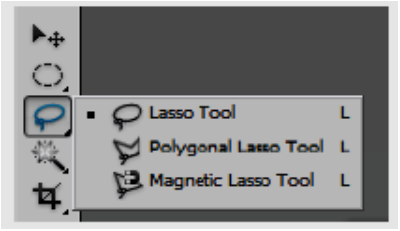
Marquee Selection Modifier keys

When drawing a selection using the marquee tools there are a number of keys on the keyboard that will alter how the selection is drawn.

- Shift key – constraints the Rectangular and elliptical marquee tools to a perfect square or circle respectively.
- Alt key – draws a selection from the centre instead of the edge.
- Space bar – allows the selection to be moved during the drawing process.
- Shift key (after the selection is drawn) – temporarily chooses the add to Section option.
- Alt key (after the selection is drawn) – temporarily chooses the Subtract from Selection option.
- Shift and Alt - temporarily chooses the Intersect with Section option.

Lasso Tools

Another set selection tools are the Lasso tools, see fig below. These tools are used to create more of a free-hand drawn selection.



The Lasso tools comprise of:

- The Lasso tool – a completely hand drawn selection.

- The Polygonal Lasso tool – draws straight lines between clicked points.
- The Magnetic Lasso tool – used for drawing selections around image contents by detecting contrast changes around the edge of the subject.

Copying a Selected item to a New Layer

The act of copying a selected item to a new layer is a crucial editing technique in Photoshop. It allows you to work non-destructively, by applying edits to a new layer instead of the original pixels that are located in the Background layer. It also allows you to duplicate items to create composite images (see fig below) or complex designs.

There are a number of ways to copy an item to a new layer. Once you've made your selection, go to Layer menu and choose New>Layer via Copy. Alternatively, you can use one of the most useful keyboard shortcuts in Photoshop to do the same task – Ctrl J. when you make the new layer it will appear as if nothing has changed, this is because the duplicate item will be in perfect register with the original image below. Select the Move tool (the first tool in the tool bar) and reposition the item as required. If you are using an image as in the preceding fig above, a large feathered selection will help the new layer blend more effectively with the one below for a more seamless result.

It is often the case that you will wish the copied item to be of a different size or orientation to the original, see fig below. This can be achieved by going to the Edit menu and choosing one of the Transform commands. In the fig below the item has been scaled down slightly and flipped horizontally, so that it appears to be a little further away and

swimming in the opposite direction. Once you've made your transform edits, you must click on the tick in the option bar in order to accept the changes (if you don't like what you've done, click on the X to cancel the edits).

Choosing Colours

Foreground and Background colours

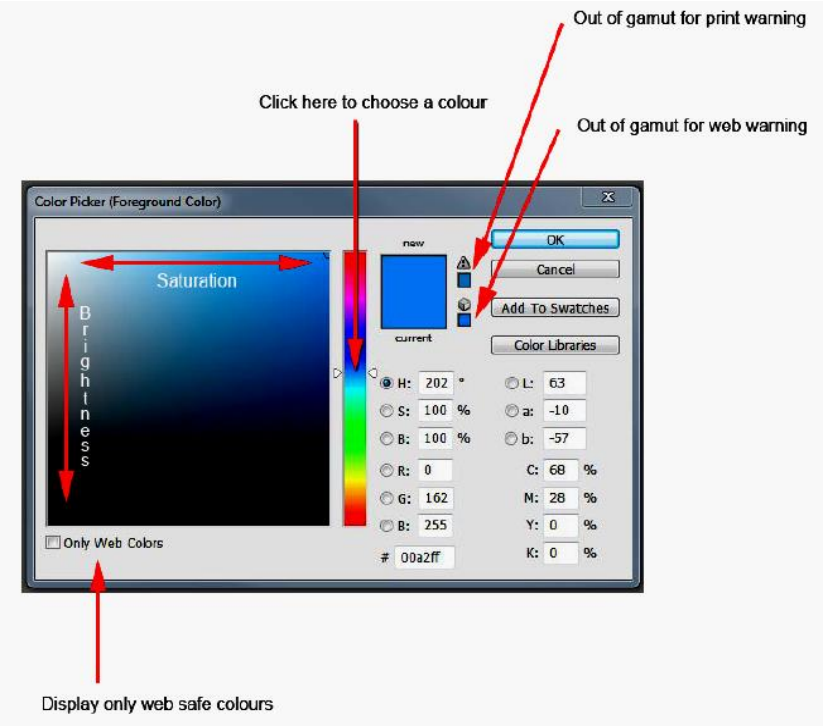
At the bottom of the tools panel there are two colour chips that represent the colours that are currently selected. There are commonly known as the foreground and background colours. The foreground colour is the colour currently use by a specific tool and is represented as the top colour chip in the tools panel. In fig below we can see that the foreground colour is set to blue and the background colour is set to red. The reason for having a background colour is that it can be switched for the foreground colour very quickly by clicking on the double headed arrow icon. The background colour can also be used in gradients – in this case drawing a gradient with the gradient tool will result in a blue to red gradient being drawn (using the default gradient settings)

The black and white icon in the preceding fig will return the colours to their default setting of black as the foreground and white as the background.

Changing the Colours

You can change the selected colours by simply clicking on either the foreground or background colour chips at the bottom of the Tools panel.

This will result in the Colour Picker dialogue box appearing, see fig below.



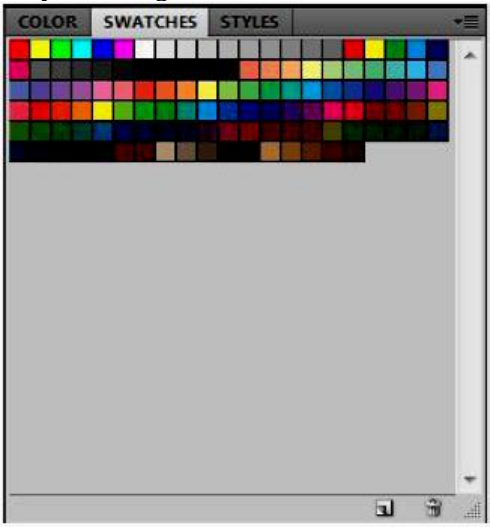
Click in the multi-coloured area to choose your colour. The brightness of the colour can be adjusted by clicking higher or lower in the large area on the left dialogue box, which represents variations of the chosen colours with respect to brightness and saturation of the colour is increased by clicking further to the right of this area.

If your colour picker dialogue box does not look like the one the fig above, there could be reasons for this. Firstly, the Operating System colour picker has selected in the general preferences, found in the edit menu. If this is the case, you can change it to Adobe version if you wish. The colours represented in the Adobe Color Picker may also appear different. The “Only Web Colors” box may be ticked or an alternative colour mode chosen. To have the dialogue box appear as in the preceding fig above, make sure that the only web colors box is not checked and that the HSB (Hue, Saturation, Brightness) color option is selected.

There are two small warning boxes that may appear to the right of the new/current box. The top one indicates an out of gamut for printing warning, which means that it's outside the range of colors that the printer can reproduce correctly. The lower one is warning that the colour may not be displayed correctly if viewed in a web browser. If you click on either one of warning icons, it will alter chosen colour to make it print and /or web safe.

Swatches Panel

The Swatches panel contains number of useful pre-set colours, see fig below. If your Swatches panel is not visible in your current workspace you can access it by choosing swatches in the Window menu.

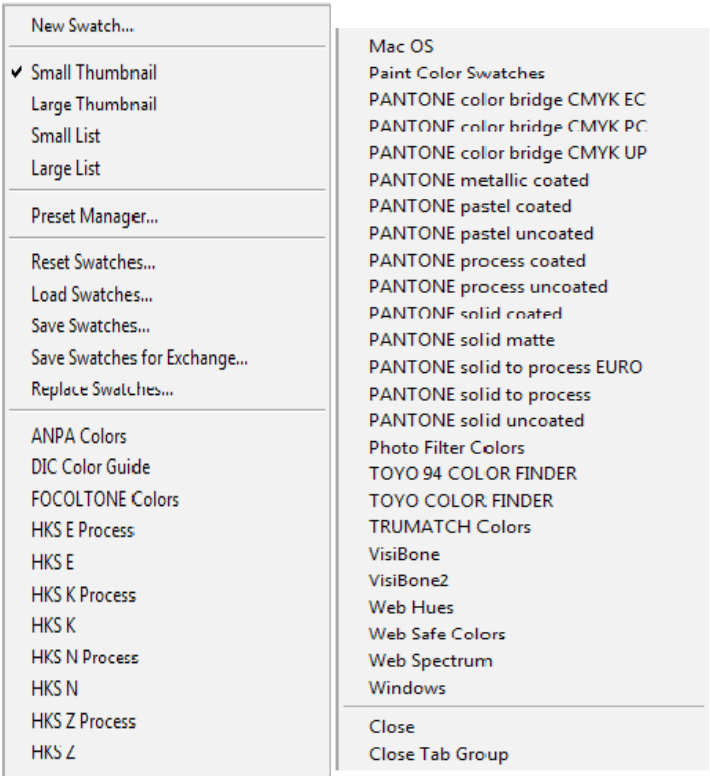


Additional swatches are available from the fly-out menu in the top right hand corner of the panel, see the next fig. You can also create your own swatches by:

- *Click on the turning page icon at the bottom of the Swatches panel to add the current foreground colour to the list.*
- *Click on the "Add to Swatches" button in the Color Picker dialogue box.*
- *Click on an empty space in the Swatches panel to add the current foreground colour to the list.*

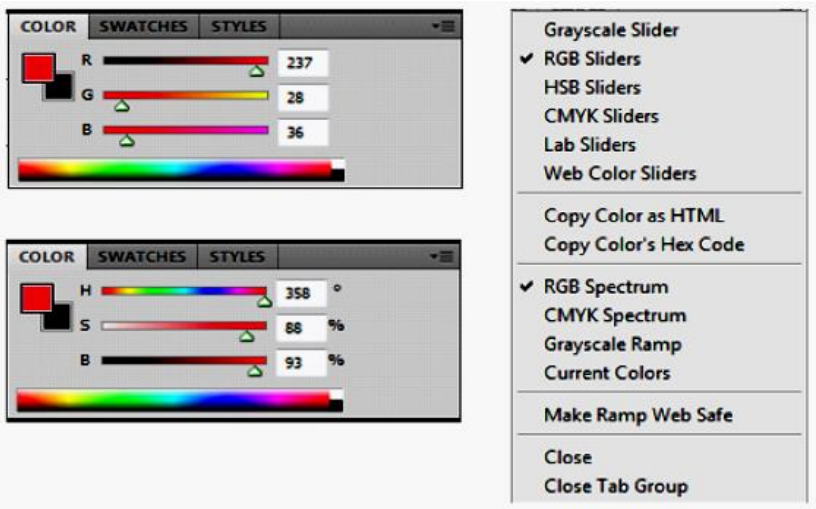
Swatches are extremely useful for graphics and web designers because you can create your own set of custom colours from a layout or web page. You can then save that set of colours from your swatches panel fly-out menu, see the fig below.

They will be quickly loaded (from the fly-out menu) the next time that you are required to create a design for a particular client and need the colour palette that was used in a previous design.

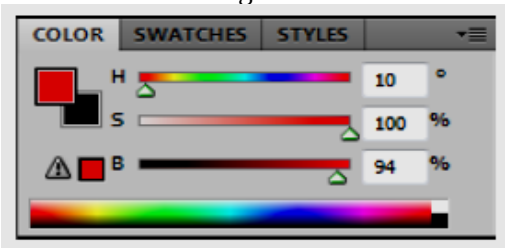


Color Panel

You can create a foreground or background colour from the colour panel, see the fig below.

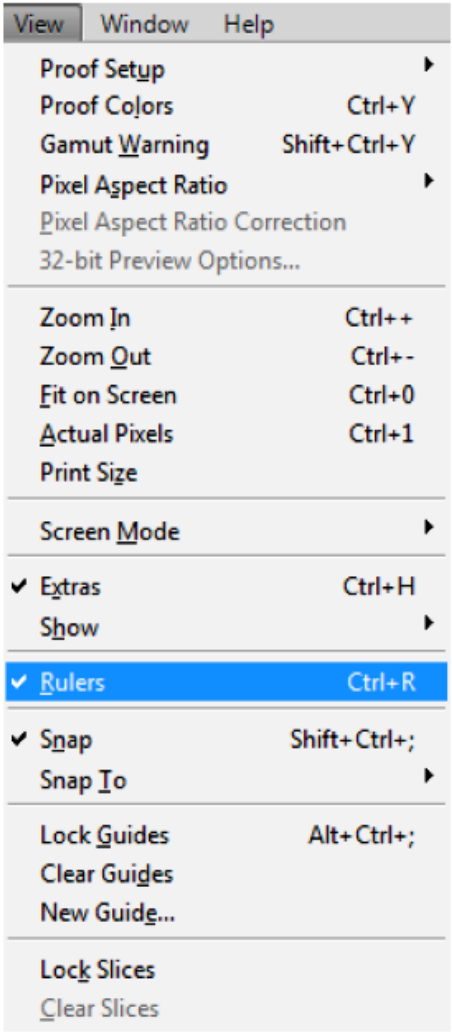


The colours are changed by either moving the slider or by clicking directly on the spectrum. Both the slider and the spectrum are configurable to accommodate working in the most popular colour spaces. The most useful setup for this panel, is to have the slider set to HSB and the colour spectrum set to RGB, if you can create a colour that is out of the gamut for your printer, a warning icon will appear, see the fig below, if you click on the icon, the colour will automatically be change to the colour in the nearest gamut colour.



Guides and Rulers

Designers of all types will find using guide and rulers a great help in laying out pages for the web or for print. The options for turning on and off ruler and guide functions can be found in the View menu.

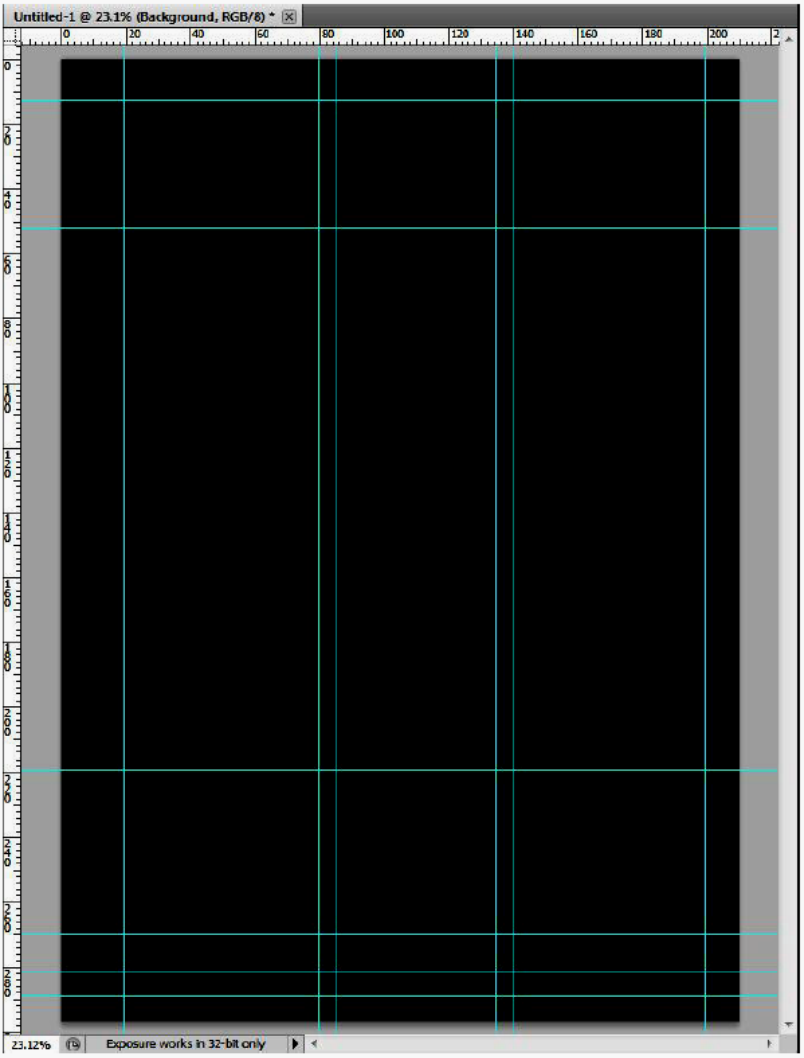


To have the Rulers appear around the outside of you work area, select Rulers from the View menu, or the keyboard shortcut (the keyboard shortcut for commonly used function are indicated to the right of the menu item), Cmd/Ctr+R(Mac/PC).

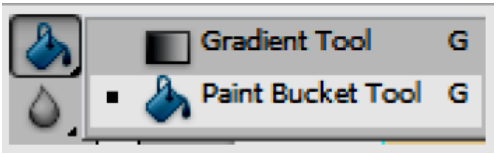
You can create guides by clicking on the New Guide option in the View menu. However, it's far quicker and easier to simply click and drag from inside one of the rulers to create a guide. Clicking and dragging from the vertical rulers will create a new vertical guide and performing the same procedure from the horizontal rulers will result in a horizontal guide.

Once your guides have been created, you can change or refine your layout by selecting the move tool and repositioning the guides. To do this you must select the move tool and place the cursor over the guide that you wish to move. When the cursor changes, you can then move the guide.

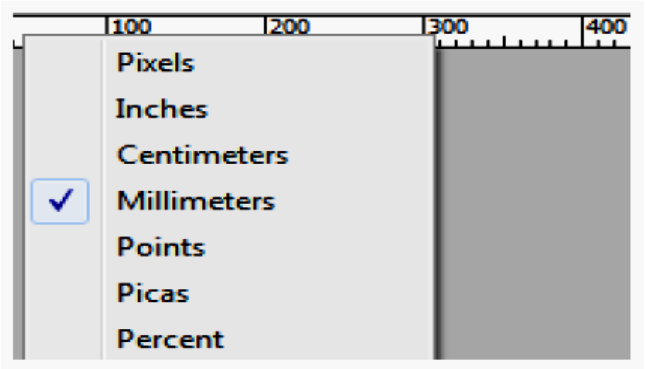
You will notice that the guides are cyan in colour by default. This can be changed if you wish by changing the colour setting in the Preferences. The Preferences are found in the Edit menu and the guide settings are found in the sub category "Guides, Grid, & Slices ". I find it easier to fill my background layer with a dark neutral colour to that i can see the guides more clearly when preparing a layout, see fig below.



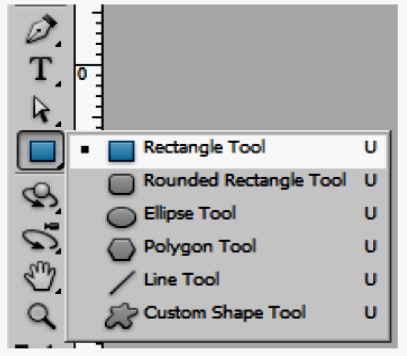
You can do this by selecting the required colour as you foreground colour and then use the paint packets tool, (ground with the gradient tool), to on the background to apply the colour. It can be returned to white (or whatever colour you like) by repeating the process once your guides are in place.



If you wish to change the units displayed on the Rulers, you changed the "Units and Rulers" setting in the Preferences. Alternatively, you can right click in the ruler and select you alternative units from the box that appears below.

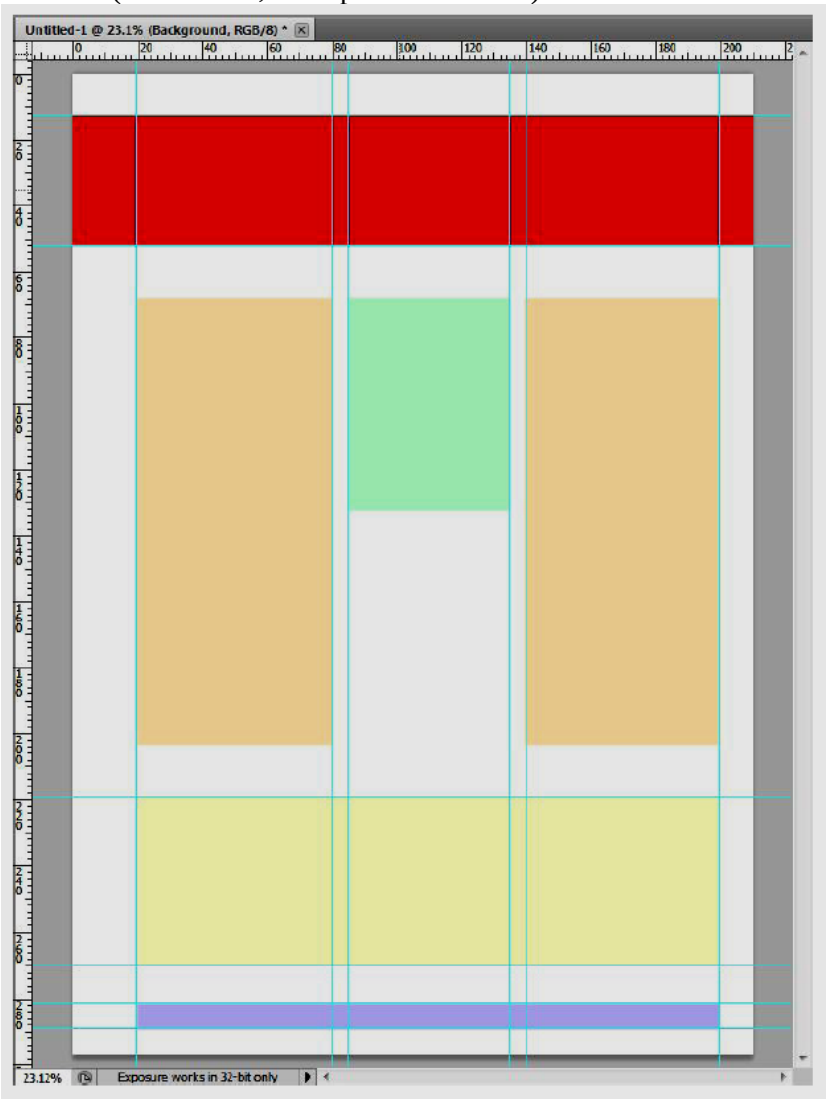


You can create a layout mock up by using the Vector Shape tools (fig below) to draw shapes to represent where specific items will appear in



your layout (fig below). Drawing with Vector Shapes has many similarities to using the Selection Marquee tools. The modifier keys for

adding and subtracting from shapes are the same as for creating selections (Shift to add, Alt/Option to subtract).



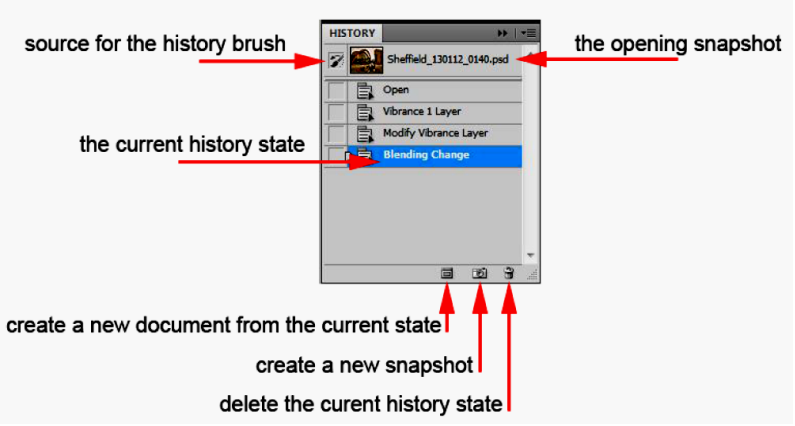
While drawing a layout, it is useful to utilize the snap setting in the View menu. This will cause the drawing tools to snap precisely to the guides as you are dragging out the shapes.

Once your design is complete you may wish to view without the guides. However, if you use the Clear Guides option from the View menu, the

guides will be gone for good. A better way to work is to hide the guides. This can be done from the Show options in the View menu or by simply pressing Cmd/Ctrl+;(Mac/PC) to toggle the guides on and off.
Your guides will be saved with your documents, but they will not appear in a print or an exported web slices.

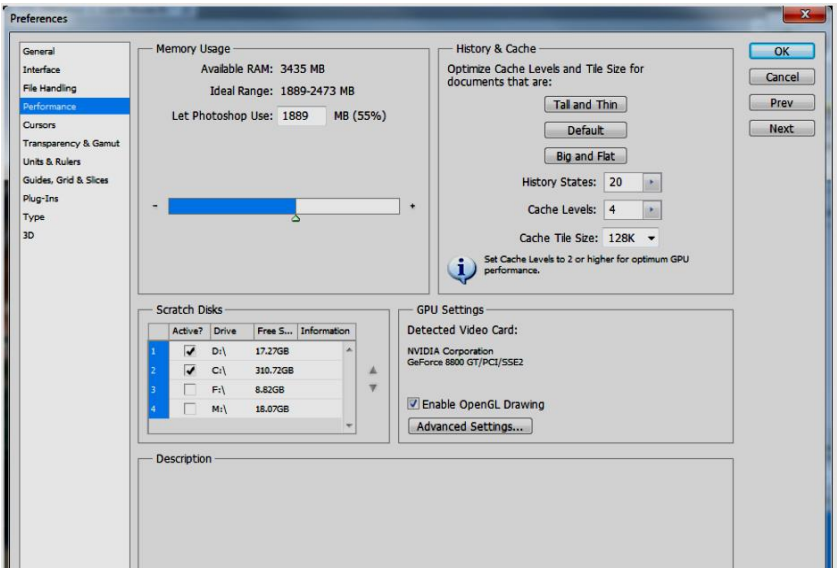
History

If you make a mistake in Photoshop you can undo that mistake by going to the edit menu and choosing either Undo or Step Backward. Clicking Undo will take you back a single step, the command will then change to Redo (followed by the name of the action that you have just performed). Selecting the Step Backward option take you once set backwards in the History panel, see the *fig below.*



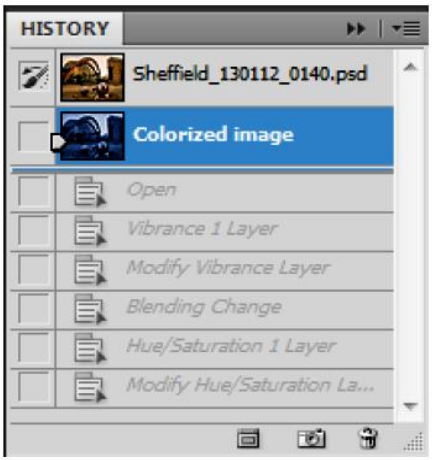
The History panel can be opened from the Window menu and contains a record of operations performed up to a finite number. The number of steps visible in the History panel are *set* in the performance section of the Preferences, see fig below. The Preferences are found in the Edit menu.
The default number of history states is 20, in the fig below. The minimum value is 1000. It is very tempting to set a high number of History states. However, this will have serious consequences with respect to performance of your computer as Photoshop has to remember the exact configuration of your documents in each of the History states. I recommend 20-30 History states for a beginner. When you become more experienced at using Photoshop try reducing the number of History states to 8-12. This will free up some of your computer resources so that it performs other functions faster. I also recommend

setting the History states to a low value if you are trying to run Photoshop on a minimum specifications machine.



Snapshots

When you create or open a document an initial Snapshot appears at the top of the History panel and is the default source for the History brush, *see the 1st fig under History*. If you make many edits to your document and make a mess of things, you can always return the document to its original state by clicking on this Snapshot.



If you are making a several edits to your document are about to run out of History states you can click on the camera shaped icon at the bottom of the History panel in order to create another Snapshot. The new Snapshot will remember all of the edits that you have created to that point, *see preceding fig above.* You can then proceed with your edits, knowing that if you make a mistake you can click on the new snapshot and don't have to start all the way from the beginning again. It's a good idea to rename any new Snapshots that you make by assigning them a descriptive name. To rename any Snapshots, double click on Snapshot name and type in the new name.

An alternative to working with Snapshots is to create an entirely new document from a History state. You can do this by clicking on the icon shown in fig below.

A new document from a History state will have that state as its opening Snapshot and the History for that document will be empty.

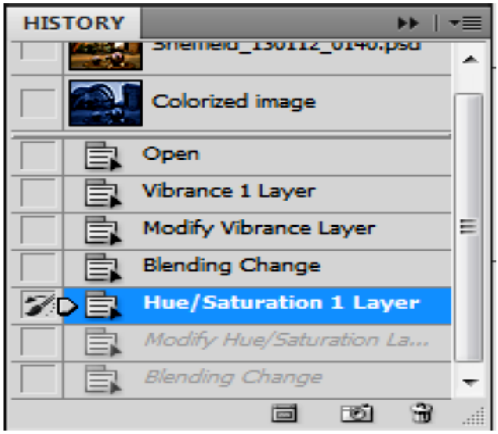
Keyboard shortcuts:

Action	Keyboard Shortcut (Mac and PC)
Undo	Cmd/Ctrl + Z (Mac/PC)
Redo	Cmd/Ctrl + Z (Mac/PC)
Step Backward (in History states)	Cmd + Option/Ctrl + Alt + Z (Mac/PC)
Step Forward (in History states)	Cmd + Shift/Ctrl + Shift + Z (Mac/PC)

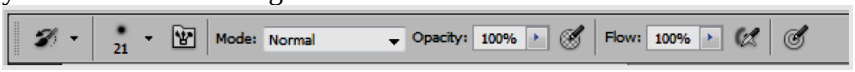
History Brush & Fill History

You can apply any History state to specific area in two different ways. Firstly you can select History Brush Tool from the tools panel.

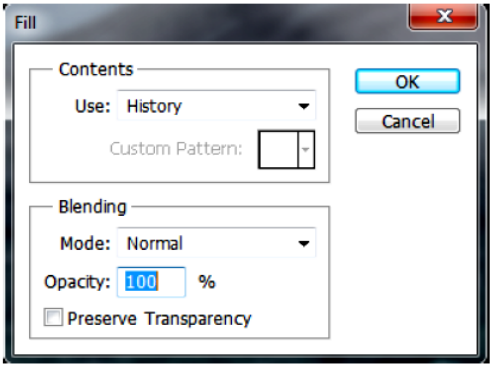
You then select the History State that you wish to paint with by clicking in the left hand box corresponding to that history states position in the History panel see the fig below.



Adjust the brushes properties from the Options panel (see the fig below) and point on the document. It is best to create a new blank layer stack to point the History state on. This way you will be working in a good non-destructive workflow and you can always discard the layer if you don't like the changes.



Another way of applying a history state to a specific area is to make a selection using any of the methods described in the section, '***Simple Selections'***. Click in the left hand box at the appropriate History state in the History panel. Create a new empty layer at the top of the Layer stack, go to the Edit menu and choose Fill. In the dialogue box that appears (see the fig below), use the drop down menu next to the word use: to select history then click OK.

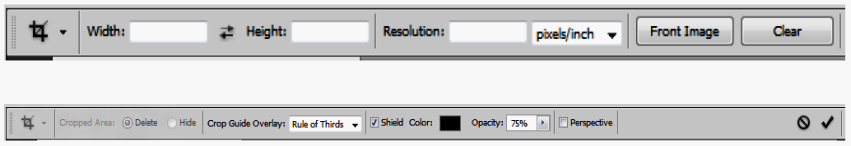


The selection will be filled with selected history, see the fig below.

Cropping

There are two main ways in which we can crop a document in Photoshop. The first and most commonly used method is to use the Crop tool, see fig below.

The options panel that is associated with the crop tool is unusual in that, before the crop area is drawn there are one set of options. After the crop area has been selected a second set of options will appear, (see the fig below).



The first set of Crop tool options are to enter specific dimensions and resolution for the document. This is very useful if you need the document output to specific criteria, to fit a certain print size or if the image has to be placed inside a design or web page layout. The second set of options pertain to the shield that is drawn with the Crop tool and defines the crop area. You can change the colour and opacity of the

shield and you can also use the small drop down menu to determine which type of grid overlay or visible.

Basic Printing

Printing is one of the areas of Photoshop that causes people problems. Typically, their prints don't look like the on-screen version of the document. You have to accept that your monitor and printer reproduce colours in every different ways and that they will never be completely identical in their output.

The very best method of getting a good print output that closely resemble the version you can see on your monitor, is to buy (or borrow) a calibration device. There are many calibration devices on the market. However, the best ones enable you to calibrate both your monitor and printer so that their out matches as closely as possible. This kind of device will create a custom "Profiles ", which Photoshop can use to display and print your documents accurately.

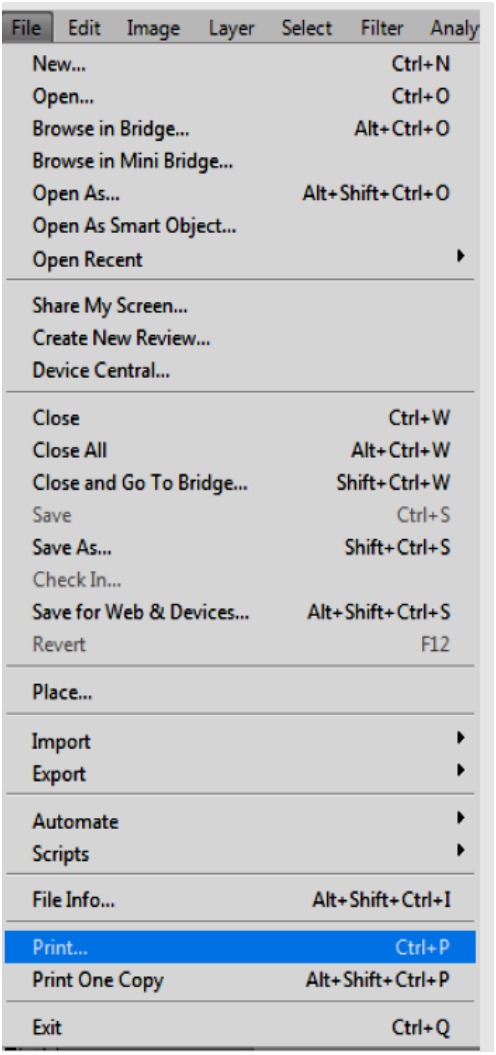
A good calibration device can be expensive, so let's look at how there can be better results from Photoshop if we don't have access to one.

- *Go to your printer manufacturer website. Download and install the latest drivers for your printer.*

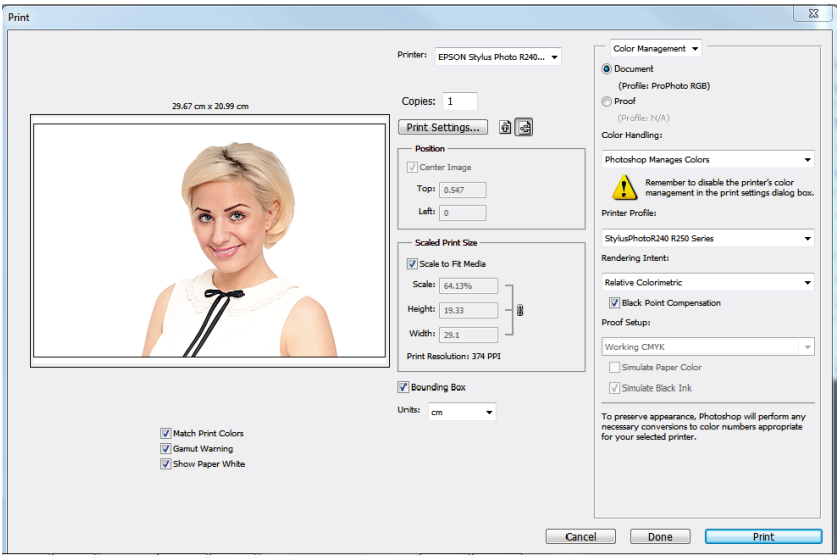
- *Make sure that you are viewing your monitor in a colour neutral environment. Colours that are around us strongly influence how we see colours on the screen. If you have strong colours in the room that you are working in, try working in the dark.*
- *Check your documents for out of gamut colours -colours that your printer is unable to reproduce, such as strong, heavily saturated colours.*
- *If you are working with image, make sure that you adjust your image so that there is good detail in both the highlight and shadow areas.*
- *Let Photoshop handle the printing. Turn off your printers colour management options.*
- *If you are making high quality photo or fine art prints on high quality paper, check the manufacture's website to see if they have produced a custom profile for the particular paper stock that you are using.*

Photoshop Print Dialogue Box

When you wish to make a print go to the File menu and choose the Print option.

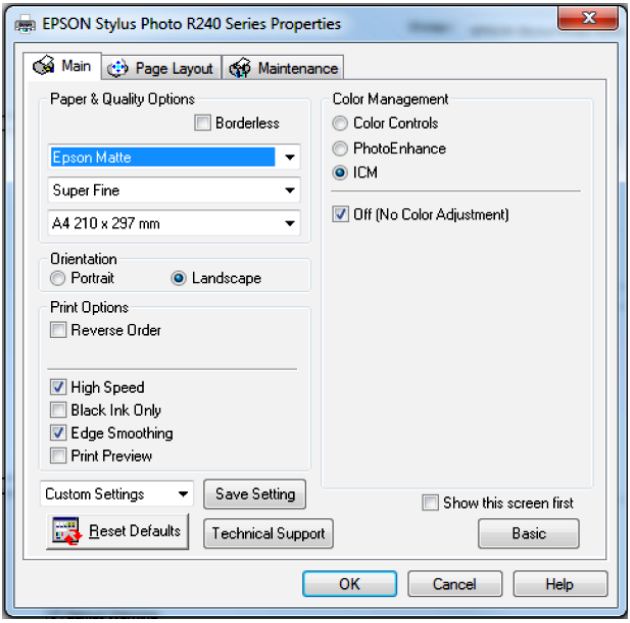


Once the Print option is chosen, the Print Dialogue box will be displayed, as shown below.



Make sure that the following settings are made:

- Select your printer from the drop down list at the top of the dialogue box.
- Set Photoshop Manages Colors from the Color Handling drop down list.
- In the Printer Profile section, have this set to either your printer or any custom that you may wish to use.
- For photographics document, have the Rendering intent drop down menu set to either Perceptual or Relative Colorimetric and make sure that the Black Point Compensation Check box is ticked.
- For documents containing graphics or charts, you may use of the Rendering Intents, but you may be able to reproduce stronger colours if you use the saturation options.
- Click on Print Settings to open up your printer's settings (this will be different for every printer - refer to your user manual for how to make the required settings). Select the resolution, paper size and paper type that you wish to use. Find the colour Management Settings for your printer and turn them off, see fig below.



If you want to produce a high quality print, it is often a good idea to use the "Scaled Print Size" options to make a smaller test print. Enter a percentage value in the scale box (make sure that the Scale to Fit Media check box is unchecked or you will not able to do this). This will save you ink in the long run, as you will probably have to make some adjustments to your document and make a few print before you are happy with the results. Prints are almost always darker than their own screen version, so it's good idea to make a level adjustment layer and brighten the image slightly before you print. Once you have make a print that you are happy with, you can turn off or discard the Levels Adjustment Layer.

Flash Animation–*Practical Session*

Introduction

Flash is widely used in the creative industry to develop engaging projects integrating video, sound, graphics, and animation. You can create original content in Flash or import assets from other Adobe applications such as Photoshop or Illustrator, quickly design animation and multimedia, and use ActionScript 3.0 to integrate sophisticated interactivity.

Use Flash Professional to generate graphics and animation assets, to build innovative and immersive websites, to create standalone applications for the desktop, or to create apps to distribute to mobile devices running on the Android or iOS system. The 2014 release of Adobe Flash Professional CC is briefly discussed in this appendix.

Installing Flash

You must purchase the Adobe Flash Professional application as part of Adobe Creative Cloud. The following specifications are the minimum required system configurations.

Windows

- Intel® Pentium 4, Intel Centrino®, Intel Xeon®, or Intel Core™ Duo (or compatible) processor
- Microsoft® Windows® 7 (64 bit), Windows 8 (64 bit), or Windows 8.1 (64 bit)
- 4 GB of RAM
- 1024x900 display (1280x1024 recommended)
- Java Runtime Environment 1.7 (included)
- QuickTime 7.7x software recommended
- 4 GB of available hard-disk space for installation; additional free space required during installation (cannot install on removable flash storage devices)
- Broadband Internet connection and registration are necessary for required software activation, validation of subscriptions, and access to online services.

Mac OS

- Multicore Intel® processor
- Mac OS X v10.9 64-bit, 10.8 64-bit, or 10.7 64-bit
- 4 GB of RAM

- 1024x900 display (1280x1024 recommended)
- Java™ Runtime Environment 1.7
- QuickTime 10.x software recommended
- 4 GB of available hard-disk space for installation; additional free space required during installation (cannot install on a volume that uses a case-sensitive file system or on removable flash storage devices)
- Broadband Internet connection and registration are necessary for required software activation, validation of subscriptions, and access to online services.

For updates on system requirements and complete instructions on installing the software, visit www.adobe.com/products/flash/tech-specs.html.

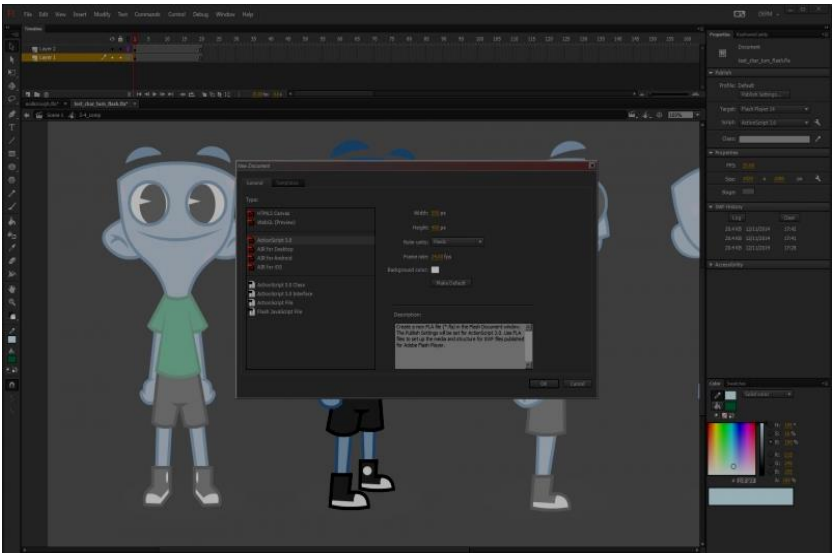
Install Flash from Adobe Creative Cloud at creative.adobe.com/ and make sure that you have your login and password accessible.

Animation basics

In this section we will go through the basics of the Flash interface and tools, the basic concepts of nested animation and ultimately a walk cycle from start to finish. We will be using Flash CC 2014 but the basic concepts such as keyframes, tweens, eases, symbols and instancing, transcend each version of Flash over the past six years or so.

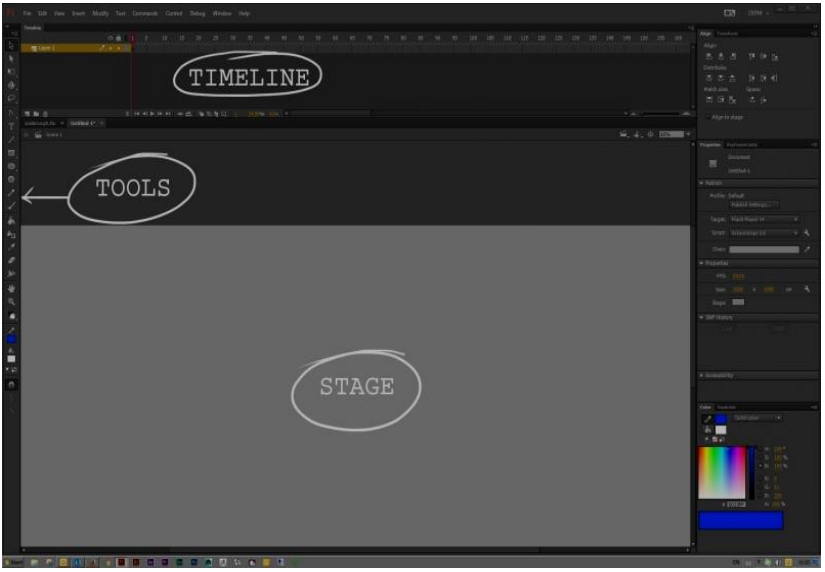
New Document

When opening a new document you will see the various options. Generally for animation we leave all values alone, except for the Width/Height, which is set to 1920x1080, the frame rate, which depends on the project, but for now it will be at 24, and the background colour. We prefer to work on a grey background but choose anything that suits you.



Interface

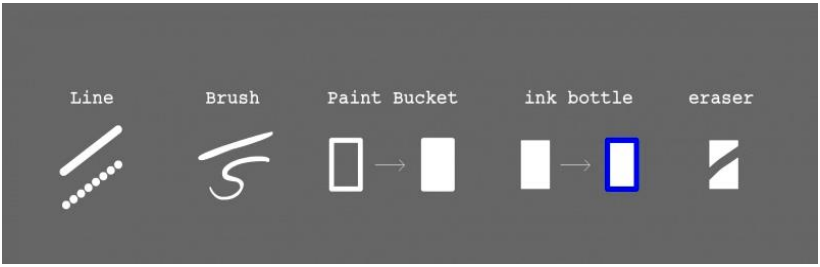
Once you open the new document, you will see a lot of panels. For now, we will concentrate on 3 of these; the ***timeline***, the ***stage*** and the ***tool*** panel. See the figure below in order to become familiar with them:



Tools

The main tools we'll be using are *the brush tool, the line tool, the paint bucket tool, ink bottle tool and the eraser*. Usually, when animating, we use the *brush* & *eraser* tools to sketch out rough ***poses*** and ***inbetweens***, then once happy we can use the line and bucket tools to create finished drawings. Details later!

For the purposes of discussing the basics, we will also discuss the 'Oval tool' to draw circles.



Strokes and fills

There are 2 types of graphical elements when drawing in flash; Strokes and fills. Think of them as the *outlines* and the *bits coloured in*. That may be a simplified description but once you get playing with lines and fills you will quickly see how they work. Try this; use the Oval tool to create a circle or ellipse. By default it will be an outline with the centre filled in. Now select the filled part and delete it. What remains is the stroke. Now use the bucket tool to fill it again. Now delete the outline. What remains is the fill. It's important to get the idea of this, so play around with it.

Stage

The stage is where we will draw everything we need for our animations. Characters, Backgrounds and effects are all created here. Anything that exists outside of the stage will not be visible when exporting your movie.

Timeline

The timeline is the panel from which we will be able to see the amount of frames we have in our animation, the positioning and distance between our main drawings (keyframes), the various layers we use for rough drawings, and various elements and the 'tweens' between keyframes. *Details later!*

Symbols - *Nested Timelines*

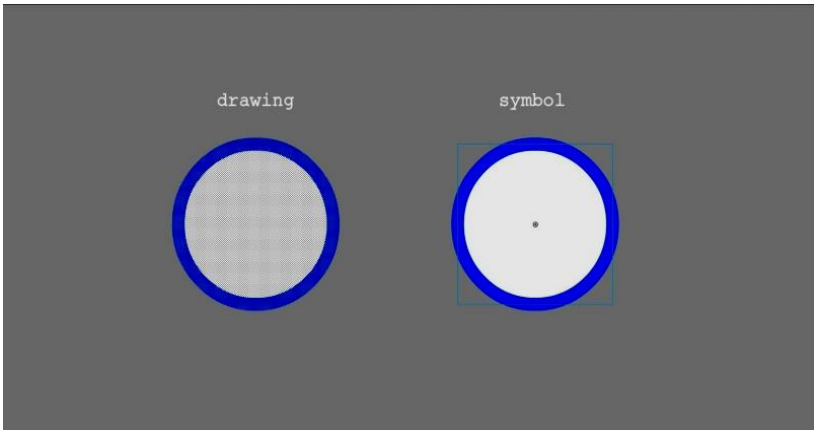
To fully explain symbols let's define what a 'drawing' is.

It sounds funny, but in Flash knowing the difference between a drawing and a symbol will save a *lot* of heartache down the line. A 'drawing', is raw graphics on the stage that have been created using any of the drawing tools, such as the line tool, oval, brush, rectangle etc. It can be erased with the eraser; it can be added to or manipulated and ultimately is the core of your animation.

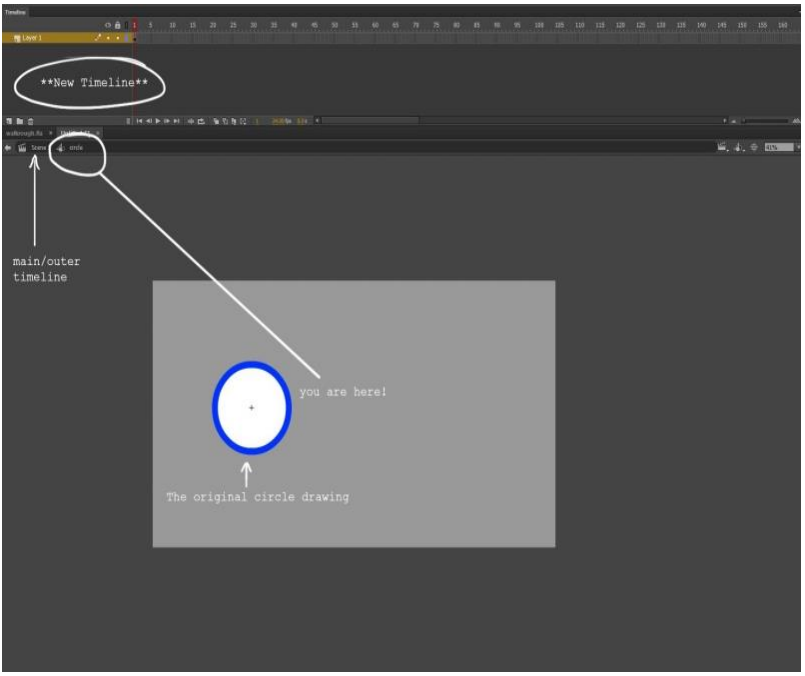
A *symbol* on the other hand, is created from a drawing as a *container* for the drawing. It can't be added to or manipulated in the same way as a drawing. Think of it as placing an apple in a box. Before placing it in the box, it's edible, squashable etc. Once placed in the box it's still there as an apple but you can no longer manipulate it in the same way. But now you can do things to the box that you could not with the apple, and yet the apple stays the same. Sounds strange?

Do the following:

- Create 2 circles on the stage.
- Select one of them (using the selection tool)
- Hit the F8 key
- You're about to turn that drawing into a symbol - give it a name & click ok.
- Congrats, you've created a symbol. Notice the blue box around the circle.



- Before we proceed, go to the timeline and double click the layer where it says *Layer 1*. Re-name its Outer Timeline.
- Now; double click your new symbol. Notice anything different? Your newly changed layer is back to *layer 1*, one of your circles is grayed out and the other is back to being a regular *drawing*. This is because you are now ***inside the symbol*** (or using my earlier analogy, you're inside the box with the apple).



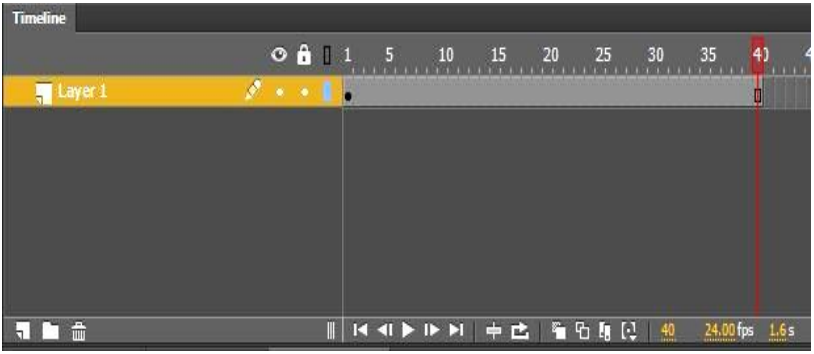
- Inside the symbol, you now have a brand new timeline, separate but inside the main timeline where you first created your circles. This is important stuff to understand because what this means is **that you can create animation inside animations.**
- Now manipulate the circle drawing somehow. Use the eraser tool or brush or line and change it in some way.
- Once you've done this. Double click anywhere away from your drawing on the stage. This will now bring you back to the *main timeline*. Notice the layer you had changed earlier and the changes you made to your circle inside the symbol remain.

Symbols - Tweening

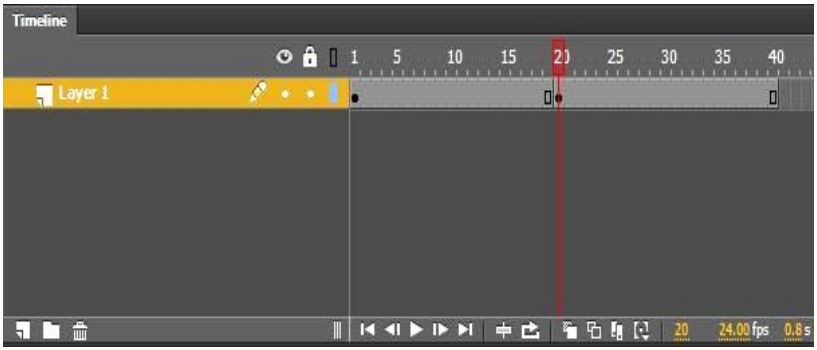
Now, let's do some animating.

Create a new document by going to File-New or press Ctrl + N. Choose your stage dimensions and colour and click ok. Create a circle as before using the oval tool. Now select the new circle and hit the **F8 key** to convert the drawing into a symbol.

In the timeline, go to frame 40 and hit the **F5 key**. What this does is add frames (not keyframes) to the timeline. This is how long your animation will be. Press enter and watch the slider traverse the new frames you have created.

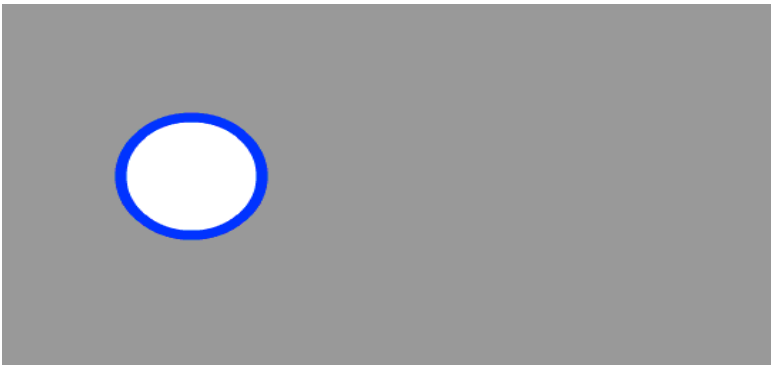


Now, for Keyframes. A Keyframe is essentially a selected frame on which the drawing or symbol changes. To illustrate this go to frame 20 and hit the **F6 key**. This will place a new keyframe on that place on your timeline.



While on frame 20, go to your circle symbol and move it anywhere on the stage. Now if you hit enter or scrub the timeline you will see that once the timeline reaches frame 20 it changes to the new position.

Now this one will be used a lot from now on; Right click in the area between the two keyframes in the timeline and select ***create classic tween***. The area should go blue. What this does is fill in the gaps in the positions of each keyframe. Hit enter or scrub the timeline. You should have something similar to the image below:



Now you can drag the second keyframe anywhere in your timeline to adjust its speed.

Let's close the case here for now!

WHAT's NEXT?

The End!!!

Contacting MH7 Global Publishers/ Independent Research Group

This book is a positive implication of a continuous preparation, research and speech delivery into Web Programming *Client-Side Scripting* and its consistent relevance. I welcome thoughts, helpful updates and reactions from readers, though I am not always able to respond to most letters on time. My personal e-mail address: habeebmamman@yahoo.fr, index.habeebmamman.com

For those people in organizations and into research seeking to implement the ideas presented here, development tools and services are available through my research group, MH7 Global Publishers/ Independent Research Group. Services includes: *organizing academic/seminar conferences, free computing research activities, design and implementation of third party apps, and training of trainers for organizations.* To contact MH7:

MH7 Global Publishers/ Independent Research Group Services
Physical Address: (*Request through email*)
Phone: (*Request through email*)
Email: habeebmamman@yahoo.fr, index.habeebmamman.com
Website: www.habeebmamman.com

About the MH7 Global Publishers/ Independent Research Group

Mamman A. Habeeb is the architect of MH7 Global Publishers/ Independent Research Group: The 7 Rules of Excellence (*emerged in 2010*)**.** The research interests of the group are: '*Novelty Detection and Game Playing Using Neural Networks*'.

The MH7 group in collaboration with **Oxford Research and Publications International** are the organizers of 22nd Conference. *Theme: Developing World and Intellectual, and Research for Achieving the new Sustainable Development: to be held at* ESEP-Le Berger Universite, Cotonou-Benin.

MH7 forthcoming academic journal is titled '***International Journal of Computational/Digital Philosophy* (IJCDP)'**.

Mamman is the acting departmental coordinator of Computer Science and Technology (2015- to date) at ESEP-Le Berger Universite Cotonou and a visiting lecturer to a University.

Mamman developed the Computing Department of ESPAM-FORMATION University, Cotonou (2011-2014) as the department chair, when the department population was still about 40 students.

He is still an average leader in the subject of **Web Programming**. He has had fun teaching and writing about **Data/Telecommunication Networks, Concept of Programming Languages**, *Android Programming, Internet Programming, Computing Research, Object Oriented Languages and Artificial Intelligence with Prolog.*

He spent the summer of 2013 and 2014 at the Blackberry Developers Conference Android-Platform Z10 and Z30. He spent the autumn of 2013 at University of Lagos Conference of Peoples, Land, and Water: The Natural Connection. ***He also spent the summer of 2016 at the Oxford Research and Publication International Conference: held at Usmanu Danfodio University, Sokoto.***

Mamman has written several books and published several papers in international journals and conferences. He also develops Computer Science and Technology curriculum for both undergraduate and post graduate programmes.

Mamman had a bright early career and has received several scholarships and research grants. He also received a postgraduate study

research grant in Computer Science by research and thesis from University of Kent Canterbury-U.K (2011).
Mamman is an average leading authority on computer programming, computational research and computational intelligence.