Web Engineering

3rd Edition



Web Engineering

3rd Edition



www.bibliotex.com

WEB ENGINEERING

3RD EDITION



www.bibliotex.com
email: info@bibliotex.com

e-book Edition 2023
ISBN: 978-1-98468-225-3 (e-book)

This book contains information obtained from highly regarded resources. Reprinted material sources are indicated. Copyright for individual articles remains with the authors as indicated and published under Creative Commons License. A Wide variety of references are listed. Reasonable efforts have been made to publish reliable data and views articulated in the chapters are those of the individual contributors, and not necessarily those of the editors or publishers. Editors or publishers are not responsible for the accuracy of the information in the published chapters or consequences of their use. The publisher assumes no responsibility for any damage or grievance to the persons or property arising out of the use of any materials, instructions, methods or thoughts in the book. The editors and the publisher have attempted to trace the copyright holders of all material reproduced in this publication and apologize to copyright holders if permission has not been obtained. If any copyright holder has not been acknowledged, please write to us so we may rectify.

Notice: Registered trademark of products or corporate names are used only for explanation and identification without intent of infringement.

© 2023 3G E-learning LLC

In Collaboration with 3G E-Learning LLC. Originally Published in printed book format by 3G E-Learning LLC with ISBN 978-1-98467-975-8

EDITORIAL BOARD



Aleksandar Mratinković was born on May 5, 1988. in Arandjelovac, Serbia. He has graduated on Economic high school (2007), The College of Tourism in Belgrade (2013), and also has a master degree of Psychology (Faculty of Philosophy, University of Novi Sad). He has been engaged in different fields of psychology (Developmental Psychology, Clinical Psychology, Educational Psychology and Industrial Psychology) and has published several scientific works.



Dan Piestun (PhD) is currently a startup entrepreneur in Israel working on the interface of Agriculture and Biomedical Sciences and was formerly president-CEO of the National Institute of Agricultural Research (INIA) in Uruguay. Dan is a widely published scientist who has received many honours during his career including being a two-time recipient of the Amit Golda Meir Prize from the Hebrew University of Jerusalem, his areas of expertise includes stem cell molecular biology, plant and animal genetics and bioinformatics. Dan's passion for applied science and technological solutions did not stop him from pursuing a deep connection to the farmer, his family and nature. Among some of his interest and practices counts enjoying working as a beekeeper and onboard fishing.



Hazem Shawky Fouda has a PhD. in Agriculture Sciences, obtained his PhD. From the Faculty of Agriculture, Alexandria University in 2008, He is working in Cotton Arbitration & Testing General Organization (CATGO).



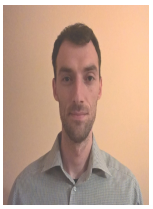
Felecia Killings is the Founder and CEO of LiyahAmore Publishing, a publishing company committed to providing technical and educational services and products to Christian Authors. She operates as the Senior Editor and Writer, the Senior Writing Coach, the Content Marketing Specialist, Editor-in-Chief to the company's quarterly magazine, the Executive and Host of an international virtual network, and the Executive Director of the company's online school for Authors. She is a former high-school English instructor and professional development professor. She possesses a Master of Arts degree in Education and a Bachelor's degree in English and African American studies.



Dr. Sandra El Hajj, Ph.D. in Health Sciences from Nova Southeastern University, Florida, USA is a health professional specialized in Preventive and Global Health. With her 12 years of education obtained from one of the most prominent universities in Beirut, in addition to two leading universities in the State of Florida (USA), Dr. Sandra made sure to incorporate interdisciplinary and multicultural approaches in her work. Her long years of studies helped her create her own miniature world of knowledge linking together the healthcare field with Medical Research, Statistics, Food Technology, Environmental & Occupational Health, Preventive Health and most noteworthy her precious last degree of Global Health. Till today, she is the first and only doctor specialized in Global Health in the Middle East area.



Fozia Parveen has a Dphil in Sustainable Water Engineering from the University of Oxford. Prior to this she has received MS in Environmental Sciences from National University of Science and Technology (NUST), Islamabad Pakistan and BS in Environmental Sciences from Fatima Jinnah Women University (FJWU), Rawalpindi.



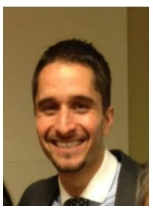
Igor Krunic 2003-2007 in the School of Economics. After graduating in 2007, he went on to study at The College of Tourism, at the University of Belgrade where he got his bachelor degree in 2010. He was active as a third-year student representative in the student parliament.Then he went on the Faculty of science, at the University of Novi Sad where he successfully defended his master's thesis in 2013. The crown of his study was the work titled Opportunities for development of cultural tourism in Cacak". Later on, he became part of a multinational company where he got promoted to a deputy director of logistic. Nowadays he is a consultant and writer of academic subjects in the field of tourism.



Dr. Jovan Pehcevski obtained his PhD in Computer Science from RMIT University in Melbourne, Australia in 2007. His research interests include big data, business intelligence and predictive analytics, data and information science, information retrieval, XML, web services and service-oriented architectures, and relational and NoSQL database systems. He has published over 30 journal and conference papers and he also serves as a journal and conference reviewer. He is currently working as a Dean and Associate Professor at European University in Skopje, Macedonia.



Dr. Tanjina Nur finished her PhD in Civil and Environmental Engineering in 2014 from University of Technology Sydney (UTS). Now she is working as Post-Doctoral Researcher in the Centre for Technology in Water and Wastewater (CTWW) and published about eight International journal papers with 80 citations. Her research interest is wastewater treatment technology using adsorption process.



Stephen obtained his PhD from the University of North Carolina at Charlotte in 2013 where his graduate research focused on cancer immunology and the tumor microenvironment. He received postdoctoral training in regenerative and translational medicine, specifically gastrointestinal tissue engineering, at the Wake Forest Institute of Regenerative Medicine. Currently, Stephen is an instructor for anatomy and physiology and biology at Forsyth Technical Community College.



Michelle holds a Masters of Business Administration from the University of Phoenix, with a concentration in Human Resources Management. She is a professional author and has had numerous articles published in the Henry County Times and has written and revised several employee handbooks for various YMCA organizations throughout the United States.

TABLE OF CONTENTS

Preface xi

CHAPTER 1 INTRODUCTION TO WEB TECHNOLOGIES 1

Introduction 2
Web Technologies: Importance And Carrier 3
Careers in Web Technologies 6
Job Role of Web Developer 7
How the Website Works? 11
Client And Server Scripting Languages 13
Domains and Hosting 15
Responsive Web Designing 17
Websites Creation 18
Types of Websites 24
Web Standards 26
W3C Standard 27

CHAPTER 2 HTML 37

Introduction 38
Overview of HTML 38
Structure 39
Types of Codes 40
An Evolving Language 40
Viewing a Web Page's Code 41
As Compared to HTM 41
Basic HTML Document 42
HTML Tags 42
HTML Document Structure 44
The <!DOCTYPE> Declaration 45
Creating an HTML Document 45
Basic Tags of HTML 46
Heading Tags 48
Paragraph Tag 50

Line Break Tag 51
Centering Content 52
Horizontal Lines 53
Preserve Formatting 54
Nonbreaking Spaces 55
Elements of HTML 56
HTML Tag vs. Element 56
Nested HTML Elements 56
Attributes of HTML 57
Core Attributes 58
Titled Heading Tag Example 60
Internationalization Attributes 61
HTML Formatting 64
Bold Text 64
Italic Text 65
Underlined Text 66
Strike Text 66
Monospaced Font 67
Superscript Text 68
Subscript Text 69
Inserted Text 69
Deleted Text 70
Larger Text 71
Smaller Text 72
Grouping Content 72
Phrase Tags in HTML 74
Emphasized Text 75
Marked Text 75
Strong Text 76
Text Abbreviation 77
Acronym Element 78
Text Direction 78
Special Terms 79
Quoting Text 80
Short Quotations 81
Text Citations 82
Computer Code 82
Keyboard Text 83
Programming Variables 84
Program Output 85
Address Text 86
Images in HTML 86

HTML Images Syntax 87
The alt Attribute 87
Image Size - Width and Height 88
Width and Height, or Style? 89
Images in Another Folder 91
Images on Another Server 92
Animated Images 92
HTML - Tables 94
Table Heading 95
Cellpadding and Cellspacing Attributes 97
Colspan and Rowspan Attributes 98
Tables Backgrounds 100
Table Height and Width 101
Table Caption 102
Table Header, Body, and Footer 104
Nested Tables 106
LISTS IN HTML 107
HTML Unordered Lists 108
The type Attribute 109
HTML Ordered Lists 111
The type Attribute 113
The start Attribute 117
HTML Definition Lists 119
HTML FRAMES 120
Disadvantages of Frames 120
Creating Frames 120
The <frameset> Tag Attributes 123
The <frame> Tag Attributes 124
Browser Support for Frames 124
Frame's name and target attributes 125

CHAPTER 3 EXTENSIBLE MARKUP LANGUAGE (XML) 131

Introduction 132
Basic Concept Of XML 133
XML Usage 139
Markup 140
Declaration of XML 141
XML Tags 142
Character Entities 144
Validation of XML 146
XML Schemas 147

Applications of XML 152
Simple API For XML 154
Benefits of SAX 154
Drawbacks of SAX 156
XML processing with SAX 156
Document Object Model (DOM) 157
Model of DOM 158
Nodes of DOM 161
Node Tree 162
Methods of DOM 163
Loading of DOM 165
Traversing of DOM 172
Navigation of DOM 176
Accessing of DOM 183

CHAPTER 4 CASCADING STYLE SHEETS (CSS) 189

Introduction 190
Overview of CSS 191
History of CSS 191
Significance of Cascading Style Sheets 192
Advantages of CSS 193
Who Creates and Maintains CSS? 193
Inline Styles 194
Embedded Style Sheets 195
Conflicting Styles 199
Linking External Style Sheets 203
W3C CSS Validation Service 206
Positioning Elements 207
Backgrounds 212
Element Dimensions 215
Text Flow And The Box Model 218
User Style Sheets 226

CHAPTER 5 WEB SERVERS AND SERVLETS 233

Introduction 234
Concept of Web Servers And Servlets 235
Web Server Working 238
Architecture 239
Tomcat Web Server 240
Servelets 241
Servlets-Environment Setup 244

Lifecycle of a Servlet 247
JSDK 254
The Servelet AP 255
Reading Http Request Parameters 258
Reading Initialization Parameters 260
The javax.servelet HTTP Package 264
Servlets - Client HTTP Request 267
Servlets - Session Tracking 271

CHAPTER 6 JAVASERVER PAGES (JSP) 279

INTRODUCTION 280
OVERVIEW OF JAVASERVER PAGES (JSP) 281
Why Use JSP? 282
Advantages of JSP 282
Life Cycle of a JSP Page 283
Creating a simple JSP Page 286
How to run a simple JSP Page? 287
Setup The JSP Environment 287
Setting up Java Development Kit 287
Setting up Web Server: Tomcat 288
Setting up Classpath 290
JSP Architecture 291
JSP Processing 291
Syntax In JSP 293
Elements of JSP 293
JSP Directives 304
JSP The page Directive 305
The include Directive 306
The taglib Directive 307

CHAPTER 7 JAVASCRIPT EVENTS 311

Introduction 312
JavaScript Events 312
Mouse Events 313
JavaScript Event Types 316
User Interface Events 316
Focus and Blur Events 317
Mouse Events 318
Keyboard Events 319
Form Events 320
Mutation Events and Observers 320
HTML5 Events 321

CSS Events 322
JavaScript addEventListener() 322
Syntax 322
Parameter Values 323
Event Bubbling or Event Capturing 328
JavaScript onclick Event 330
Syntax 331
JavaScript dblclick event 335
Syntax 336
JavaScript onload 340
Syntax 340
JavaScript onresize event 344
Syntax 345

CHAPTER 8 WEB API ESSENTIALS 353

Introduction 354
Application Programing Interface 355
Explanation 356
Documentation 357
API in Object-oriented Languages 358
API Libraries and Frameworks 359
API and Protocols 360
Web APIs 362
Implementations 363
Language Bindings and Interface Generators 365
Create Web API project 365
Web API Controller 373
Configure Web API 378
API Uses 380
Operating Systems 381
Remote APIs 382
Use in Web APIs 383
API Testing 385
Challenges with API Testing 387
Start API testing 389

INDEX 393

PREFACE

Web engineering, an emerging new discipline, advocates a process and systematic approach to developing high-quality web-based systems. It promotes the establishment and application of sound scientific, technical and managerial principles, as well as disciplined and systematic approaches to the development, deployment and maintenance of web-based systems. Earlier web developers considered web engineering as a specialized field of software engineering, but it encompasses more than it, since every web application involves content, hypertext structure and presentation to a potentially very diverse audience, web developers need a wide range of domain knowledge provide. Web engineering is a holistic approach and deals with all aspects of web-based system development, starting with conception and development through to implementation, performance evaluation and ongoing maintenance. Building and deploying a web-based system involves several iterative steps. Most web-based systems are constantly evolving to keep information current and to meet user needs. Web engineering represents a proactive approach to building web applications. Web engineering methodologies have been successfully applied to a number of web applications. In this edition of the book, the text has been revised and restructured, and a new chapter has been added. This edition contains eight chapters. With explicit descriptions and scripting languages such as VBScript, JavaScript, and more, this book will be a valuable tool for anyone involved in web engineering.

Chapter 1 starts with an overview of web technologies. Knowing the basic categories of web technologies is essential if you plan to work in web development.

Chapter 2 describes the structure of Web pages as HTML elements are the building blocks of HTML pages.

Chapter 3 covers the Extensible Markup Language (XML) concepts. In third chapter; Cascading Style Sheets (CSS); Web Servers and Servlets; and Java Server Pages (JSP) programming technology that enables the creation of dynamic web pages and applications, are covered.

Chapter 4 focuses on Cascading Style Sheets (CSS). CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript. CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts.

Chapter 5 sheds light on Web Servers and Servlets. A web server is computer software and underlying hardware that accepts requests via HTTP (the network protocol created to distribute web content) or its secure variant HTTPS.

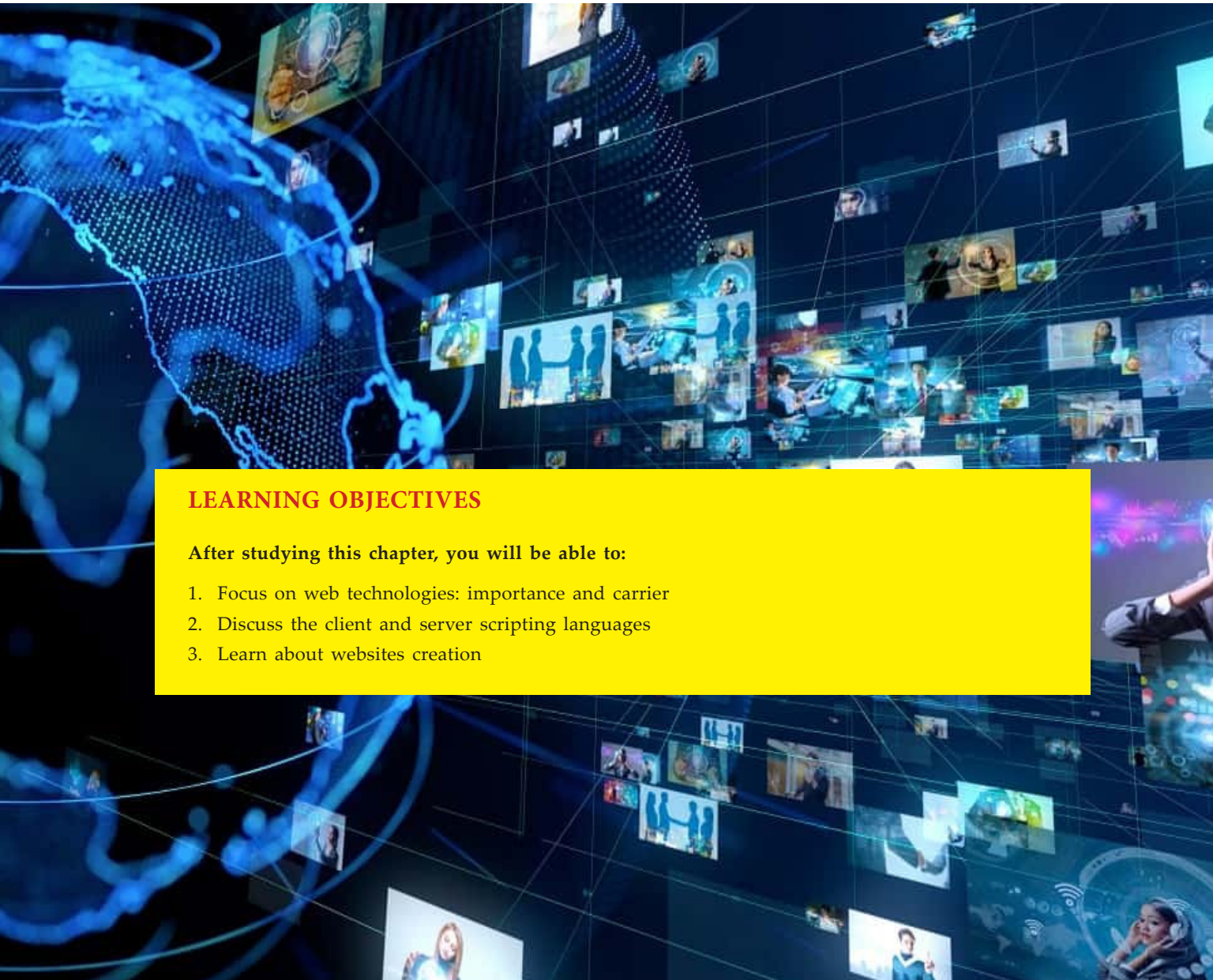
Chapter 6 lays emphasis on JavaServer Pages (JSP) that helps software developers create dynamically generated web pages based on HTML, XML, or other document types.

Chapter 7 deals with JavaScript Events. The Javascript interacts with the documents HTML code using events, which are triggered when a particular moment of interest happens in the document or the browser window. Javascript Events makes the webpages interactive and responsive.

Chapter 8 in this edition focuses on web API essentials usually limited to a web application's client-side (including any web frameworks being used), and thus usually does not include web server or browser implementation details such as SAPIs or APIs unless publicly accessible by a remote web application.

CHAPTER 1

INTRODUCTION TO WEB TECHNOLOGIES



LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Focus on web technologies: importance and carrier
2. Discuss the client and server scripting languages
3. Learn about websites creation

INTRODUCTION

Web Technology refers to the various tools and techniques that are utilized in the process of communication between different types of devices over the internet. A web browser is used to access web pages. Web browsers can be defined as programs that display text, data, pictures, animation, and video on the Internet. Hyperlinked resources on the World Wide Web can be accessed using software interfaces provided by Web browsers. You probably know that computers don't communicate with each other the way that people do. Instead, computers require codes, or directions. These binary codes and commands allow computers to process needed information. Every second, billions upon billions of ones and zeros are processed in order to provide you with the information you need.

The methods by which computers communicate with each other through the use of markup languages and multimedia packages is known as web technology. In the past few decades, web technology has undergone a dramatic transition, from a few marked up web pages to the ability to do very specific work on a network without interruption. Let's look at some examples of web technology.

Web technologies are often divided according to where they are used. That is, are they focused on the client-side or server-side of web applications? Knowing the basic categories of web technologies is essential if you plan to work in web development. Thus, here is a list of web development technologies by category:

- Browsers
- HTML and CSS
- Programming Languages
- Frameworks
- Web Servers
- Databases
- Protocols
- Lastly, data Formats

Although many technologies are involved in creating a web application, some are more central to building a web application.

WEB TECHNOLOGIES: IMPORTANCE AND CARRIER

Web technology is the establishment and use of mechanisms that make it possible for different computers to communicate and share resources. Web technologies are infrastructural building blocks of any effective computer network: Local Area Network (LAN), Metropolitan Area Network (MAN) or a Wide Area Network (WAN), such as the Internet. Communication on a computer could never be as effective as they are without the plethora of web technologies in existence. Web technology is the development of the mechanism that allows two of more computer devices to communicate over a network. For instance, in a typical office setting, a number of computers plus additional devices such as printers may be interconnected via a network, allowing for quick and convenient transmission of information. The processes involved in web technology are complex and diverse, which is why major businesses employ whole departments to deal with the issue. Web technology has revolutionized communication methods and has made operations far more efficient.

The main advantage of web technology is that it offers convenience and a high speed of communication in the computer world. Whether in the office or the home, processes using a computer are more swift and straightforward with the use of a network. Web technology allows messages to be sent around a system, whereas before it may have been necessary to employ a runner or leave your workspace to communicate a message. It is clear to see how web technology reduces costs and makes a company more efficient, raising business potential.



Matters involving web technology can be very complicated, and it would be difficult for someone without relevant experience to sort a network problem out. This means it is necessary to employ someone with the specific skills to solve network issues, which costs money. Additionally, the existence of a network provides the opportunity for an attack on the computer system. Weaknesses in a network could be exploited; important information could be stolen or destroyed and malware could infect the various network systems. For this reason, network security is another issue that must be considered when using web technology.

A technology which acts as in interface between web servers and clients. It includes markup languages, programming interfaces, standards to define document identity and display. Basically web technology is the process which allows two more computing devices over a network. In order to face the new market environment which is in constant change, the company must place the customer in the center of its attention. As a result, the company will not follow, first of all, the benefit brought by a certain successful business, but to develop long-term business relationships with the same customers. The integration of Web technologies has an important place into the process of accomplishing companies' objectives to increase the competitiveness degree on the market by generating customers' loyalty. Developing a web-site makes it possible a very good communication with the clients, and this leads, finally, to a constant adaptation of the company's offer to the continuously changing customers' requests.

Web technology is interpreted with more scripting languages, i.e. if the web developer writes a code which is easily understandable by the web browsers. On the other-hand, compiled scripting languages needs translator to interpret the code into machine language code. Different web technologies have different server dependencies, so before choosing the particular web technology, determine the services hosted by that technology.

List of Different Web Technologies

ASP: ASP (Active Server Pages), today intuitively offers many web application development services to the clients across the

world. It is one of the popular web technology created by Microsoft to develop dynamic web pages, applications and web services. This web hosting technology creates codes using VB Script and Java Script languages that can be developed and processed on all computers through windows operating system. The .net development company's services have brought innovation in the field of web development.

PHP: PHP (Hypertext Preprocessor) is a server side scripting language designed for web development to produce dynamic web pages. It is a web technology that offers choices other than ASP UNIX based servers. The potential of PHP web application is that the PHP code is embedded into the body of an HTML page document which makes it a great tool for web development. In the recent times, for most of the web application PHP development India becomes the obvious choice for various ecommerce and web portal designers.

JSP/JavaScript: JSP (Java Server Pages) web technology enables web developers and designers to create informative dynamic web pages. With the intro of JSP, website owners find revolutionary changes in the development of websites. It is a prototype based scripting language that supports dynamic, weakly typed and first-class functions. JSP promotes rapid development of platform independent web applications. The main feature of JavaScript scripting language is that it can be used to create dynamic HTML codes and almost every popular browsers support this web technology.

Perl: Perl is a script based programming language that offers huge number of uses to web application development technology. It includes major features like multiple programming paradigms (Procedural, object-oriented, and functional styles), reference built-in support for text processing, and a large collection of third-party modules.

Perl scripts are very simple & easy-to-use and are used by Unix/Windows administrators widely. It is similar to C programming language and used in UNIX-based servers. Using Perl, the programmers can automate tasks, send automatically generated emails, monitor logs, and transfer files, logs and data.

CGI: CGI (Common Gateway Interface) similar to ASP and PHP, is one of the most widely user server applications on the

internet. CGI applications can be written in any programming languages like C, C++, Java, and Perl etc. to design & develop dynamic and interactive web pages. Some of CGI scripts you can find are web site management, password protection, shopping carts and many more. Perl is the most popular scripting language used for CGI programming.

VBScript: VBScript (Visual Basic Scripting Edition) is an Active Scripting language that is easy to learn and commonly used for server side scripting.

Visual Basic .NET: VB.NET (Visual Basic .NET) is an object oriented computer programming language & advanced form of visual basic programming language that hosts number of features to help web developers to create bug free custom web applications in a simple manner.

Careers in Web Technologies

Literally any one can! The job of a web developer is that of a programmer who can write efficient codes in various languages used for the web. "Becoming a good programmer" does not actually need any formal education. You just need to decide on which language you want to learn, buy a couple of books for that language, read tutorials on internet & then try your hands on coding. You need to write lots of code yourself, learn by developing simple & small applications initially & then move up yourself to big & challenging applications. That is the only way you can become a good programmer. However having a formal education in the following streams will definitely make your job easier graduation in mathematics, engineering in computer sciences & electronics, graduation in computer sciences. The reason is these courses teach basics of programming as part of their curriculum. But at the same time anyone can become a good programmer by learning themselves. All you need is a taste for logic & programs

Web technology is a blanket term that can be used to cover all the people who plan, build, and manage a given website-which can include everyone from product and project managers to writers, designers, information architects, programmers, and database administrators. Web professionals are charged with nothing less than conceiving, designing, building, programming,

populating with content, branding, marketing, supporting, and managing websites.

Web development combines a large dose of creativity with a large background in technical knowledge. Although much of what one needs to know as a Web developer can be gathered in college classes, preparation in high school can help. Some high schools might be better equipped to handle the creative end than the technical end.

Web developers, like most computer specialists, are scattered throughout every segment of the U.S. economic scene. However, the greatest concentration is in retail business, especially market segments that rely on a strong online presence. These businesses require a great expertise in designing pages that are easily navigable for the customer.

Also, as in other computer-based careers, the job market is growing faster than the national average and is expected to continue for the next several years. Computer skills remain among the better-paying positions, especially for recent graduates. Top Web developers, who combine creativity with technical know-how, can command salaries high above the average computer professional.

Job Role of Web Developer

The job of the Web development team is to create a compelling website. Such websites, generally speaking, are designed to support a business, be it selling things (Amazon.com) or enabling other kinds of transactions (Charles Schwab & Co., eBay), providing financial advice (The Motley Fool) or other online content (The Onion), or helping people search the Web (Google).

Web developers conceive of the website strategy, working for or in consultation with the decision-makers at a company. They figure out the hardware that the site should use, the software necessary to make it function properly, the design and navigation that will get the public to use the site in a way that will support its business, and the information that will keep users coming back. Web developers also program the site so that it functions effectively, adding tools like community discussions and newsletter sign-up capability. They also set up reporting

tools and databases to record traffic to the site and what visitors are doing there. (Buying things? Chatting with others? Clicking through on banner or popup ads? Reading?)

These days, because Web surfers are increasingly accessing the Internet via wireless devices, be they Wi-Fi- or Bluetooth-enabled computers, cell phones, or personal digital assistants, Web dev professionals are increasingly facing the need to optimize the websites they run for wireless devices.

Some Web development positions require technical skills. Creating the back end of a website (the database and hardware infrastructure on which it sits and through which orders are fulfilled, for instance) and the front end (the design and navigation and tools used by site visitors, such as stock quotes, relocation calculators, polls) requires programming skills.

Other positions may require a familiarity with technology, but not technical skills. Producing a site-coordinating between front-end and back-end developers, making sure the site supports the company's brand, working with users and designers to come up with a navigable site, ensuring that the content supports the site's business objectives-does not necessarily require high-level technical skills, though a familiarity with technology is generally a prerequisite.

Product management and content development jobs can also require an understanding of a specific industry or business niche; for instance, a potential content developer at an investment-advice website will typically need to display an understanding of stocks and the markets to get the job.

Content developers write, edit, shape, and aggregate information. Project managers work across functions to make sure projects get finished on time. Their role requires exceptional communication and organization skills. Since all roles typically interact with others, interpersonal skills are a must in this field

A decade ago, a person with a little computer expertise and some HTML skills might be considered a Web developer simply because few people had better backgrounds. Fortunately (or unfortunately), Web development has come a long way since then. Web development has become a profession with standards. More expertise than ever is needed to succeed.

Web developers produce sites and systems on the World Wide Web for organizations. The sites might be purely for information sites, but they are based on Web information systems that combine analysis, design, and authoring with multimedia development and traditional computing skills, particularly programming and using databases.

A Web developer, or Internet developer, is often responsible for site design, creation and day-to-day operation. Again, the job title might differ from company to company. Many companies contract with consultants or freelancers rather than train staff to create an entire site. They might use an on-site staff to keep the site operating and to update it periodically. Other companies might hire a Web developer to create the site and maintain it once it is in operation.

Work in entry-level Web development can include:

- Gathering relevant content and information from key company personnel;
- Developing the Web design concept and page organization;
- Presenting the site for approval or refinement from company personnel;
- Designing, building, and testing Web pages and links; and
- Updating contents and maintaining the site.

Developers who design sites from the initial concept must gather data to identify client needs and turn the data into functional and technical specifications for a Web site. This can include selecting programming languages and writing the code. Developers can recommend Web hardware and software, evaluate Web technologies, and develop standards for the site.

The responsibilities or required skills for a Web developer can be divided into three categories. Design skills involve content development, graphics and layout, and security. Technical skills are needed to write code, develop databases, and implement testing. Management skills are used to evaluate the hardware and software systems, train other designers and developers, coordinate with other company personnel, and provide customer service. Web developers have to evaluate the needs of the individual or company for whom they are creating a site. They assist

in planning an Internet strategy that allows those goals to be achieved. A Web developer first must talk with Web site owners and users to understand what the Web site owners want their Web site to look like and to do.

Web sites can consist of intranets, which are confined to internal employees, and extranets, which include groups of external users. Developers must design a site that allows specific users to have appropriate access. They have to develop, assess, and communicate security policies and standards among these constituencies.

Once the needs and desires have been assessed, the developer might create a storyboard to help the Web site owner understand what the site will look like. These storyboards are collections of sketches, much like the tools that movie and television directors use to plot one scene's transition to the next. The Web storyboard simulates transitions between pages showing how the user will move through the site and to make sure the pages flow logically. If the storyboard is approved, then the building of the site can begin.

The creative side is followed by the technical side. Web developers establish the tools and software required to produce and operate the Web environment. A variety of software is needed to construct the site. A simple site might incorporate easy-to-use software such as FrontPage or Dreamweaver. More complicated sites are likely to be built from scratch using HTML in conjunction with other computer programming languages, such as Java or Visual Basic.

The variety of technology programs and applications seems endless, but the Web developer needs to remain up-to-date on the latest versions of common publishing tools. They must understand which applications work best with a particular system, or which is more likely to produce the desired expectations of the Web site owner. They test applications and sites to make sure that users can navigate easily while getting their transactions completed securely. Good Web developers encourage feedback to make adjustments and improvements as needed.

Management skills are a third necessity for the Web developer. Although he or she might spend considerable time in front of a computer, the Web developer must interact with others in the

company. This can include training people within that company to work with the system, or even selling company officials on the idea once it has been created. It can involve market research and consultations with management and various experts.

Sometimes the job requirements might extend to customer service. In these circumstances, developers must take feedback from Web site users, provide information, and update the site as needed. It is helpful to have some understanding of e-commerce to be better prepared for the business goals of a Web site and to understand the importance of solid customer relations.

Entry-level Web developers might be called upon to perform most or all the above duties. However, professionals that work for larger companies are more likely to have additional personnel who either assist with the duties or are assigned specific duties outright. In any case, each of these skill sets will be useful in succeeding as a Web developer.

How the Website Works?

Before trying to make your own web page and launch it on the Internet, first you need to know how web pages work. Here are the basic terms:

- The server receives the request for a page sent by your browser.
- The browser connects to the server through an IP Address; the IP address is obtained by translating the domain name.
- In return, the server sends back the requested page.

Web pages are written in HTML, Hypertext Markup Language. A markup language is a computer language that describes the layout, format and content of a page. The Web browser renders the page according to the HTML code.

Web servers are computers whose job is to respond to a browser's request for a web page and deliver it through the internet. Pages hosted on a web server can be displayed to anyone all over the world. It is like a hard drive that stores your website files and images. In order to host your page on a web server you need to pay a hosting charge.

In order to display your website on the internet, you need a web hosting provider. You could set up your own hosting server at home, but it would take a huge amount of knowledge and require a lot of time to set up. Paying a hosting service offers you a lot more freedom for your site, gives you opportunity to do professional work and is a reliable solution. To create your own website, you first need a domain name.

You can get the domain name from your web hosting provider or from a separate provider specialized in domains names. Sometimes web hosting providers offer you a domain name along with your hosting account. It may appear to be the easiest option but could be more costly.

There are a lot of web hosting providers available in a wide range of prices depending on the services such as number of domains, hosting space or bandwidth provided. Choose carefully, or you could face slow servers, periods of malfunction or no support.

When you enter something like Google.com the request goes to one of many special computers on the Internet known as **Domain Name Servers** (DNS). All these requests are routed through various routers and switches. The domain name servers keep tables of machine names and their IP addresses, so when you type in Google.com it gets translated into a number, which identifies the computers that serve the Google Website to you.

When you want to view any page on the Web, you must initiate the activity by requesting a page using your browser. The browser asks a domain name server to translate the domain name you requested into an IP address. The browser then sends a request to that server for the page you want, using a standard called Hypertext Transfer Protocol or HTTP.

The server should constantly be connected to the Internet, ready to serve pages to visitors. When it receives a request, it looks for the requested document and returns it to the Web browser. When a request is made, the server usually logs the client's IP address, the document requested, and the date and time it was requested. This information varies server to server.

An average Web page actually requires the Web browser to request more than one file from the Web server and not just

the HTML / XHTML page, but also any images, style sheets, and other resources used in the web page. Each of these files including the main page needs a URL to identify each item. Then each item is sent by the Web server to the Web browser and Web browser collects all this information and displays them in the form of Web page.

CLIENT AND SERVER SCRIPTING LANGUAGES

Client-side scripting is a process by which a web-based computer program runs on the user's computer rather than on the computer hosting the website. Specifically, it runs through a type of program known as a script, which is handled by the user's web browser. The main reason for client-side scripting is to allow a web page to be created specifically around the user's own data and options, rather than being a fixed page that always appears the same to every user.

This made websites very limited in dealing with large sets of data, such as a railway schedule. With static pages, the only solution was to print the schedule in full and let the user hunt down the relevant detail.

This problem was solved by the development of dynamic web pages. These can adapt to meet a specific situation, such as if a user is searching for a journey between two stations within a specific time period. The result of the search is displayed through a dynamic web page, which is automatically created for that query.

Server-side scripting is a term primarily used with regards to serving custom content via the hypertext transfer protocol (HTTP) on a web server by having the server execute small programs. These programs, usually written in a scripting language, are executed by the server when a client request arrives. Depending on the parameters given by the client at the time of the request, the script then generates a web page for the client. Web pages created in this way are often referred to as dynamic pages. This is in contrast to a client-side script that is sent from the server to the requesting client and then executed.

The server-side scripts were often times small, executable files containing a series of commands to pass to the operating system. The web daemon, the software running the web server, would use these shell scripts to further execute an additional program residing on the host computer. This general technique was then defined in 1993 in the common gateway interface (CGI) standard developed by the Internet Engineering Task Force (IETF). In most cases, these early CGI scripts were used to send database responses from the server back to the requesting client.

As interpreted scripting languages were developed, such as Perl, and PHP: Hypertext Preprocessor (PHP), server-side scripting methods also evolved. HTTP daemons were updated to include use extensions that allowed for these various scripting languages to be called from the web daemon itself, instead of being passed to the operating system on the host. With this addition, bits of scripting language could be included within the HTML document. As the web server reads through a document before sending it out across the web, it checks for and executes any of the script within the document. Since the content called up by server-side scripting resides in a database, it can be virtually anything. Content such as product descriptions, price variations, weblog entries, images, and even formatting is stored in the database. It is also possible to nest one server-side script snippet within another, where the first script grabs certain data from the database, but also calls a second script which accesses secondary data. This is useful, for example, when delivering some fairly custom standard content with additional nested content that may be time sensitive or as yet unknown, such as comments on a weblog entry. Web site maintenance, then, involves updating data in the database, which will then affect every page on the website with the included script to call that data.

Many of the server-side scripting techniques have been further developed into what are known as content management systems (CMS). PHP is probably the most frequently used language for this purpose, running at the core of many CMS implementations. Here, a user operating a website running on a CMS will edit her HTML documents to include what are called tags. The tags are essentially cues to the CMS to tell it what script to include in the document. The CMS can then be customized by creating additional scripts and their associated tags.

Domains and Hosting

The domain name is much like an entry in a phone book. Computers communicate by using numbers, called IP addresses, to contact each other, much like you use a phone number to dial a specific person's phone. If you want people to find your business's phone number, you want to be listed in a phone book.

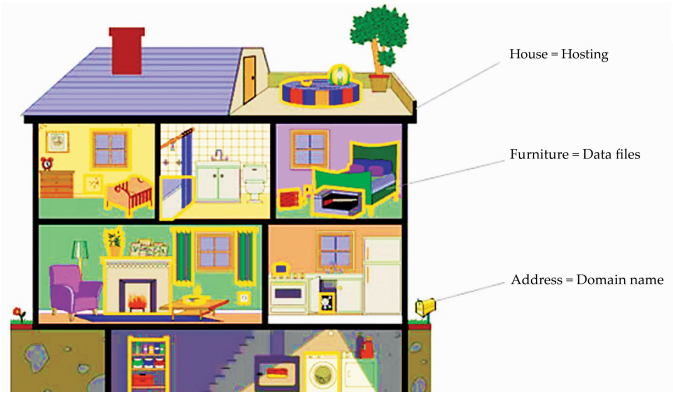
The phone book tells people looking for your phone number "Company A's Phone number is xxx-xxx-xxxx" just as a domain tells people (i.e. their computers) "domainA.com is hosted on the server xxx.xxx.xxx.xxx" Without the domain, you would have to tell your customers "Hey my site is located at 123.456.789.123/~mysite/" instead of "mysite.com" You can see how, without a domain, having a site or hosting is impractical.

The web-hosting or server portion is much like the space that you rent out to have your business in. It is merely the space itself. It does not include furnishings like shelves for your products, just as the web-hosting account does not include a site for you to sell your products.

Luckily, in the web-hosting world, it is very easy to furnish the space provided by your host, because you can install many framework applications through the Fantastico icon within your cPanel. Without the hosting services, you would not have a place for your files to reside, so your domain would then become like a disconnected phone number in the phone directory, and your site files would have nowhere to stay.

Domain Name and Web Hosting Explained

To simplify: A domain name, is like the address of your home; web hosting on the other hand, is the space of your house where you place your furniture. Instead of street name and area code, set of words or/and numbers are used for the website's naming'. The same goes with hosting, computer hard disk and computer memory are used instead of instead of wood and steel for storing and processing data files. The idea is presented clearer with the diagram below.



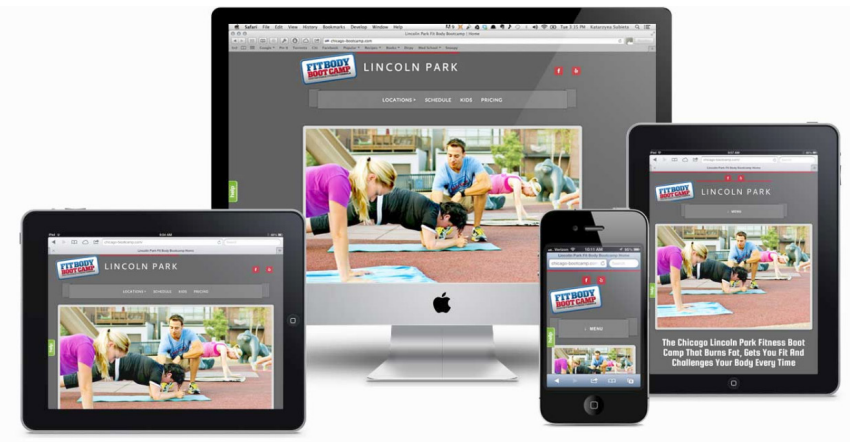
When you register a domain, it gives you sole ownership and rights to the name of your site. No one else in the market has the access to the actual name of that particular domain besides you. A domain registration is a "right to use and control" a domain name. It is _not_ related to any content or services associated with that domain. Domain registrations are quite simple. They are valid for a period of time, they generally have some contact information associated with them, and may have some DNS servers too. (DNS servers are often called name servers.)

A web hosting normally refers to the web server (big computer) that stores lots of data files). A web hosting providers normally rent out web servers and network connection to the end-users or the resellers. For most cases, the hosting providers will be the parties handling most server maintenance work (such as backup, root configuration, maintenance, disaster recoveries, etc.); but for certain cases, the end users will need to get everything cover by themselves.

Web-Hosting is a set of services. We configure our web-servers to hand out web pages and handle e-mail for your domain name. That sounds a lot simpler than it is. Web pages can be complex. They are often generated by programs which store and retrieve information from databases. The world of e-mail is not simple either. It can be as simple as forwarding messages to a mailbox at an existing ISP, or it can include multiple layers of filtering (virus, content analysis for spam detection, challenge/response) before being stored for later retrieval. The message might be retrieved by our Webmail system, or an e-mail client on a home or business computer like Outlook or Eudora.

Responsive Web Designing

Responsive Web design is the approach that suggests that design and development should respond to the user's behavior and environment based on screen size, platform and orientation. The practice consists of a mix of flexible grids and layouts, images and an intelligent use of CSS media queries. As the user switches from their laptop to iPad, the website should automatically switch to accommodate for resolution, image size and scripting abilities. In other words, the website should have the technology to automatically respond to the user's preferences. This would eliminate the need for a different design and development phase for each new gadget on the market. Responsive Web Design makes your web page look good on all devices (desktops, tablets, and phones). Responsive Web Design is about using CSS and HTML to resize, hide, shrink, enlarge, or move the content to make it look good on any screen:



Responsive web design (RWD) is an approach to web design aimed at crafting sites to provide an optimal viewing and interaction experience—easy reading and navigation with a minimum of resizing, panning, and scrolling—across a wide range of devices (from desktop computer monitors to mobile phones).

A site designed with RWD adapts the layout to the viewing environment by using fluid, proportion-based grids, flexible images, and CSS3 media queries, an extension of the media rule, in the following ways:

- The fluid grid concept calls for page element sizing to be in relative units like percentages, rather than absolute units like pixels or points.
- Flexible images are also sized in relative units, so as to prevent them from displaying outside their containing element.
- Media queries allow the page to use different CSS style rules based on characteristics of the device the site is being displayed on, most commonly the width of the browser.

Responsive web design is becoming more important as the amount of mobile traffic now accounts for more than half of total internet traffic. This trend is so prevalent that Google has begun to boost the ratings of sites that are mobile friendly if the search was made from a mobile device. This has the net effect of penalizing sites that are not mobile friendly.

WEBSITES CREATION

Technology has really advanced over the past few years. The internet has grown faster than ever before. Facebook has taken over, Google is king, and normal folks like yourself can build your own website for free. More and more people are creating websites online as a way to express themselves and get in touch with others. Businesses cannot afford not to have websites as a way of reaching more potential customers. Building a website is a great way to share your ideas and thoughts with the world. But if you have never done one, it can seem daunting. There's all that http-dot-whatever and <tag this=””> and <tag that=””> and how do you get pictures and text in there? Well fear not, this will help you to grasp the intricacies very quickly!

Designing Your Website

1. Get inspired. Look at websites with great designs and think about why they are great designs. It usually comes down to the information, resources, links, and pages being laid out in a way that is easy to see and use. To get ideas about how to design your own site, look at sites which do similar things to get ideas about where you should put different types of content.

- Stay realistic to your skills.
- Ease of access is the most important thing. If you do not have a certain piece of information easily visible, make sure that getting to that information is very logical.
- Generally the simpler the design, the fewer the pages, the better.

2. Choose a topic and purpose. If you already have a fairly good idea about what your website will focus on, skip this step. If not, here are some things to help you figure that out. First, understand that there are billions of people on the Internet, and a large percentage have websites. If you limit yourself to something that has not been done, you'll never get started.

- When you think, "Internet," what is the first thing that comes to your mind? E-commerce? Music? News? Socializing? Blogging? Those are all good places to start.
- You could create a website that's dedicated to your favorite band, and have a chat area where people can talk about it.
- You can build a page for your family, but be careful about things like this. The internet is full of unsavory characters and information you put up about your family could end up being used against you. Consider adding password protection to your personal family website.
- If you are a news junkie, or want something less filtered than traditional media, build a website and get publicly available feeds from news providers such as Reuters, BBC, AP, and others. Build your own customized news aggregator (what used to go by the quaint name of "newspaper"), then see and show all the news that's fit to digitize.
- If you are creative at writing then you can start a blog where you can write about anything you want and attract monthly readers!

3. Make a plan. Building your website is going to take a commitment of time and possibly money, so set a limit on both, and then dig in. The plan does not have to be a big, complicated spreadsheet, or a fancy graphic presentation, but at the very least, you will want to consider what it will do for you and the visitors, what you'll put on the website, what goes where on the webpages.

4. Gather the content. There are lots of different types of content and many have their own considerations. You'll need to figure out what's best for your website and your needs. Some things to consider including:

- A store. If you want to sell things, you'll need to figure out how you want the items to be available. If you have relatively few things to sell, you might want to consider having a store with a hosting service. Society6, Amazon, and Cafepress are all well-established store hosts which let you sell a variety of items and set your own prices.
- Media. Do you want to display videos? Music? Do you want to host your own files or do you want them hosted somewhere else? YouTube and SoundCloud are great examples of hosting options, but you'll need to be sure that the way you design your website allows these media types to display correctly.
- Images. Are you a photographer? An artist? If you plan on putting original images on your website, you might want to use a format that can help keep them from getting stolen. Make sure the images are relatively small or that they are hidden behind some Flash code, so as to keep them from being easily saveable.
- Widgets. These are mini-programs which run on your website, usually to help you keep track of who visits, what they are looking for, and where they are from. You can also find widgets for booking appointments, displaying a calendar, etc. Look into what might be useful for you (just make sure the widget comes from a reputable source).
- Contact information. Do you want to have contact information on your webpage? For your own safety, you should be careful about what kind of information you have available. You should never display things like your home address or home phone number, as information like this can be used to steal your identity. You may want to set up a PO Box or a special email address for people to contact you at, if you do not have a business address.

5. Draw a flow chart. For most people, the website starts on the home page. This is the page that everybody sees when

they first go to www.yourSite.com. But where do they go from there? If you spend some time thinking about how people might interact with your site, you'll have a much easier time down the line when you are making navigation buttons and links.

6. Plan for user devices and situations. In recent years, smartphones and tablets have become incredibly popular platforms for browsing the internet, and they require websites to be designed for them. If you really want to make a website that will stand the test of time and be accessible to the highest number of viewers, plan on making different versions of your site for different devices, or plan to use a responsive design that adjusts as necessary.

Building Your Website

1. Decide what method or tool you'll use to build it. When you have the basic idea down and have a plan for how it will be laid out, the next to think about is how you are going to build it. The options seem endless, and people will try to sell you this or that 'fantastic' application, and every other thing that you "absolutely must have" on your site, however the reality is that there are a few great tools for building websites, and one of them will be best-suited to your situation and needs.

2. Build it yourself. This is the first option. If you have a website-building application like Adobe Dreamweaver, it is not very difficult to create a website from scratch. You might need to do some coding but do not panic! HTML looks complicated, but it is like listening to Shakespeare—it is hard at first, but once you get the feel of it, it is not that difficult.

- Pros: website design software simplifies the process of building sites by letting you drag-and-drop images, text, buttons, movies, and anything else you can think of, usually without ever having to dig into HTML. Many web design applications will even let you create sites specifically for your smart phone or pad. If you are building a basic, personal website, this is really a great way to go.
- Cons: there is a learning curve, and though you do not have to dig into HTML, it is not totally geek-free. If you are in a hurry, this might not be the best solution.

Perhaps the biggest con, though, is that if you are not a graphic designer, you could end up with a page that hurts the eyes. To mollify this somewhat, there are a number of free templates in the applications, and on the internet, but be aware of your limitations—if you have any!

3. Use a **content management system** (CMS). This is the second option. WordPress is an example of a great option for building websites. It helps you create web pages and blog posts quickly and easily, set up the menus, allow and manage user comments, and has thousands of themes and plugins that you can choose from and use for free. Drupal and Joomla are other great CMS options. Once the CMS is hosted, you can manage your site from anywhere (in the world) that has an Internet connection.

- Pros: Very easy to use, quick to get started with one click install, and lots of options for the beginner (with enough depth for more experienced users).
- Cons: Some themes are limiting, and not all are free.

4. Build the website from scratch. This is the third option. If you decide to build your website from scratch, you'll need to start using HTML and CSS. There are ways to extend your HTML skills and add more features and more depth to your website. If you are developing a professional website, these tools will help you get that edge that is needed in any business venture.

- CSS, which stands for "Cascading Style Sheets". CSS gives more flexibility for styling the HTML, and makes it much easier to make basic changes—fonts, headers, color schemes—in one place, and have those changes ripple through the site.
- XHTML is a web language set by W3C's standards. Almost identical to HTML, it follows a stricter set of rules for marking up information. What this means, for the most part, is minor changes to the way you write code.
- Look into HTML5. It is the fifth revision of the core HTML standard, and will eventually subsume the current version of HTML (HTML4), and XHTML as well.
- Learn a client-side scripting language, such as JavaScript. This will increase your ability to add interactive elements to your site, such as charts, maps, etc.

- Learn a server-side scripting language. PHP, ASP with JavaScript or VB Script or Python can be used to change the way web pages appear to different people, and lets you edit or create forums. They can also help store information about people who visit your site, like their username, settings, and even temporary "shopping carts" for commercial sites.
- AJAX (Asynchronous JavaScript and XML) is a technique of using a browser sided language and a server sided language to make the page get new information from the server without refreshing the page, often greatly reducing user wait time and vastly improving the user's experience but increasing bandwidth usage. For a website that will see a lot of traffic, or an eCommerce site, this is an excellent solution.

5. Hire a professional. This is the fourth and final option. If you are not up to designing your own website, or learning new coding languages—especially for more advanced sites—hiring a professional may be your best option. Before you hire, ask to see a portfolio of their work, and check their references carefully.

Test Driving Your Website and Going Live

1. Register your domain name. If you are on a budget, there are strategies for buying a cheap domain name. Find a domain name that is easy to remember and easy to spell. If you use domains ending with .com, you will end up with more traffic, but most of the easy ones are taken, so be creative!

- Look to Network Solutions, GoDaddy, or Register.com are good in US and uk2.net if you are in the UK to research and find the ideal domain name for your website. Wordpress also includes a feature whereby you can use a name that's tagged with their site, for example, mywebsite.wordpress.com. But if the name you choose is also available as a .com, they will notify you when you sign up.
- You can purchase domain names if they have been "parked" or are for sale online through business sales sites. It is a good idea to get legal and financial advice before purchasing an expensive domain name.

2. Inspect your website. Before you post your site, it is wise to test it thoroughly. Most web design software has a way to test your site without taking it online. Look for missing tags, broken links, search engine optimization, and website design flaws. These are all factors which may affect your website's traffic and revenues. You may also generate a free full-functioning site map to submit to search engines like Google, in a matter of minutes.

3. Test your website. When you finish your website, do usability testing. You can do this by asking a few friends or family members to try it out. Give them a specific task like "edit your profile" or "buy an alpaca sweater from the bargains page." Sit behind them and watch them navigate—do not help them. You will likely find areas where you need to improve navigation or clarify some instructions. Alternately use something like zurb.com to user test on specific demographics for different types of engagement. When testing a website in 2014 it is becoming important to keep the platform in mind and make sure the website is usable from smartphones and tablets as well as desktops. Keep a list of things you notice that seem difficult or non-intuitive for the user.

4. Launch it! Choose a web host and upload your website. Your web host may have an FTP feature, or you can download your own FTP program like FileZilla or CyberDuck. If you hired a professional to design the website, they should be able to take care of this for you (but it still pays to ask questions so that you understand what is happening).

Types of Websites

A Web site is a related collection of World Wide Web (WWW) files that includes a beginning file called a home page. A company or an individual tells you how to get to their Web site by giving you the address of their home page. From the home page, you can get to all the other pages on their site. For example, the Web site for IBM has the home page address of http://www.ibm.com. (The home page address actually includes a specific file name like index.html but, as in IBM's case, when a standard default name is set up, users do not have to enter the file name.) IBM's home page address leads to thousands of pages. (But a Web site can also be just a few pages.)

Since site implies a geographic place, a Web site can be confused with a Web server. A server is a computer that holds the files for one or more sites. A very large Web site may be spread over a number of servers in different geographic locations. IBM is a good example; its Web site consists of thousands of files spread out over many servers in world-wide locations. But a more typical example is probably the site you are looking at, whatis.com. We reside on a commercial space provider's server with a number of other sites that have nothing to do with Internet glossaries.

A synonym and less frequently used term for Web site is "Web presence." That term seems to better express the idea that a site is not tied to specific geographic location, but is "somewhere in cyberspace." However, "Web site" seems to be used much more frequently.

There are many static websites on the Internet, you may not be able to tell immediately if it is a static or dynamic website, but the chances are, if the site looks basic and is for a smaller company, and simply delivers information without any bells and whistles, it could be a static website.

Static Websites

There are many static websites on the Internet, you may not be able to tell immediately if it is a static or dynamic website, but the chances are, if the site looks basic and is for a smaller company, and simply delivers information without any bells and whistles, it could be a static website.

Advantages of static websites

- Quick to develop
- Cheap to develop
- Cheap to host

Disadvantages of static websites

- Requires web development expertise to update site
- Site not as useful for the user
- Content can get stagnant

Dynamic Websites

Dynamic sites on the other hand can be more expensive to develop initially, but the advantages are numerous. At a basic level, a dynamic website can give the website owner the ability to simply update and add new content to the site. For example, news and events could be posted to the site through a simple browser interface. Dynamic features of a site are only limited by imagination. Some examples of dynamic website features could be: content management system, e-commerce system, bulletin / discussion boards, intranet or extranet facilities, ability for clients or users to upload documents, ability for administrators or users to create content or add information to a site (dynamic publishing).

Advantages of dynamic websites

- Much more functional website
- Much easier to update
- New content brings people back to the site and helps in the search engines
- Can work as a system to allow staff or users to collaborate

Disadvantages of dynamic websites

- Slower / more expensive to develop
- Hosting costs a little more

Web Standards

Web standards are rules and guidelines established by the World Wide Web Consortium (W3C) developed to promote consistency in the design code which makes up a web page. Without getting technical, simply it is the guideline for the mark-up language which determines how a web page displays in a visitor's browser window.

The advantages in adhering to these standards are many:

- Web pages will display in a wide variety of browsers and computers, including new technology like iPhones, Droids, iPads, PDA devices, mobile phones, which greatly increases the viewing audience.

- W3C Standards promote the use of "Cascading Style Sheets" (CSS) or design code which is attached to the web page rather than embedded in the page. The use of style sheets significantly reduces the page file size which means not only a faster page loading time but lower hosting costs for frequently visited sites due to using less bandwidth.
- Design features such as colors and fonts can be easily changed by just modifying one style sheet instead of editing every individual page in a web site, reducing the costs to modify your site.
- Search Engines are able to access and index pages designed to web standards with greater efficiency.

W3C Standard

W3C standards define an Open Web Platform for application development that has the unprecedented potential to enable developers to build rich interactive experiences, powered by vast data stores that are available on any device. Although the boundaries of the platform continue to evolve, industry leaders speak nearly in unison about how HTML5 will be the cornerstone for this platform. But the full strength of the platform relies on many more technologies that W3C and its partners are creating, including CSS, SVG, WOFF, the Semantic Web stack, XML, and a variety of APIs.

W3C develops these technical specifications and guidelines through a process designed to maximize consensus about the content of a technical report, to ensure high technical and editorial quality, and to earn endorsement by W3C and the broader community.

If you are learning about Web technology, you may wish to start with the introduction below, and follow links for greater detail.

Web Design and Applications Header link

Web Design and Applications involve the standards for building and Rendering Web pages, including HTML, CSS, SVG, Ajax, and other technologies for Web Applications ("WebApps"). This

section also includes information on how to make pages accessible to people with disabilities (WCAG), to internationalize them, and make them work on mobile devices.

Web of Devices Header link

W3C is focusing on technologies to enable Web access anywhere, anytime, using any device. This includes Web access from mobile phones and other mobile devices as well as use of Web technology in consumer electronics, printers, interactive television, and even automobiles.

Web Architecture Header link

Web Architecture focuses on the foundation technologies and principles which sustain the Web, including URIs and HTTP.

Semantic Web Header link

In addition to the classic "Web of documents" W3C is helping to build a technology stack to support a "Web of data," the sort of data you find in databases. The ultimate goal of the Web of data is to enable computers to do more useful work and to develop systems that can support trusted interactions over the network. The term "Semantic Web" refers to W3C's vision of the Web of linked data. Semantic Web technologies enable people to create data stores on the Web, build vocabularies, and write rules for handling data. Linked data are empowered by technologies such as RDF, SPARQL, OWL, and SKOS.

XML Technology Header link

XML Technologies including XML, XML Namespaces, XML Schema, XSLT, Efficient XML Interchange (EXI), and other related standards.

Web of Services Header link

Web of Services refers to message-based design frequently found on the Web and in enterprise software. The Web of Services is based on technologies such as HTTP, XML, SOAP, WSDL, SPARQL, and others.

Browsers and Authoring Tools Header link

The web's usefulness and growth depends on its universality. We should be able to publish regardless of the software we use, the computer we have, the language we speak, whether we are wired or wireless, regardless of our sensory or interaction modes. We should be able to access the web from any kind of hardware that can connect to the Internet stationary or mobile, small or large. W3C facilitates this listening and blending via international web standards. These standards ensure that all the crazy brilliance continues to improve a web that is open to us all.

CASE STUDY

CROSS-PLATFORM WEB APPLICATION CAPABILITY

Recent statistics from the ITU show that there are almost six billion mobile-cellular subscriptions worldwide. An ever-increasing amount of new subscriptions concern smartphones, being able to run applications, or ”apps”. Add to this the advent of tablets and the fact that there are more than 1 billion mobile-broadband subscriptions worldwide, the mobile landscape is rapidly changing, with apps being one of the novelties paving the way. For companies looking to save time and money, mobile devices with the right applications installed have become excessively valuable.

HTML5 is the much promised newest version of the Hyper Text Markup Language and while not yet officially launched, is already starting to be implemented by many browsers. HTML5 provides new features involving restructuring of documents, media capabilities, a new canvas element for rendering graphics, as well as new form elements to better standardize forms across browsers. Besides the official features of the W3C standard for HTML5, there are a number of related APIs that are often considered to be part of the official HTML5 standard. These include such features as Drag and Drop, more detailed media interaction, and the Geolocation API along with many others in development by various groups. The word ”app” has become a common one in todays language and it generally refers to an application to be used on a mobile device such as a mobile phone or tablet. Most apps that people talk about are native applications, downloaded and installed from an official local such as Google Play or Apple’s App Store. A native application typically has native compiled code and is usually faster than interpreted languages like JavaScript (that is commonly used in web applications). Native applications run on one operating system and require being rewritten in a different language when they are run on multiple platforms. The main alternative to a native app though is the web app, requiring no installation or approval from an outside organization. The web app is created using typical web programming skills and abilities and runs through the user’s browser. Nothing needs to be installed, so a web application is able to theoretically run on any platform or device with a browser and internet connection. The world of the web app though is not itself perfect, there are still numerous compatibility issues due to a lack of standardization. A third route of hybrid applications, part native and part web, have been growing in popularity recently. It uses a web view in order to run JavaScript but can also access the mobile’s features as though it were a native app.

We compare the strengths and drawbacks of native applications versus web apps with emphasis on cross-platform functionality. We conduct a case study and develop a prototype based on novel web technology (e.g. the emerging HTML5

standard and related frameworks) with the goal of recreating the functionality and graphical properties of an existing native application. The purpose is to compare the differences of the main types of mobile applications to the results arrived at in this case study. Our beliefs are that the results from this particular case can inform design of cross-platform applications in general.

Case: Home Care Application

The study was conducted in a real world setting, our case being a system developed by a leading Swedish provider of ICT services in the health and welfare sector. The end-user application is an Android app supporting mobile personal working with health and care services. The application is crucial for workers as they can quickly access information while out in the field.

System Structure

In this situation a company has created a native android application that they then sell as part of a package solution to various other companies and municipalities working in the Swedish sector of home care. This sector involves field workers going out to people's residences and providing support for the elderly and disabled (henceforth known as 'the clients') so that they can continue to live on their home instead of being put into care homes. The workers provide cleaning, food, social contact, connections with doctors, personal hygiene, and even help getting up in the morning. In order to accomplish and document their daily tasks, the workers have been supplied with mobile android phones by their employers along with an already existing native android application.

This application connects with a number of other systems as seen in figure. The mobile device connects with a web service that then connects to a number of systems and databases in the solution that handle scheduling, documents, notes, and information about clients and the daily schedules for the field workers. ProCapita is a work system with built-in quality control while LapsCare adds back-end planning functionality. The InMovit then meld these systems, creating mobile back-end interfaces for web services and applications. The native application has functions for unlocking doors using Bluetooth, GPS location, writing and reading documentation, daily schedules for the worker whose phone it is, and a large amount of information about the client. The information can be doctor numbers, not to pet the dog (it bites), shower schedule, close contacts, hair appointments, and anything else a worker may need to know. As there are often multiple workers or even substitute workers unfamiliar with the particular client, it is vitally important that information be up to date and accurate.

Implementing the Web Application

The previously mentioned capabilities are currently present as part of a native Android application that is downloaded and installed on the mobile phones as needed. This system has drawbacks though in that it can only be run on mobiles running the android operating system and cannot be used on any of Apple's products such as the iPad or iPhone nor on Windows mobiles, etc. A solution to this was proposed with creating a web-based application instead of the native application.

Our web application is designed to copy the look and feel as well as functionality of the original Android specific app – i.e. the app should display log-in functionality, client browsing, individual scheduling and ability to access and read documentation about the clients. The GUI should be as close to the native app as possible. Also, our web app should support the opening of a native application through the web app, as there are other related systems (e.g. systems for opening and locking doors) that is still not available other than in native form. The application is written using HTML5, JSON (to create simple machine readable messages), and jQuery (to help and improve interaction with HTML elements and the server). There is also code designed to display information after it is received by the browser, e.g. navigation and GUI adaptation. Included are also a number of helper functions, e.g. handling date object conversion.

Evaluation and Results

The application was evaluated using simplified cognitive walkthrough, testing the application's functionality through pre-defined tasks such as logging in, choosing clients, editing schedules etc. Two iterative test/design rounds where carried out during the study. To capture the cross-platform properties of the applications, nine heterogeneous testbeds where used, combining different devices (e.g. Amazon Kindle, Apple iPad, HTC, LG and Samsung smartphones), different OSs (namely different versions of Android and iOs) and different browsers (e.g. Silk, Safari, Dophin Mini, Firefox mobile). Concerning functionality, recreation of the log-in routine was simple, but as the native system demands unicode support for encryption, some browsers did not offer support. In terms of client browsing, individual scheduling and ability to access and read documentation about the clients, this functionality was fully copied and implemented, as was support for opening of a third party native app through the web app.

Working from screen shots taken in the native Android application we could produce near pixel accurate reproductions of the proportions of various parts of the screen. Colors where also very similar to those displayed in the native version of the app. Some fonts differ between the versions, and this is due to the company's explicit

desire. Neglecting this, we could easily have recreated the look of the original fonts. Another dissimilarity between the look of the different apps is the background image, replaced in our web app version to reduce the data traffic involved in downloading a picture instead of going for a solid colored background. The web application allows any mobile device with internet and a browser that supports Unicode font encoding and HTML5's local Storage ability, to run the device. The type of operating system does not matter.

There is however still challenges to overcome when developing cross-platform applications. Browsers that could not support local Storage or other important HTML5 features did not deliver 100% functionality and/or look and feel. This problem is gradually decreasing though as more browsers add in HTML5 features, phones update their browsers, and mobiles become more capable of handling complicated websites. Our tests also showed that though functionality sometimes was not completely recreated, the web app could be opened and used on almost any platform. By not writing separate native applications a company avoids having to make potentially expensive hardware upgrades as well as requiring only one set of programmers instead of many programmers with speciality skills for each and every platform they would like to have an application work with. An advantage of having only one application, a web-based application, is that a company only needs to keep one application version updated. It is our belief that emerging web standards like HTML5 and related frameworks can play an important roll in creating mobile systems and applications with cross-platform functionality. Future work will include user testing in a real use case, comparisons with other similar systems and projects and information of application design.

Questions

1. Compare the strengths and drawbacks of native applications versus web apps with emphasis on cross-platform functionality.
2. How to implement the web application.

KEY TERMS

Web technology
Domain Name Server
Content Management System (CMS)
Scripting language
Web development
Storyboard
Common Gateway Interface (CGI)
Skills
Common Gateway Interface (CGI)
Technical skills

DISCUSSION QUESTIONS

1. What are the job roles of web developer? Explain.
2. Focus on website works.
3. What do you understand by the domains and hosting?
4. Differentiate between static and dynamic website.
5. Explain importance of web and W3C standard.
6. How to creation websites?
7. Focus on web design and applications header link.
8. Define the web architecture header link.

REFERENCES

1. Černáková, Business Information Systems, Nitra: Forpress, 2014.
2. E. Jurišová, English for IT Students, Nitra: Forpress, 2014.
3. E. Marcotte, "Responsive Web Design," 25/05/2010. Online.. Available: http://alistapart.com/article/responsive-web-design. Accessed 10/11/2013..
4. Eyetools, "The Importance of the Golden Triangle," Online.. Available: http://www.mediative.com/research/golden-triangle. Accessed 21/04/2014..
5. F. Schindler, Data Structures and Algorithms, Praha: Wolters Kluwer, 2014.
6. F. Schindler, Essentials of Programming, Nitra: Forpress, 2014.
7. G. Roelofs, "Portable Network Graphics," Online.. Available: http://www.libpng.org/pub/png/. Accessed 03/05/2013..
8. GNU Operating system, "GNU GENERAL PUBLIC LICENSE," Online.. Available: http://www.gnu.org/copyleft/gpl.html. Accessed 24/10/2013..
9. J. Fogel, Operating Systems, Nitra: Forpress, 2014.
10. J. J. Garret, "Ajax: A New Approach to Web Applications," 18/02/2005. Online.. Available: https://web.archive.org/web/20080702075113/http://www.adaptivepath.com/ideas/essays/archives/000385.php. Accessed 21/08/2013..
11. J. Kultán, Introduction into Databases, Nitra: Forpress, 2014.
12. J. Lacko and E. Ružický, Web Technologies and Design, Nitra: Forpress, 2014.
13. J. Nielsen, "F-Shaped Pattern For Reading Web Content," 17/04/2006. Online.. Available: http://www.nngroup.com/articles/f-shaped-pattern-reading-web-content/. Accessed 21/04/2014..
14. J. Voříšek, Principles of Development and Operation of Business Information Systems, Praha: Wolters Kluwer, 2014.
15. M. Šperka and I. Lvovich, Introduction to Information Technology, Nitra: Forpress, 2014.
16. M. Šperka, Graphical User Interface in Java, Praha: Wolters Kluwer, 2014.
17. P. Farkaš, Transcontrol Codes, Praha: Wolters Kluwer, 2014.
18. PHP.net, "PHP.net," Online.. Available: http://www.php.net/. Accessed 14/09/2013..
19. T. Páleník, Computer Networks, Nitra: Forpress, 2014.
20. V. Palko, Discrete Mathematics, Nitra: Forpress, 2014.
21. V. Řepa, Process Management and Modelling, Nitra: Forpress, 2014.

CHAPTER 2

HTML



LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Give the overview of HTML
2. Discuss the basic HTML document
3. Focus on creating an HTML document
4. Basic tags of HTML
5. Elements of HTML
6. Attributes of HTML
7. HTML formatting
8. Phrase tags in HTML
9. Images in HTML
10. Lists in HTML
11. HTML frames

INTRODUCTION

HTML (Hypertext Markup Language) is the set of markup symbols or codes inserted in a file intended for display on a World Wide Web browser page. The markup tells the Web browser how to display a Web page's words and images for the user. Each individual markup code is referred to as an element (but many people also refer to it as a tag). Some elements come in pairs that indicate when some display effect is to begin and when it is to end.

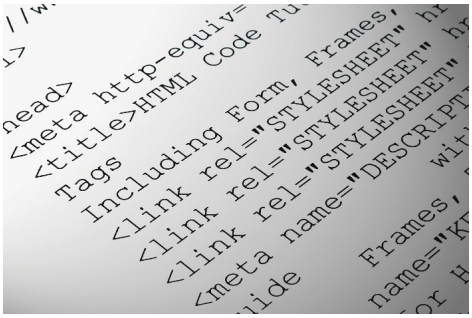


HTML is a formal Recommendation by the World Wide Web Consortium (W3C) and is generally adhered to by the major browsers, Microsoft's Internet Explorer and Netscape's Navigator, which also provide some additional non-standard codes. The current version of HTML is HTML 4.0. However, both Internet Explorer and Netscape implement some features differently and provide non-standard extensions. Web developers using the more advanced features of HTML 4 may have to design pages for both browsers and send out the appropriate version to a user. Significant features in HTML 4 are sometimes described in general as dynamic HTML. What is sometimes referred to as HTML 5 is an extensible form of HTML called Extensible Hypertext Markup Language (XHTML).

OVERVIEW OF HTML

HyperText Markup Language (HTML) is a type of computer language that is used to create pages that can be posted on the Internet or sent via email. Although it might seem complex to many people, it is considered to be a relatively simple language.

All text, graphics, and design elements a page designed with this language are "tagged" with codes that instruct the web browser or email program how to display those elements. The tags also provide layout and formatting information so that the web page or email will look as close as possible to the way its designer intended it to look. For the novice website designer or anyone else who needs to know a certain code or wants to learn how to create an entire website, there are many software utilities, programs and websites that can assist him or her in writing HTML code.



HTML (**H**yper**t**ext **M**arkup **L**anguage) is the most widely used language to write Web Pages.

- **Hypertext** refers to the way in which Web pages (HTML documents) are linked together. Thus, the link available on a webpage is called Hypertext.
- As its name suggests, HTML is a **Markup Language** which means you use HTML to simply "mark-up" a text document with tags that tell a Web browser how to structure it to display.

Originally, HTML was developed with the intent of defining the structure of documents like headings, paragraphs, lists, and so forth to facilitate the sharing of scientific information between researchers.

Now, HTML is being widely used to format web pages with the help of different tags available in HTML language.

Structure

HTML coding is structured like a tree, with each different tag nested within it. In most cases, each formatting element requires

a start tag and an end tag, and different tags should not overlap. This is what is meant by "nested;" if tag 2 opens after tag 1, then tag 2 should be closed first so that the formatting element of tag 2 is entirely enclosed within tag 1. Elements are the individual components that make up the code, and include opening and closing tags and the content between them. Attributes provide more information about the element, and are made up of the attribute and its value, connected by an equal sign.

To create an HTML element, the user creates a tag that starts and finishes with angle brackets and places it before the text that needs to be formatted. The code — usually one or more letters, numbers, words, and/or symbols — inside the angle brackets indicates what the element is and the attributes that content should have, such as its size, font, or other characteristics. To end the formatting, the user types the first angle bracket, then a backslash, then repeats the element code and closes the bracket. *For example*, <title>What Is HTML?</title> is the code used to format the title of this article; the "strong" element tag is nested within the "title" tag.

Types of Codes

There are many codes to allow for different text formatting, including italics, tables, paragraphs, and hyperlinks to web pages. Codes also can indicate to the browser or email program how to display or use other elements, such as pictures, graphics, video, and sound. Other types of codes without angle brackets can be used to create punctuation marks, diacritics, and other symbols that might appear in text. Although all web browsers and many email programs use HTML, each might interpret and display the code a little differently, and designers often must consider these variations when creating a web page.

An Evolving Language

Since the development of HTML in the early 1990s by British computer scientist Tim Berners-Lee, there have been many changes and versions. These versions have been maintained by the World Wide Web Consortium (W3C) since 1996. In January 2008, the First Public Working Draft of HTML 5 was published by the working group that was developing this specification.

Still under development as of 2011, this revision was expected to dramatically change application development for the web. It introduces a number of new elements, including those for site structure, interactivity, and audio and video support, as well as new attributes.

Viewing a Web Page's Code

The code used to create any web page can be seen by navigating a browser to the page, then choosing the correct option from the browser's menu. In most browsers, the user can click on the "view" menu and select an option such as "source," "view source" or "page source." This will cause a pop-up window to appear, and it will show the code that was used to create that web page.

It is important to note that not all of the content found on all web pages is written in HTML. Extensible Markup Language (XML) and Extensible HyperText Markup Language (XHTML) are other types of markup languages used in web development. In addition, style sheets — like Cascading Style Sheets (CSS) — are used to attach style to HTML documents. Languages like Flash® and Java® are used to create interactive content. Many other programming languages can be used to add specific functionality to a website.

As Compared to HTM

Practically speaking, there's little difference between HTM and HTML extensions, since both are read as an HTML file by most machines.

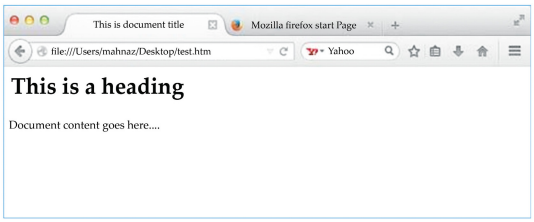
The reason there were two different extensions to begin with is that certain types of computers, like those that ran on 16 bit DOS or Windows® 3 systems, could not read four character extensions, and so needed the three character HTM extension. Most systems that can read four character extensions are automatically programmed to recognize HTM files as HTML files, though computer users may occasionally need a converter to change a file from HTM into a format that the system recognizes.

BASIC HTML DOCUMENT

In its simplest form, following is an example of an HTML document –

```
<!DOCTYPE html>
<html>
<head>
<title>This is document title</title>
</head>
<body>
<h1>This is a heading</h1>
<p>Document content goes here.....</p>
</body>
</html>
```

Either you can use **Try it** option available at the top right corner of the code box to check the result of this HTML code, or let's save it in an HTML file **test.htm** using your favorite text editor. Finally open it using a web browser like Internet Explorer or Google Chrome, or Firefox etc. It must show the following output –



HTML Tags

HTML is short for Hypertext Markup Language, the text-based language used to write Web pages. HTML tags are coding instructions embedded in the HTML document. A Web browser is designed to read the instructions, or tags, in order to render the page graphically. In other words, a Web browser translates these tags into visual effects that mold the way an HTML document looks to the viewer.

The simplest HTML tags arrange text into blocks, designate the font or letter-style, and size of the type. Bold, italic and underline are each identified within an tag that precedes the letter or word(s) to be effected. The tag appears again at the end of the chosen text with a slash to indicate the effect ends there. For example, see how the following sentence appears in HTML:

HTML tags are the core of Hypertext Markup Language. HTML tags are the core of Hypertext Markup Language.

The "b" stands for **bold** and HTML tags are always enclosed in angled brackets, with the closing tag starting with a forward slash. If the forward slash is forgotten, the remainder of the page will assume the effect.

In addition to changing fonts, tags also create hyperlinks, or clickable text. The hyperlink tag includes an embedded website address (URL or Uniform Resource Locator). By placing a hyperlink tag around a phrase or name, clicking on it will take the surfer to the desired address. This can be a remote website or another page within the same website.

HTML tags can also be used to take a surfer to another spot within the same page. This is handy for Frequently Asked Questions (FAQs) or indexes that reside on a single page. By clicking on a question or an indexed subject, the visitor can jet to the exact information he or she needs without paging down through the entire document.

Other tags embed graphics, movies, sound effects, animation or Flash scripts in webpages. Frames, borders, background and page layouts are also designated by HTML.

On websites with multiple pages there are often characteristics the webmaster wants to repeat on each page. These might include the font type, website colors, background and text blocking or layout. Rather than repeat these tags on every page, HTML allows for a bit of code at the top of each page that points the browser to a master style sheet which contains the HTML tags that apply. A master style sheet is known as *Cascading Style Sheet (CSS)* because its effects can "cascade" across several pages.

In the middle 1990s when the Internet switched from a text-based environment to a graphical environment, thanks to HTML and Web browsers that interpreted the language, the only way to create a website was to manually code the pages using HTML tags and a text-based editor. Shortly HTML editors became available that did a lot of the coding for the user, making it easy for anyone to create a basic website. Today, ready-made scripts and forms can be embedded with a click, allowing anyone with basic needs to be their own webmaster.

As told earlier, HTML is a markup language and makes use of various tags to format the content. These tags are enclosed within angle braces **<Tag Name>**. Except few tags, most of the tags have their corresponding closing tags. *For example*, **<html>** has its closing tag **</html>** and **<body>** tag has its closing tag **</body>** tag etc.

Above example of HTML document uses the following tags –

Sr. No	Tag	Description
1	**<!DOCTYPE...>**	This tag defines the document type and HTML version.
2	**<html>**	This tag encloses the complete HTML document and mainly comprises of document header which is represented by <head>...</head> and document body which is represented by <body>...</body> tags.
3	**<head>**	This tag represents the document's header which can keep other HTML tags like <title>, <link> etc.
4	**<title>**	The <title> tag is used inside the <head> tag to mention the document title.
5	**<body>**	This tag represents the document's body which keeps other HTML tags like <h1>, , <p> etc.
6	**<h1>**	This tag represents the heading.
7	**<p>**	This tag represents a paragraph.

World Wide Web Consortium (W3C) recommends to use lowercase tags starting from HTML 4.

HTML Document Structure

A typical HTML document will have the following structure –

```
Document declaration tag
<html>
   <head>
       Document header related tags
   </head>

   <body>
       Document body related tags
   </body>
</html>
```

The <!DOCTYPE> Declaration

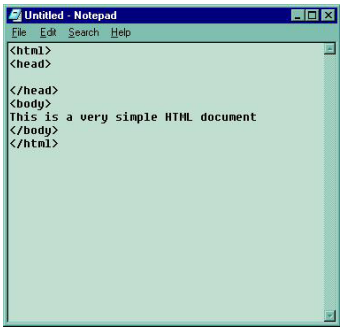
The <!DOCTYPE> declaration tag is used by the web browser to understand the version of the HTML used in the document. Current version of HTML is 5 and it makes use of the following declaration –

```
<!DOCTYPE html>
```

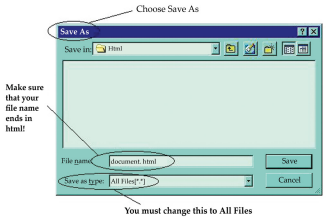
There are many other declaration types which can be used in HTML document depending on what version of HTML is being used. We will see more details on this while discussing <!DOCTYPE...> tag along with other HTML tags.

CREATING AN HTML DOCUMENT

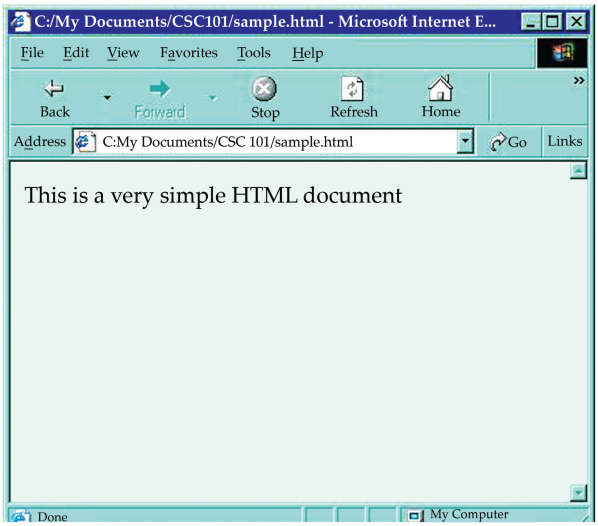
To create a new HTML document, open a new document in Notepad (Notepad should be located under programs/accessories). You will type all your HTML code into this document. To start, type the following HTML code into Notepad.



When you are done typing save your document. When you save an HTML document in Notepad, you must specify that it should be saved as an HTML document. To do this, choose Save As from the file menu. Type in the file name followed by .html. You must also change the Save as Type to All Files or Notepad will automatically save your document as a text document.



Next open an Internet browser (ex. Netscape or Internet Explorer). Choose Open or Open Page from the file menu. Locate and select your HTML document. If you cannot find your HTML document, go back to the previous step and make sure that you saved it properly. Your HTML document should now be displayed in the browser window.



You have now created your first webpage. If you want to make changes to your HTML code, you must save your HTML document in Notepad and then click on Reload or Refresh in the browser window in order to view your changes.

BASIC TAGS OF HTML

The basic tags include <html>, <title>, <meta>, and <body>. We introduce each of the four in the following:

<html>

This tag is used to indicate that this is a HTML document. Most HTML documents should start and end with this tag.

<head>

This tag is used to indicate the header section of the HTML document, which typically includes the <title> and <meta> tags,

and is not displayed in the main window of the browser.

<title>

This indicates the title of this HTML page. The title is what is displayed on the upper left corner of your browser when you view a web page. *For example*, right now you can see "Basic Tags: html, head, title, meta, body" there. That is the title of this page.

The title tag is important when it comes to search engine ranking. Many of the search engines pay special attention to the text in the <title> tag. This is because (logically) that words in the <title> tag indicate what the page content is.

<meta>

The <meta> tag information is not directly displayed when the page is rendered on the browser. Rather, this is used for the author of the HTML page to record information related to this page. Two common attributes are **name** and **content**. The <meta> tag used to hold great importance in search engine optimization, with authors carefully drafting what's inside the tag to gain better search engine ranking, but recently its importance has been decreasing steadily.

<body>

The <body> tag includes the HTML body of the document. Everything inside the <body> tag (other than those within the <script> tag) is displayed on the browser inside the main browser window.

The <body> tag may contain several attributes. The most commonly used ones are listed below:

- **bgcolor**: This is the background color of the entire HTML document, and may be specified either by the color name directly or by the six-digit hex code.
- **alink**: The color of the links.
- **vlink**: The color of the visited links.
- **topmargin**: The margin from the top of the browser window.

- **leftmargin**: The margin from the left of the browser window.

So, in general, all HTML documents have the following format:

<html>

<head>

<title>

Here is the title of the HTML document.

</title>

<meta name=" " content=" " />

... (there may be one or more meta tags)

</meta>

</head>

<body>

Here is the body of the HTML document.

</body>

</html>

Once you have the essential tags sorted out, there are a number of basic tags you will use a lot in order to add content such as:

- Headings and Subheadings
- Paragraphs of Text
- Links to Other Pages
- Images or Photographs

Heading Tags

Any document starts with a heading. You can use different sizes for your headings. HTML also has six levels of headings, which use the elements **<h1>, <h2>, <h3>, <h4>, <h5>,** and **<h6>**. While displaying any heading, browser adds one line before and one line after that heading.

Example

```
<!DOCTYPE html>
    <html>
       <head>
          <title>Heading Example</title>
       </head>

       <body>
          <h1>This is heading 1</h1>
          <h2>This is heading 2</h2>
          <h3>This is heading 3</h3>
          <h4>This is heading 4</h4>
          <h5>This is heading 5</h5>
          <h6>This is heading 6</h6>
       </body>

    </html>
```

This will produce the following result –

This is heading 1

This is heading 2

This is heading 3

This is heading 4

This is heading 5

This is heading 6

Paragraph Tag

The ** ** tag offers a way to structure your text into different paragraphs. Each paragraph of text should go in between an opening and a closing tag as shown below in the example –

Example

```
<!DOCTYPE html>
    <html>

      <head>
        <title>Paragraph Example</title>
      </head>

      <body>
        <p>Here is a first paragraph of text.</p>
        <p>Here is a second paragraph of text.</p>
        <p>Here is a third paragraph of text.</p>
      </body>

    </html>
```

This will produce the following result –

Here is a first paragraph of text.

Here is a second paragraph of text.

Here is a third paragraph of text.

Line Break Tag

Whenever you use the **
** element, anything following it starts from the next line. This tag is an example of an **empty** element, where you do not need opening and closing tags, as there is nothing to go in between them.

The <br /> tag has a space between the characters **br** and the forward slash. If you omit this space, older browsers will have trouble rendering the line break, while if you miss the forward slash character and just use <br> it is not valid in XHTML.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Line Break  Example</title>
   </head>

   <body>
      <p>Hello<br />
         You delivered your assignment ontime.<br />
         Thanks<br />
         Mahnaz</p>
   </body>

</html>
```

This will produce the following result –

Hello

You delivered your assignment on time.

Thanks

Mahnaz

Centering Content

You can use ** ** tag to put any content in the center of the page or any table cell.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Centring Content Example</title>
   </head>

   <body>
      <p>This text is not in the center.</p>

      <center>
         <p>This text is in the center.</p>
      </center>
   </body>

</html>
```

This will produce following result –

This text is not in the center.

This text is in the center.

Horizontal Lines

Horizontal lines are used to visually break-up sections of a document. The **<hr>** tag creates a line from the current position in the document to the right margin and breaks the line accordingly.

For example, you may want to give a line between two paragraphs as in the given example below –

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Horizontal Line Example</title>
  </head>

  <body>
    <p>This is paragraph one and should be on top</p>
    <hr />
   <p>This is paragraph two and should be at bottom</p>
  </body>

</html>
```

This will produce the following result –

This is paragraph one and should be on top

This is paragraph two and should be at bottom

Again **<hr />** tag is an example of the **empty** element, where you do not need opening and closing tags, as there is nothing to go in between them.

The **<hr />** element has a space between the characters **hr** and the forward slash. If you omit this space, older browsers will have trouble rendering the horizontal line, while if you miss the forward slash character and just use **<hr>** it is not valid in XHTML

Preserve Formatting

Sometimes, you want your text to follow the exact format of how it is written in the HTML document. In these cases, you can use the preformatted tag **<pre>**.

Any text between the opening **<pre>** tag and the closing **</pre>** tag will preserve the formatting of the source document.

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Preserve Formatting Example</title>
  </head>

  <body>
    <pre>
      function testFunction( strText ){
        alert (strText)
      }
    </pre>
  </body>

</html>
```

This will produce the following result –

Try using the same code without keeping it inside **<pre>...</pre>** tags

Nonbreaking Spaces

Suppose you want to use the phrase "12 Angry Men." Here, you would not want a browser to split the "12, Angry" and "Men" across two lines –

An example of this technique appears in the movie "12 Angry Men."

In cases, where you do not want the client browser to break text, you should use a nonbreaking space entity ** ** instead of a normal space. For example, when coding the "12 Angry Men" in a paragraph, you should use something similar to the following code –

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Nonbreaking Spaces Example</title>
   </head>

   <body>
      <p>An example of this technique appears in the movie
"12 Angry Men."</p>
   </body>

</html>
```

ELEMENTS OF HTML

An **HTML element** is defined by a starting tag. If the element contains other content, it ends with a closing tag, where the element name is preceded by a forward slash as shown below with few tags −

Start Tag	Content	End Tag
	This is paragraph content.	
<h1>	This is heading content.	</h1>
	This is division content.	

So here ** .... ** is an **HTML element**, **<h1>...</h1>** is another HTML element. There are some HTML elements which don't need to be closed, such as **<img.../>**, **<hr />** and **
** elements. These are known as **void elements**.

HTML documents consists of a tree of these elements and they specify how HTML documents should be built, and what kind of content should be placed in what part of an HTML document.

HTML Tag vs. Element

An HTML element is defined by a *starting tag*. If the element contains other content, it ends with a *closing tag*.

For example, ** ** is starting tag of a paragraph and ** ** is closing tag of the same paragraph but ** This is paragraph ** is a paragraph element.

Nested HTML Elements

It is very much allowed to keep one HTML element inside another HTML element −

Example

```
<!DOCTYPE html>
<html>
```

```
<head>
   <title>Nested Elements Example</title>
</head>

<body>
   <h1>This is <i>italic</i> heading</h1>
   <p>This is <u>underlined</u> paragraph</p>
</body>

</html>
```

This will display the following result −



ATTRIBUTES OF HTML

We have seen few HTML tags and their usage like heading tags **<h1>, <h2>,** paragraph tag **<p>** and other tags. We used them so far in their simplest form, but most of the HTML tags can also have attributes, which are extra bits of information.

An attribute is used to define the characteristics of an HTML element and is placed inside the element's opening tag. All attributes are made up of two parts − a **name** and a **value**

- The **name** is the property you want to set. *For example*, the paragraph **<p>** element in the example carries an attribute whose name is **align**, which you can use to indicate the alignment of paragraph on the page.
- The **value** is what you want the value of the property to be set and always put within quotations. The below example shows three possible values of align attribute: **left, center** and **right**.

Attribute names and attribute values are case-insensitive. However, the World Wide Web Consortium (W3C) recommends

lowercase attributes/attribute values in their HTML 4 recommendation.

Example

```
<!DOCTYPE html>
    <html>

      <head>
        <title>Align Attribute  Example</title>
      </head>

      <body>
        <p align = “left”>This is left aligned</p>
        <p align = “center”>This is center aligned</p>
        <p align = “right”>This is right aligned</p>
      </body>

    </html>
```

This will display the following result –

This is left aligned

This is center aligned

This is right aligned

Core Attributes

The four core attributes that can be used on the majority of HTML elements (although not all) are –

- Id
- Title

- Class
- Style

The Id Attribute

The **id** attribute of an HTML tag can be used to uniquely identify any element within an HTML page. There are two primary reasons that you might want to use an id attribute on an element –

- If an element carries an id attribute as a unique identifier, it is possible to identify just that element and its content.
- If you have two elements of the same name within a Web page (or style sheet), you can use the id attribute to distinguish between elements that have the same name.

We will discuss style sheet in separate tutorial. For now, let's use the id attribute to distinguish between two paragraph elements as shown below.

Example

```
<p id = "html">This para explains what is HTML</p>

<p id = "css">This para explains what is Cascading Style Sheet</
p>
```

The title Attribute

The **title** attribute gives a suggested title for the element. They syntax for the **title** attribute is similar as explained for **id** attribute –

The behavior of this attribute will depend upon the element that carries it, although it is often displayed as a tooltip when cursor comes over the element or while the element is loading.

Example

```
<!DOCTYPE html>

<html>
```

```
<head>
   <title>The title Attribute Example</title>
</head>

<body>
 <h3 title = "Hello HTML!">Titled Heading Tag Example</
h3>
</body>

</html>
```

This will produce the following result –

Titled Heading Tag Example

Now try to bring your cursor over "Titled Heading Tag Example" and you will see that whatever title you used in your code is coming out as a tooltip of the cursor.

The class Attribute

The **class** attribute is used to associate an element with a style sheet, and specifies the class of element. You will learn more about the use of the class attribute when you will learn Cascading Style Sheet (CSS). So for now you can avoid it.

The value of the attribute may also be a space-separated list of class names. For example –

```
class = "className1 className2 className3"
```

The style Attribute

The style attribute allows you to specify Cascading Style Sheet (CSS) rules within the element.

```
<!DOCTYPE html>

<html>
```

```
<head>
   <title>The style Attribute</title>
</head>

<body>
   <p style = "font-family:arial; color:#FF0000;">Some text...</p>
</body>

</html>
```

This will produce the following result −

Some text...

At this point of time, we are not learning CSS, so just let's proceed without bothering much about CSS. Here, you need to understand what are **HTML attribute**s and how they can be used while formatting content.

Internationalization Attributes

There are three internationalization attributes, which are available for most (although not all) XHTML elements.

- dir
- lang
- xml:lang

The dir Attribute

The **dir** attribute allows you to indicate to the browser about the direction in which the text should flow. The dir attribute can take one of two values, as you can see in the table that follows −

Value	Meaning
ltr	Left to right (the default value)
rtl	Right to left (for languages such as Hebrew or Arabic that are read right to left)

Example

```
<!DOCTYPE html>
<html dir = "rtl">

   <head>
      <title>Display Directions</title>
   </head>

   <body>
      This is how IE 5 renders right-to-left directed text.
   </body>

</html>
```

This will produce the following result –

This is how IE 5 renders right-to-left directed text.

When *dir* attribute is used within the <html> tag, it determines how text will be presented within the entire document. When used within another tag, it controls the text's direction for just the content of that tag.

The lang Attribute

The **lang** attribute allows you to indicate the main language used in a document, but this attribute was kept in HTML only for backwards compatibility with earlier versions of HTML. This attribute has been replaced by the **xml:lang** attribute in new XHTML documents.

The values of the *lang* attribute are ISO-639 standard two-character language codes. Check **HTML Language Codes: ISO 639** for a complete list of language codes.

Example

```
<!DOCTYPE html>
<html lang = "en">

   <head>
      <title>English Language Page</title>
   </head>

   <body>
      This page is using English Language
   </body>

</html>
```

The xml:lang Attribute

The *xml:lang* attribute is the XHTML replacement for the *lang* attribute. The value of the *xml:lang* attribute should be an ISO-639 country code.

Generic Attributes

Here's a table of some other attributes that are readily usable with many of the HTML tags.

Attribute	Options	Function
align	right, left, center	Horizontally aligns tags
valign	top, middle, bottom	Vertically aligns tags within an HTML element.
bgcolor	numeric, hexidecimal, RGB values	Places a background color behind an element

background	URL	Places a background image behind an element
id	User Defined	Names an element for use with Cascading Style Sheets.
class	User Defined	Classifies an element for use with Cascading Style Sheets.
width	Numeric Value	Specifies the width of tables, images, or table cells.
height	Numeric Value	Specifies the height of tables, images, or table cells.
title	User Defined	"Pop-up" title of the elements.

HTML FORMATTING

HTML Formatting is *a process of formatting text for better look and feel*. There are many formatting tags in HTML. These tags are used to make text bold, italicized, or underlined. These are just three of the ten options available to indicate how text can appear in HTML and XHTML.

Bold Text

Anything that appears within **...** element, is displayed in bold as shown below –

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Bold Text Example</title>
  </head>

  <body>
   <p>The following word uses a <b>bold</b> typeface.</p>
```

```
</body>

</html>
```

This will produce the following result –

The following word uses a **bold** typeface.

Italic Text

Anything that appears within **...** element is displayed in italicized as shown below –

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Italic Text Example</title>
   </head>

   <body>
    <p>The following word uses an <i>italicized</i> typeface.</
p>
   </body>

</html>
```

This will produce the following result –

The following word uses an *italicized* typeface.

Underlined Text

Anything that appears within **<u>...</u>** element, is displayed with underline as shown below –

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Underlined Text Example</title>
   </head>

   <body>
      <p>The following word uses an <u>underlined</u> typeface.</p>
   </body>

</html>
```

This will produce the following result –

The following word uses an underlined typeface.

Strike Text

Anything that appears within **<strike>...</strike>** element is displayed with strikethrough, which is a thin line through the text as shown below –

Example

```
<!DOCTYPE html>
<html>
```

```
<head>
   <title>Strike Text Example</title>
</head>

<body>
   <p>The following word uses a <strike>strikethrough</
strike> typeface.</p>
</body>

</html>
```

This will produce the following result –

The following word uses a ~~strikethrough~~ typeface.

Monospaced Font

The content of a **<tt>...</tt>** element is written in monospaced font. Most of the fonts are known as variable-width fonts because different letters are of different widths (for example, the letter 'm' is wider than the letter 'i'). In a monospaced font, however, each letter has the same width.

Example

```
<!DOCTYPE html>
<html>

<head>
   <title>Monospaced Font Example</title>
</head>
```

```
<body>
   <p>The following word uses a <tt>monospaced</tt>
typeface.</p>
</body>

</html>
```

This will produce the following result –

The following word uses a `monospaced` typeface.

Superscript Text

The content of a **^{...}** element is written in superscript; the font size used is the same size as the characters surrounding it but is displayed half a character's height above the other characters.

Example

```
<!DOCTYPE html>
<html>

<head>
   <title>Superscript Text Example</title>
</head>

<body>
   <p>The following word uses a <sup>superscript</sup>
typeface.</p>
</body>

</html>
```

This will produce the following result –

The following word uses a superscript typeface.

Subscript Text

The content of a **_{...}** element is written in subscript; the font size used is the same as the characters surrounding it, but is displayed half a character's height beneath the other characters.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Subscript Text Example</title>
   </head>

   <body>
      <p>The following word uses a <sub>subscript</sub>
typeface.</p>
   </body>

</html>
```

This will produce the following result –

The following word uses a subscript typeface.

Inserted Text

Anything that appears within **<ins>...</ins>** element is displayed as inserted text.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Inserted Text Example</title>
   </head>

   <body>
     <p>I want to drink <del>cola</del> <ins>wine</ins></p>
   </body>

</html>
```

This will produce the following result –

I want to drink cola wine

Deleted Text

Anything that appears within **~~...~~** element, is displayed as deleted text.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Deleted Text Example</title>
   </head>
```

```
<body>
  <p>I want to drink <del>cola</del> <ins>wine</ins></p>
</body>

</html>
```

This will produce the following result –

I want to drink cola wine

Larger Text

The content of the **<big>...</big>** element is displayed one font size larger than the rest of the text surrounding it as shown below –

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Larger Text Example</title>
   </head>

   <body>
     <p>The following word uses a <big>big</big> typeface.</
p>
   </body>

</html>
```

This will produce the following result –

The following word uses a big typeface.

Smaller Text

The content of the **<small>...</small>** element is displayed one font size smaller than the rest of the text surrounding it as shown below –

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Smaller Text Example</title>
   </head>

   <body>
      <p>The following word uses a <small>small</small> typeface.</p>
   </body>

</html>
```

This will produce the following result –

The following word uses a small typeface.

Grouping Content

The ** ** and **** elements allow you to group together several elements to create sections or subsections of a page.

For example, you might want to put all of the footnotes on a page within a <div< element to indicate that all of the elements within that element relate to the footnotes. You might then attach a style to this element so that they appear using a special set of style rules.

Example

```
<!DOCTYPE html>
<html>

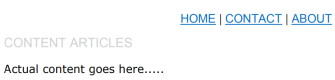
   <head>
      <title>Div Tag Example</title>
   </head>

   <body>
      <div id = "menu" align = "middle" >
         <a href = "/index.htm">HOME</a> |
         <a href = "/about/contact_us.htm">CONTACT</a> |
         <a href = "/about/index.htm">ABOUT</a>
      </div>

      <div id = "content" align = "left" bgcolor = "white">
         <h5>Content Articles</h5>
         <p>Actual content goes here.....</p>
      </div>
   </body>

</html>
```

This will produce the following result –



The element, on the other hand, can be used to group inline elements only. So, if you have a part of a sentence

or paragraph which you want to group together, you could use the element as follows.

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Span Tag Example</title>
  </head>

  <body>
   <p>This is the example of <span style = “color:green”>span tag</span>
        and the <span style = “color:red”>div tag</span> alongwith CSS</p>
  </body>

</html>
```

This will produce the following result –

This is the example of span tag and the div tag alongwith CSS

These tags are commonly used with CSS to allow you to attach a style to a section of a page.

PHRASE TAGS IN HTML

The phrase tags have been desicolgned for specific purposes, though they are displayed in a similar way as other basic tags like **<b>, <i>, <pre>**, and **<tt>**.

Emphasized Text

Anything that appears within **...** element is displayed as emphasized text.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Emphasized Text Example</title>
   </head>

   <body>
      <p>The following word uses an <em>emphasized</em>
typeface.</p>
   </body>

</html>
```

This will produce the following result –

The following word uses an *emphasized* typeface.

Marked Text

Anything that appears with-in **<mark>...</mark>** element, is displayed as marked with yellow ink.

Example

```
<!DOCTYPE html>
<html>
```

```
<head>
   <title>Marked Text Example</title>
</head>

<body>
   <p>The following word has been <mark>marked</mark>
with yellow</p>
</body>

</html>
```

This will produce the following result –

The following word has been marked with yellow

Strong Text

Anything that appears within **...** element is displayed as important text.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Strong Text Example</title>
   </head>

   <body>
      <p>The following word uses a <strong>strong</strong>
typeface.</p>
```

```
</body>

</html>
```

This will produce the following result –

The following word uses a **strong** typeface.

Text Abbreviation

You can abbreviate a text by putting it inside opening <abbr> and closing </abbr> tags. If present, the title attribute must contain this full description and nothing else.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Text Abbreviation</title>
   </head>

   <body>
         <p>My best friend's name is  <abbr title = "Abhishek">Abhy</abbr>.</p>
   </body>

</html>
```

This will produce the following result –

My best friend's name is Abhy.

Acronym Element

The **<acronym>** element allows you to indicate that the text between <acronym> and </acronym> tags is an acronym. At present, the major browsers do not change the appearance of the content of the <acronym> element.

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Acronym Example</title>
  </head>

  <body>
        <p>This chapter covers marking up text in
<acronym>XHTML</acronym>.</p>
  </body>

</html>
```

This will produce the following result –

This chapter covers marking up text in XHTML.

Text Direction

The **<bdo>...</bdo>** element stands for Bi-Directional Override and it is used to override the current text direction.

Example

```
<!DOCTYPE html>
<html>
```

```
<head>
   <title>Text Direction Example</title>
</head>

<body>
   <p>This text will go left to right.</p>
    <p><bdo dir = "rtl">This text will go right to left.</
bdo></p>
</body>

</html>
```

This will produce the following result –

This text will go left to right.

.tfel ot thgir og lliw txet sihT

Special Terms

The **<dfn>...</dfn>** element (or HTML Definition Element) allows you to specify that you are introducing a special term. It's usage is similar to italic words in the midst of a paragraph.

Typically, you would use the <dfn> element the first time you introduce a key term. Most recent browsers render the content of a <dfn> element in an italic font.

Example

```
<!DOCTYPE html>
<html>
   <head>
      <title>Special Terms Example</title>
   </head>
```

```
<body>
 <p>The following word is a <dfn>special</dfn> term.</p>
</body>

</html>
```

This will produce the following result –

The following word is a *special* term.

Quoting Text

When you want to quote a passage from another source, you should put it in between **<blockquote>...</blockquote>** tags.

Text inside a <blockquote> element is usually indented from the left and right edges of the surrounding text, and sometimes uses an italicized font.

Example

```
<!DOCTYPE html>
<html>

 <head>
    <title>Blockquote Example</title>
 </head>

 <body>
   <p>The following description of XHTML is taken from the W3C Web site:</p>
 <blockquote>XHTML 1.0 is the W3C's first Recommendation for XHTML, following on
```

from earlier work on HTML 4.01, HTML 4.0, HTML 3.2 and HTML 2.0.</blockquote>

</body>

</html>

This will produce the following result –

The following description of XHTML is taken from the W3C Web site:

XHTML 1.0 is the W3C's first Recommendation for XHTML, following on from earlier work on HTML 4.01, HTML 4.0, HTML 3.2 and HTML 2.0.

Short Quotations

The **<q>...</q>** element is used when you want to add a double quote within a sentence.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Double Quote Example</title>
   </head>

   <body>
      <p>Amit is in Spain, <q>I think I am wrong</q>.</p>
   </body>

</html>
```

This will produce the following result –

Amit is in Spain, “I think I am wrong”.

Text Citations

If you are quoting a text, you can indicate the source placing it between an opening **<cite>** tag and closing **</cite>** tag

As you would expect in a print publication, the content of the <cite> element is rendered in italicized text by default.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Citations Example</title>
   </head>

   <body>
      <p>This HTML tutorial is derived from <cite>W3 Standard for HTML</cite>.</p>
   </body>

</html>
```

This will produce the following result –

This HTML tutorial is derived from *W3 Standard for HTML*.

Computer Code

Any programming code to appear on a Web page should be placed inside **<code>...</code>** tags. Usually the content of the

<code> element is presented in a monospaced font, just like the code in most programming books.

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Computer Code Example</title>
  </head>

  <body>
    <p>Regular text. <code>This is code.</code> Regular text.</p>
  </body>

</html>
```

This will produce the following result –

Regular text. `This is code`. Regular text.

Keyboard Text

When you are talking about computers, if you want to tell a reader to enter some text, you can use the **<kbd>...</kbd>** element to indicate what should be typed in, as in this example.

Example

```
<!DOCTYPE html>
<html>
```

```
<head>
   <title>Keyboard Text Example</title>
</head>

<body>
   <p>Regular text. <kbd>This is inside kbd element</kbd> Regular text.</p>
</body>

</html>
```

This will produce the following result –

Regular text. `This is inside kbd element` Regular text.

Programming Variables

This element is usually used in conjunction with the **<pre>** and **<code>**elements to indicate that the content of that element is a variable.

Example

```
<!DOCTYPE html>
<html>

<head>
   <title>Variable Text Example</title>
</head>

<body>
   <p><code>document.write("<var>user-name</var>")</code></p>
```

```
</body>

</html>
```

This will produce the following result –

document.write("*user-name*")

Program Output

The **<samp>...</samp>** element indicates sample output from a program, and script etc. Again, it is mainly used when documenting programming or coding concepts.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>Program Output Example</title>
   </head>

   <body>
      <p>Result produced by the program is <samp>Hello World!</samp></p>
   </body>

</html>
```

This will produce the following result –

Result produced by the program is `Hello World!`

Address Text

The **<address>...</address>** element is used to contain any address.

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>Address Example</title>
  </head>

  <body>
   <address>388A, Road No 22, Jubilee Hills - Hyderabad</address>
  </body>

</html>
```

This will produce the following result –

388A, Road No 22, Jubilee Hills - Hyderabad

IMAGES IN HTML

Images can improve the design and the appearance of a web page.

Example

```
<!DOCTYPE html>
<html>
<body>
```

<img src="pulpitrock.jpg" alt="Mountain View" width="500" height="377">

</body>

</html>

This will produce the following result –



HTML Images Syntax

In HTML, images are defined with the **<img>** tag.

The <img> tag is empty, it contains attributes only, and does not have a closing tag.

The src attribute specifies the URL (web address) of the image:

<img src="*url*">

The alt Attribute

The alt attribute provides an alternate text for an image, if the user for some reason cannot view it (because of slow connection, an error in the src attribute, or if the user uses a screen reader).

The value of the alt attribute should describe the image:

Example

<!DOCTYPE html>

```
<html>

<body>

<p>The alt attribute should reflect the image content, so users who cannot see the image gets an understanding of what the image contains:</p>

<img src="img_chania.jpg" alt="Flowers in Chania" width="460" height="345">

</body>

</html>
```

This will produce the following result –

The alt attribute should reflect the image content, so users who cannot see the image gets an understanding of what the image contains:



Image Size - Width and Height

You can use the **style** attribute to specify the width and height of an image.

Example

```
<!DOCTYPE html>

<html>

<body>
```

<img src="img_girl.jpg" alt="Girl in a jacket" style="width :500px;height:600px;">

</body>

</html>

This will produce the following result –



The width and height attributes always defines the width and height of the image in pixels.

Note: Always specify the width and height of an image. If width and height are not specified, the page might flicker while the image loads.

Width and Height, or Style?

Both the width, height, and style attributes are valid in HTML5.

However, we suggest using the style attribute. It prevents styles sheets from changing the size of images:

Example

<!DOCTYPE html>

<html>

<head>

```
<style>
/*This stylesheet sets the width of all images to 100%: */
img {
    width:100%;
}
</style>
</head>
<body>

<p>
```

The image below has the width attribute set to 128 pixels, but the stylesheet overrides it, and sets the width to 100%.

```
</p>
<img src="html5.gif" alt="HTML5 Icon" width="128" height="128">
<p>The image below uses the style attribute, where the width is set to 128 pixels which overrides the stylesheet:</p>
<img src="html5.gif" alt="HTML5 Icon" style="width:128px;height:128px;">
</body>
</html>
```

This will produce the following result –

The image below has the width attribute set to 128 pixels, but the stylesheet overrides it, and sets the width to 100%.



The image below uses the style attribute, where the width is set to 128 pixels which overrides the stylesheet:



Images in Another Folder

If not specified, the browser expects to find the image in the same folder as the web page.

However, it is common to store images in a sub-folder. You must then include the folder name in the src attribute:

Example

```
<!DOCTYPE html>
<html>
<body>

<img src="/images/html5.gif" alt="HTML5 Icon" style="wi
dth:128px;height:128px;">
</body>
</html>
```

This will produce the following result –



Images on Another Server

Some web sites store their images on image servers.

Actually, you can access images from any web address in the world:

Example

```
<!DOCTYPE html>
<html>
<body>

<img src="https://www.w3schools.com/images/w3schools_green.jpg" alt="W3Schools.com" style="width:104px;height:142px;">

</body>
</html>
```

This will produce the following result –



Animated Images

HTML allows animated GIFs:

Example

```
<!DOCTYPE html>
<html>
```

```
<body>

<p>The GIF standard allows moving images.</p>

<img src="programming.gif" alt="Computer man" style="w
idth:48px;height:48px;">

</body>
</html>
```

This will produce the following result –

The GIF standard allows moving images.



Image as a Link

To use an image as a link, put the <img> tag inside the <a> tag:

Example

```
<!DOCTYPE html>
<html>
<body>

<p>The image is a link. You can click on it.</p>

<a href="default.asp">
  <img src="smiley.gif" alt="HTML tutorial" style="width:4
2px;height:42px;border:0;">
</a>
```

Add “border:0;” to prevent IE9 (and earlier) from displaying a border around the image.

</body>

</html>

This will produce the following result –

The image is a link. You can click on it.



Add “border:0;” to prevent IE9 (and earlier) from displaying a border around the image.

HTML - TABLES

The HTML tables allow web authors to arrange data like text, images, links, other tables, etc. into rows and columns of cells.

The HTML tables are created using the **<table>** tag in which the **<tr>** tag is used to create table rows and **<td>** tag is used to create data cells. The elements under <td> are regular and left aligned by default

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Tables</title>
   </head>

   <body>
      <table border = “1”>
         <tr>
```

```
      <td>Row 1, Column 1</td>
      <td>Row 1, Column 2</td>
   </tr>

   <tr>
      <td>Row 2, Column 1</td>
      <td>Row 2, Column 2</td>
   </tr>
</table>

</body>
</html>
```

This will produce the following result –

Row 1, Column 1	Row 1, Column 2
Row 2, Column 1	Row 2, Column 2

Here, the **border** is an attribute of <table> tag and it is used to put a border across all the cells. If you do not need a border, then you can use border = "0".

Table Heading

Table heading can be defined using **<th>** tag. This tag will be put to replace <td> tag, which is used to represent actual data cell. Normally you will put your top row as table heading as shown below, otherwise you can use <th> element in any row. Headings, which are defined in <th> tag are centered and bold by default.

Example

```
<!DOCTYPE html>
```

```
<html>

   <head>
      <title>HTML Table Header</title>
   </head>

   <body>
      <table border = "1">
         <tr>
            <th>Name</th>
            <th>Salary</th>
         </tr>
         <tr>
            <td>Ramesh Raman</td>
            <td>5000</td>
         </tr>

         <tr>
            <td>Shabbir Hussein</td>
            <td>7000</td>
         </tr>
      </table>
   </body>

</html>
```

This will produce the following result –

Name	Salary
Ramesh Raman	5000
Shabbir Hussein	7000

Cellpadding and Cellspacing Attributes

There are two attributes called *cellpadding* and *cellspacing* which you will use to adjust the white space in your table cells. The cellspacing attribute defines space between table cells, while cellpadding represents the distance between cell borders and the content within a cell.

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Table Cellpadding</title>
  </head>

  <body>
    <table border = "1" cellpadding = "5" cellspacing = "5">
      <tr>
        <th>Name</th>
        <th>Salary</th>
      </tr>
      <tr>
        <td>Ramesh Raman</td>
        <td>5000</td>
      </tr>
```

```
      <tr>
         <td>Shabbir Hussein</td>
         <td>7000</td>
      </tr>
   </table>
</body>

</html>
```

This will produce the following result –

Name	Salary
Ramesh Raman	5000
Shabbir Hussein	7000

Colspan and Rowspan Attributes

You will use **colspan** attribute if you want to merge two or more columns into a single column. Similar way you will use **rowspan** if you want to merge two or more rows.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Table Colspan/Rowspan</title>
   </head>

   <body>
      <table border = "1">
```

```
      <tr>
         <th>Column 1</th>
         <th>Column 2</th>
         <th>Column 3</th>
      </tr>
      <tr>
         <td rowspan = "2">Row 1 Cell 1</td>
         <td>Row 1 Cell 2</td>
         <td>Row 1 Cell 3</td>
      </tr>
      <tr>
         <td>Row 2 Cell 2</td>
         <td>Row 2 Cell 3</td>
      </tr>
      <tr>
         <td colspan = "3">Row 3 Cell 1</td>
      </tr>
   </table>
</body>

</html>
```

This will produce the following result –

Column 1	Column 2	Column 3
Row 1 Cell 1	Row 1 Cell 2	Row 1 Cell 3
	Row 2 Cell 2	Row 2 Cell 3
Row 3 Cell 1		

Tables Backgrounds

You can set table background using one of the following two ways –

- **bgcolor** attribute – You can set background color for whole table or just for one cell.
- **background** attribute – You can set background image for whole table or just for one cell.

You can also set border color also using **bordercolor** attribute.

Note – The *bgcolor*, *background*, and *bordercolor* attributes deprecated in HTML5. Do not use these attributes.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Table Background</title>
   </head>

   <body>
      <table border = "1" bordercolor = "green" bgcolor = "yellow">
         <tr>
            <th>Column 1</th>
            <th>Column 2</th>
            <th>Column 3</th>
         </tr>
         <tr>
            <td rowspan = "2">Row 1 Cell 1</td>
```

```
            <td>Row 1 Cell 2</td>
            <td>Row 1 Cell 3</td>
         </tr>
         <tr>
            <td>Row 2 Cell 2</td>
            <td>Row 2 Cell 3</td>
         </tr>
         <tr>
            <td colspan = "3">Row 3 Cell 1</td>
         </tr>
      </table>
   </body>

</html>
```

This will produce the following result –

Column 1	Column 2	Column 3
Row 1 Cell 1	Row 1 Cell 2	Row 1 Cell 3
	Row 2 Cell 2	Row 2 Cell 3
Row 3 Cell 1		

Table Height and Width

You can set a table width and height using **width** and **height** attributes. You can specify table width or height in terms of pixels or in terms of percentage of available screen area.

Example

```
<!DOCTYPE html>
<html>
```

```
<head>
   <title>HTML Table Width/Height</title>
</head>

<body>
   <table border = "1" width = "400" height = "150">
      <tr>
         <td>Row 1, Column 1</td>
         <td>Row 1, Column 2</td>
      </tr>

      <tr>
         <td>Row 2, Column 1</td>
         <td>Row 2, Column 2</td>
      </tr>
   </table>
</body>

</html>
```

This will produce the following result –

Row 1, Column 1	Row 1 ,Column 2
Row 2 Column 1	Row 2, Column 2

Table Caption

The **caption** tag will serve as a title or explanation for the table and it shows up at the top of the table. This tag is deprecated in newer version of HTML/XHTML.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Table Caption</title>
   </head>

   <body>
      <table border = "1" width = "100%">
         <caption>This is the caption</caption>

         <tr>
         <td>row 1, column 1</td><td>row 1, columnn 2</td>
         </tr>

         <tr>
         <td>row 2, column 1</td><td>row 2, columnn 2</td>
         </tr>
      </table>
   </body>

</html>
```

This will produce the following result −

This is the caption

row 1, column 1	row 1, column 2
row 2, column 1	row 2, column 2

Table Header, Body, and Footer

Tables can be divided into three portions – a header, a body, and a foot. The head and foot are rather similar to headers and footers in a word-processed document that remain the same for every page, while the body is the main content holder of the table.

The three elements for separating the head, body, and foot of a table are –

- **<thead>** – to create a separate table header.
- **<tbody>** – to indicate the main body of the table.
- **<tfoot>** – to create a separate table footer.

A table may contain several <tbody> elements to indicate *different pages* or groups of data. But it is notable that <thead> and <tfoot> tags should appear before <tbody>

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Table</title>
   </head>

   <body>
      <table border = "1" width = "100%">
         <thead>
            <tr>
               <td colspan = "4">This is the head of the table</
td>
            </tr>
```

```
      </thead>

      <tfoot>
         <tr>
         <td colspan = "4">This is the foot of the table</td>
         </tr>
      </tfoot>

      <tbody>
         <tr>
            <td>Cell 1</td>
            <td>Cell 2</td>
            <td>Cell 3</td>
            <td>Cell 4</td>
         </tr>
      </tbody>

   </table>
</body>

</html>
```

This will produce the following result –

This is the head of the table			
Cell 1	Cell 2	Cell 3	Cell 4
This is the foot of the table			

Nested Tables

You can use one table inside another table. Not only tables you can use almost all the tags inside table data tag <td>.

Example

Following is the example of using another table and other tags inside a table cell.

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Table</title>
  </head>

  <body>
    <table border = "1" width = "100%">

      <tr>
        <td>
          <table border = "1" width = "100%">
            <tr>
              <th>Name</th>
              <th>Salary</th>
            </tr>
            <tr>
              <td>Ramesh Raman</td>
              <td>5000</td>
            </tr>
```

```
            <tr>
               <td>Shabbir Hussein</td>
               <td>7000</td>
            </tr>
         </table>
      </td>
   </tr>

</table>
</body>

</html>
```

This will produce the following result –

Name	Salary
Ramesh Raman	5000
Shabbir Hussein	7000

LISTS IN HTML

HTML offers web authors three ways for specifying lists of information. All lists must contain one or more list elements. Lists may contain –

- **<ul>** – An unordered list. This will list items using plain bullets.
- **<ol>** – An ordered list. This will use different schemes of numbers to list your items.
- **<dl>** – A definition list. This arranges your items in the same way as they are arranged in a dictionary.

HTML Unordered Lists

An unordered list is a collection of related items that have no special order or sequence. This list is created by using HTML **<ul>** tag. Each item in the list is marked with a bullet.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Unordered List</title>
   </head>

   <body>
      <ul>
         <li>Beetroot</li>
         <li>Ginger</li>
         <li>Potato</li>
         <li>Radish</li>
      </ul>
   </body>

</html>
```

This will produce the following result –

- Beetroot
- Ginger
- Potato
- Radish

The type Attribute

You can use **type** attribute for <ul> tag to specify the type of bullet you like. By default, it is a disc. Following are the possible options –

<ul type = “square”>

<ul type = “disc”>

<ul type = “circle”>

Example

Following is an example where we used <ul type = “square”>

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Unordered List</title>
   </head>

   <body>
      <ul type = “square”>
         <li>Beetroot</li>
         <li>Ginger</li>
         <li>Potato</li>
         <li>Radish</li>
      </ul>
   </body>

</html>
```

This will produce the following result –

- Beetroot
- Ginger
- Potato
- Radish

Example

Following is an example where we used <ul type = “disc”> –

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Unordered List</title>
   </head>

   <body>
      <ul type = “disc”>
         <li>Beetroot</li>
         <li>Ginger</li>
         <li>Potato</li>
         <li>Radish</li>
      </ul>
   </body>

</html>
```

This will produce the following result –

- Beetroot
- Ginger
- Potato
- Radish

Example

Following is an example where we used <ul type = “circle”> –

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Unordered List</title>
   </head>

   <body>
      <ul type = “circle”>
         <li>Beetroot</li>
         <li>Ginger</li>
         <li>Potato</li>
         <li>Radish</li>
      </ul>
   </body>

</html>
```

This will produce the following result –

- Beetroot
- Ginger
- Potato
- Radish

HTML Ordered Lists

If you are required to put your items in a numbered list instead of bulleted, then HTML ordered list will be used. This list is

created by using **<ol>** tag. The numbering starts at one and is incremented by one for each successive ordered list element tagged with <li>.

Example

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Ordered List</title>
   </head>

   <body>
      <ol>
         <li>Beetroot</li>
         <li>Ginger</li>
         <li>Potato</li>
         <li>Radish</li>
      </ol>
   </body>

</html>
```

This will produce the following result –

1. Beetroot
2. Ginger
3. Potato
4. Radish

The type Attribute

You can use **type** attribute for <ol> tag to specify the type of numbering you like. By default, it is a number. Following are the possible options –

<ol type = “1”> - Default-Case Numerals.

<ol type = “I”> - Upper-Case Numerals.

<ol type = “i”> - Lower-Case Numerals.

<ol type = “A”> - Upper-Case Letters.

<ol type = “a”> - Lower-Case Letters.

Example

Following is an example where we used <ol type = “1”>

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Ordered List</title>
   </head>

   <body>
      <ol type = “1”>
         <li>Beetroot</li>
         <li>Ginger</li>
         <li>Potato</li>
         <li>Radish</li>
      </ol>
   </body>
```

```
</html>
```

This will produce the following result –

1. Beetroot
2. Ginger
3. Potato
4. Radish

Example

Following is an example where we used <ol type = "I">

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Ordered List</title>
   </head>

   <body>
      <ol type = "I">
         <li>Beetroot</li>
         <li>Ginger</li>
         <li>Potato</li>
         <li>Radish</li>
      </ol>
   </body>

</html>
```

This will produce the following result –

I. Beetroot

II. Ginger
III. Potato
IV. Radish

Example

Following is an example where we used <ol type = “i”>

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Ordered List</title>
   </head>

   <body>
      <ol type = “i”>
         <li>Beetroot</li>
         <li>Ginger</li>
         <li>Potato</li>
         <li>Radish</li>
      </ol>
   </body>

</html>
```

This will produce the following result –

i. Beetroot
ii. Ginger
iii. Potato
iv. Radish

Example

Following is an example where we used <ol type = "A" >

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Ordered List</title>
   </head>

   <body>
      <ol type = "A">
         <li>Beetroot</li>
         <li>Ginger</li>
         <li>Potato</li>
         <li>Radish</li>
      </ol>
   </body>

</html>
```

This will produce the following result –

Beetroot

A. Ginger
B. Potato
C. Radish

Example

Following is an example where we used <ol type = "a">

```
<!DOCTYPE html>
```

```
<html>

   <head>
      <title>HTML Ordered List</title>
   </head>

   <body>
      <ol type = "a">
         <li>Beetroot</li>
         <li>Ginger</li>
         <li>Potato</li>
         <li>Radish</li>
      </ol>
   </body>

</html>
```

This will produce the following result –

a. Beetroot
b. Ginger
c. Potato
d. Radish

The start Attribute

You can use **start** attribute for <ol> tag to specify the starting point of numbering you need. Following are the possible options –

<ol type = "1" start = "4"> - Numerals starts with 4.

<ol type = "I" start = "4"> - Numerals starts with IV.

<ol type = "i" start = "4"> - Numerals starts with iv.

<ol type = “a” start = “4”> - Letters starts with d.

<ol type = “A” start = “4”> - Letters starts with D.

Example

Following is an example where we used <ol type = “i” start = “4” >

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Ordered List</title>
   </head>

   <body>
      <ol type = “i” start = “4”>
         <li>Beetroot</li>
         <li>Ginger</li>
         <li>Potato</li>
         <li>Radish</li>
      </ol>
   </body>

</html>
```

This will produce the following result –

IV. Beetroot
V. Ginger
VI. Potato
VII. Radish

HTML Definition Lists

HTML and XHTML supports a list style which is called **definition lists** where entries are listed like in a dictionary or encyclopedia. The definition list is the ideal way to present a glossary, list of terms, or other name/value list.

Definition List makes use of following three tags.

- <dl> – Defines the start of the list
- <dt> – A term
- <dd> – Term definition
- </dl> – Defines the end of the list

Example

```
<!DOCTYPE html>
<html>

  <head>
    <title>HTML Definition List</title>
  </head>

  <body>
    <dl>
      <dt><b>HTML</b></dt>
      <dd>This stands for Hyper Text Markup Language</
dd>
      <dt><b>HTTP</b></dt>
      <dd>This stands for Hyper Text Transfer Protocol</dd>
    </dl>
  </body>
```

```
</html>
```

This will produce the following result –

HTML

This stands for Hyper Text Markup Language

HTTP

This stands for Hyper Text Transfer Protocol

HTML FRAMES

HTML frames are used to divide your browser window into multiple sections where each section can load a separate HTML document. A collection of frames in the browser window is known as a frameset. The window is divided into frames in a similar way the tables are organized: into rows and columns.

Disadvantages of Frames

There are few drawbacks with using frames, so it's never recommended to use frames in your webpages –

- Some smaller devices cannot cope with frames often because their screen is not big enough to be divided up.
- Sometimes your page will be displayed differently on different computers due to different screen resolution.
- The browser's *back* button might not work as the user hopes.
- There are still few browsers that do not support frame technology.

Creating Frames

To use frames on a page we use <frameset> tag instead of <body> tag. The <frameset> tag defines, how to divide the window into frames. The **rows**attribute of <frameset> tag defines horizontal frames and **cols** attribute defines vertical frames. Each frame is indicated by <frame> tag and it defines which HTML document shall open into the frame.

Note – The <frame> tag deprecated in HTML5. Do not use this element.

Example

Following is the example to create three horizontal frames –

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Frames</title>
   </head>

   <frameset rows = "10%,80%,10%">
      <frame name = "top" src = "/html/top_frame.htm" />
     <frame name = "main" src = "/html/main_frame.htm" />
      <frame name = "bottom" src = "/html/bottom_frame.
htm" />

      <noframes>
       <body>Your browser does not support frames.</body>
      </noframes>

   </frameset>

</html>
```

This will produce the following result –



Example

Let's put the above example as follows, here we replaced rows attribute by cols and changed their width. This will create all the three frames vertically –

```
<!DOCTYPE html>
<html>

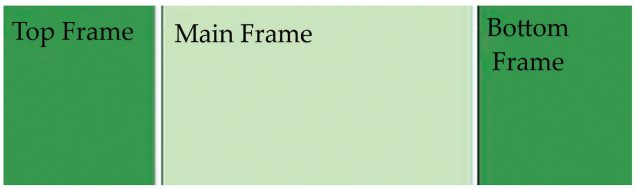
   <head>
      <title>HTML Frames</title>
   </head>

   <frameset cols = "25%,50%,25%">
      <frame name = "left" src = "/html/top_frame.htm" />
    <frame name = "center" src = "/html/main_frame.htm" />
    <frame name = "right" src = "/html/bottom_frame.htm" />

      <noframes>
       <body>Your browser does not support frames.</body>
      </noframes>
   </frameset>

</html>
```

This will produce the following result –



The <frameset> Tag Attributes

Following are important attributes of the <frameset> tag –

Sr. No	Attribute	Description
1	**cols**	Specifies how many columns are contained in the frameset and the size of each column. You can specify the width of each column in one of the four ways – Absolute values in pixels. For example, to create three vertical frames, use *cols = "100, 500, 100"*. A percentage of the browser window. For example, to create three vertical frames, use *cols = "10%, 80%, 10%"*. Using a wildcard symbol. For example, to create three vertical frames, use *cols = "10%, *, 10%"*. In this case wildcard takes remainder of the window. As relative widths of the browser window. For example, to create three vertical frames, use *cols = "3*, 2*, 1*"*. This is an alternative to percentages. You can use relative widths of the browser window. Here the window is divided into sixths: the first column takes up half of the window, the second takes one third, and the third takes one sixth.
2	**rows**	This attribute works just like the cols attribute and takes the same values, but it is used to specify the rows in the frameset. For example, to create two horizontal frames, use *rows = "10%, 90%"*. You can specify the height of each row in the same way as explained above for columns.
3	**border**	This attribute specifies the width of the border of each frame in pixels. For example, border = "5". A value of zero means no border.
4	**frameborder**	This attribute specifies whether a three-dimensional border should be displayed between frames. This attribute takes value either 1 (yes) or 0 (no). For example frameborder = "0" specifies no border.
5	**framespacing**	This attribute specifies the amount of space between frames in a frameset. This can take any integer value. For example framespacing = "10" means there should be 10 pixels spacing between each frames.

The <frame> Tag Attributes

Following are the important attributes of <frame> tag –

Sr. No	Attribute	Description
1	**src**	This attribute is used to give the file name that should be loaded in the frame. Its value can be any URL. For example, src = "/html/top_frame.htm" will load an HTML file available in html directory.
2	**name**	This attribute allows you to give a name to a frame. It is used to indicate which frame a document should be loaded into. This is especially important when you want to create links in one frame that load pages into an another frame, in which case the second frame needs a name to identify itself as the target of the link.
3	**frameborder**	This attribute specifies whether or not the borders of that frame are shown; it overrides the value given in the frameborder attribute on the <frameset> tag if one is given, and this can take values either 1 (yes) or 0 (no).
4	**marginwidth**	This attribute allows you to specify the width of the space between the left and right of the frame's borders and the frame's content. The value is given in pixels. For example marginwidth = "10".
5	**marginheight**	This attribute allows you to specify the height of the space between the top and bottom of the frame's borders and its contents. The value is given in pixels. For example marginheight = "10".
6	**noresize**	By default, you can resize any frame by clicking and dragging on the borders of a frame. The noresize attribute prevents a user from being able to resize the frame. For example noresize = "noresize".
7	**scrolling**	This attribute controls the appearance of the scrollbars that appear on the frame. This takes values either "yes", "no" or "auto". For example scrolling = "no" means it should not have scroll bars.
8	**longdesc**	This attribute allows you to provide a link to another page containing a long description of the contents of the frame. For example longdesc = "framedescription.htm"

Browser Support for Frames

If a user is using any old browser or any browser, which does not support frames then <noframes> element should be

displayed to the user. So you must place a <body> element inside the <noframes> element because the <frameset> element is supposed to replace the <body> element, but if a browser does not understand <frameset> element then it should understand what is inside the <body> element which is contained in a <noframes> element.

Frame's name and target attributes

One of the most popular uses of frames is to place navigation bars in one frame and then load main pages into a separate frame.

Let's see following example where a test.htm file has following code −

```
<!DOCTYPE html>
<html>

   <head>
      <title>HTML Target Frames</title>
   </head>

   <frameset cols = "200, *">
     <frame src = "/html/menu.htm" name = "menu_page" />
      <frame src = "/html/main.htm" name = "main_page" />

      <noframes>
       <body>Your browser does not support frames.</body>
      </noframes>
   </frameset>

</html>
```

Here, we have created two columns to fill with two frames. The first frame is 200 pixels wide and will contain the navigation menu bar implemented by menu.htm file. The second column fills in remaining space and will contain the main part of the page and it is implemented by main.htm file. For all the three links available in menu bar, we have mentioned target frame as main_page, so whenever you click any of the links in menu bar, available link will open in main page.

Following is the content of menu.htm file

```
<!DOCTYPE html>

<html>


   <body bgcolor = "#4a7d49">

      <a href = "http://www.google.com" target = "main_
page">Google</a>

      <br />

      <br />


      <a href = "http://www.microsoft.com" target = "main_
page">Microsoft</a>

      <br />

      <br />


       <a href = "http://news.bbc.co.uk" target = "main_
page">BBC News</a>

   </body>


</html>
```

Following is the content of main.htm file –

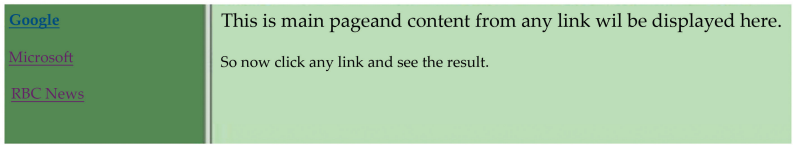
```
<!DOCTYPE html>
```

```
<html>

   <body bgcolor = "#b5dcb3">
      <h3>This is main page and content from any link will be displayed here.</h3>
      <p>So now click any link and see the result.</p>
   </body>

</html>
```

When we load **test.htm** file, it produces following result −



Now you can try to click links available in the left panel and see the result. The *targetattribute* can also take one of the following values −

Sr. No	Option	Description
1	**_self**	Loads the page into the current frame.
2	**_blank**	Loads a page into a new browser window. Opening a new window.
3	**_parent**	Loads the page into the parent window, which in the case of a single frameset is the main browser window.
4	**_top**	Loads the page into the browser window, replacing any current frames.
5	**targetframe**	Loads the page into a named targetframe.

KEY TERMS

Dynamic HTML
HTML element
Monospaced font
Web browser
Cascading Style Sheet (CSS)
Extensible Markup Language (XML)
HTML attribute
Attribute
Uniform Resource Locator (URL)

DISCUSSION QUESTIONS

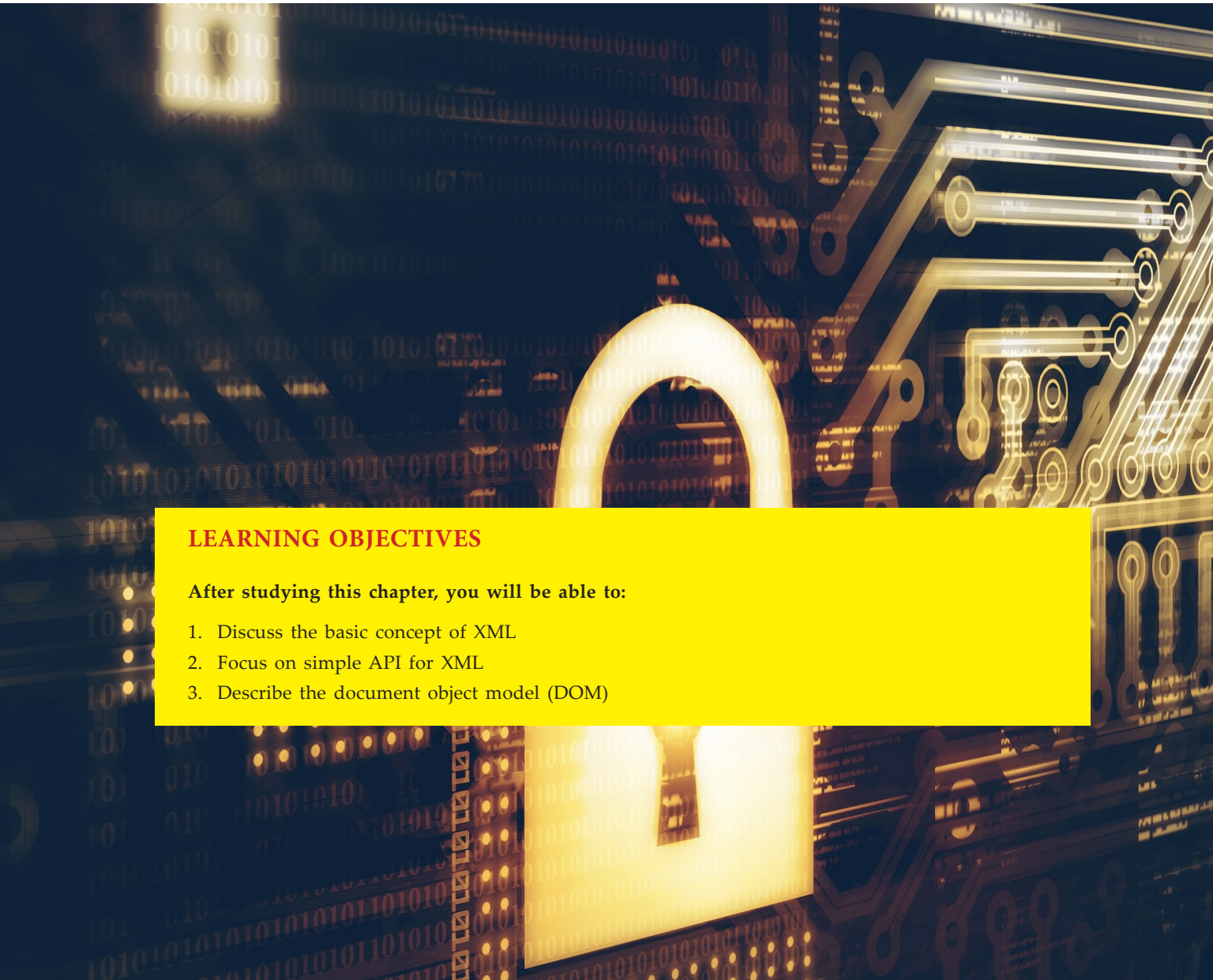
1. Define the term HTML.
2. Explain basic html document.
3. How to create an html document?
4. What are the basic tags of html?
5. Describe the elements of html.
6. How to create frames?
7. Learn about browser support for frames.
8. Focus on frame's name and target attributes

REFERENCES

1. Berners-Lee, Tim; Fischetti, Mark (2000). Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by Its Inventor. San Francisco: Harper. ISBN 978-0-06-251587-2.
2. Bray, T, Paoli, J, Sperberg-McQueen, CM, Maler, E and Yergeau, F (editors) (2008-11-26). "Extensible Markup Language (XML) 1.0 (Fifth Edition)". W3C. Retrieved 2009-03-20.
3. Nigel Shadbolt, Wendy Hall and Tim Berners-Lee (2006). "The Semantic Web Revisited" (PDF). IEEE Intelligent Systems. Retrieved October 2, 2009.
4. Rosendorf, Theodore (2009). The Typographic Desk Reference. New Castle, Delaware: Oak Knoll Press. p. 12. ISBN 978-1-58456-231-3.
5. Shinn, Nick. "Shinntype Modern Suite specification" (PDF). Shinntype. Retrieved 16 October 2015.
6. Spolsky, Joel. "User Interface Design For Programmers". Joel On Software. Retrieved 17 November 2014.
7. Strizver, Elaine. "Proportional vs. Tabular Figures". fonts.com. Monotype Imaging. Retrieved 4 August 2014.
8. Tim Berners-Lee, James Hendler and Ora Lassila (2001). "The Semantic Web". Scientific American. Retrieved October 2, 2009.
9. Berners-Lee, T.; Connolly, D. (June 1993). "Hypertext Markup Language (HTML)" (text). Retrieved 2009-03-28.
10. Berners-Lee, Tim; Connolly, Daniel (June 1993). "Hyper Text Markup Language (HTML) Internet Draft version 1.1". IETF IIIR Working Group. Retrieved 18 September 2010.
11. Berners-Lee, Tim; Connolly, Daniel (June 1993). "Hypertext Markup Language (HTML) Internet Draft version 1.2". IETF IIIR Working Group. Retrieved 18 September 2010.
12. Berners-Lee, Tim; Connolly, Daniel (28 November 1994). "HyperText Markup Language Specification – 2.0 INTERNET DRAFT". IETF. Retrieved 24 October 2010.
13. Raggett, Dave (1995-03-24). "HyperText Markup Language Specification Version 3.0 (draft)". Retrieved 2009-04-18.
14. Raggett, Dave (28 March 1995). "HyperText Markup Language Specification Version 3.0". HTML 3.0 Internet Draft Expires in six months. World Wide Web Consortium. Retrieved 17 June 2010.

CHAPTER 3

EXTENSIBLE MARKUP LANGUAGE (XML)



LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Discuss the basic concept of XML
2. Focus on simple API for XML
3. Describe the document object model (DOM)

INTRODUCTION

Extensible Markup Language (XML) is a markup language and file format for storing, transmitting, and reconstructing arbitrary data. It defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. The World Wide Web Consortium's XML 1.0 Specification of 1998 and several other related specifications—all of them free open standards—define XML. The design goals of XML emphasize simplicity, generality, and usability across the Internet. It is a textual data format with strong support via Unicode for different human languages. Although the design of XML focuses on documents, the language is widely used for the representation of arbitrary data structures such as those used in web services.

Although the simplified example above only hints at the very basics of this language, it should already be clear how it differs from HyperText Markup Language (HTML). The tags used in HTML dictate how material should be presented on a webpage, without indication as to what the material *is*. Manipulation or reuse of data inside a HTML file for further purposes is not viable. However, data contained in a webpage written in XML can be manipulated, extracted by various database systems and recycled.

When using HTML to create webpages *style sheets* are often used. Known formally as *Cascading Style Sheets*, a CSS adds style elements to a HTML webpage. Webpages can also be written in XML. In this case the equivalent of the CSS is the XSL (eXtensible Stylesheet Language), implemented the same way.

The XSL actually serves two functions. It contains instructions to XML-aware browsers how to render the code for graphic purposes, but it also contains instructions for *transforming* the data into other formats, referred to as XLST. XSLT can generate a file different in structure from the original. This is especially useful in areas like e-commerce, for example, where customer input like name, credit card number, dollar amount, etcetera is pushed through a series of system programs in the processing of payment. The transforming function of XSLT is unhindered by rendering needs, and solely concerned with moving data successfully between networks and programs for processing purposes. In an age where exchange of information is in such high demand XML is an obvious choice over HTML.

XML, created by the World Wide Web Consortium (W3C), is part of a large family of markup languages and is defined as a *metalanguage* — a language that describes other languages. One of the goals of W3C was to make XML "optionless" so that it would remain pure, unlike HTML which has many different conventions and as a result is rendered differently on various browsers, making it difficult to present **data** in a uniform fashion. Because XML-pages provide so much more flexibility than HTML pages, it is expected to replace HTML as the language of choice.

BASIC CONCEPT OF XML

Extensible Markup Language (XML) is used to describe data. The XML standard is a flexible way to create information formats and electronically share structured data via the public Internet, as well as via corporate networks.

XML code, a formal recommendation from the World Wide Web Consortium (W3C), is similar to Hypertext Markup Language (HTML**)**. Both XML and HTML contain markup symbols to describe page or file contents. HTML code describes Web page content (mainly text and graphic images) only in terms of how it is to be displayed and interacted with.

XML data is known as self-describing or self-defining, meaning that the structure of the data is embedded with the data, thus when the data arrives there is no need to pre-build the structure to store the data; it is dynamically understood within the XML. The XML format can be used by any individual or group of individuals or companies that want to share information in a consistent way. XML is actually a simpler and easier-to-use subset of the Standard Generalized Markup Language (SGML), which is the standard to create a document structure.

The basic building block of an XML document is an element, defined by tags. An element has a beginning and an ending tag. All elements in an XML document are contained in an outermost element known as the root element. XML can also support nested elements, or elements within elements. This ability allows XML to support hierarchical structures. Element names describe the content of the element, and the structure describes the relationship between the elements.

An XML document is considered to be "well formed" (that is, able to be read and understood by an XML parser) if its format complies with the XML specification, if it is properly marked up, and if elements are properly nested. XML also supports the ability to define attributes for elements and describe characteristics of the elements in the beginning tag of an element.

For example, XML documents can be very simple, such as the following:

<?xml version="1.0" standalone="yes"?>

<conversation>

<greeting>Hello, world!</greeting>

<response>Stop the planet, I want to get off!</response>

</conversation>

Applications for XML are endless. For example, computer makers might agree upon a standard or common way to describe the information about a computer product (processor speed, memory size, and so forth) and then describe the product information format with XML code. Such a standard way of describing data would enable a user to send an intelligent agent (a program) to each computer maker's Web site, gather data, and then make a valid comparison.

XML's benefits sometimes appeared revolutionary in scope shortly after it was introduced. However, as a concept, it fell short of being revolutionary. It also fell short of being the panacea. The over-application of XML in so many areas of technology diminished its real value, and results in a great deal of unnecessary confusion. Perhaps most damaging is the predictable behavior of many vendors that look to recast XML using their own set of proprietary extensions. Although some want to add value to XML, others seek only to lock in users to their products.

XML's power resides in its simplicity. It can take large chunks of information and consolidate them into an XML document meaningful pieces that provide structure and organization to the information.

The XML Document Type Declaration, commonly known as DTD, is a way to describe XML language precisely. DTDs check vocabulary and validity of the structure of XML documents against grammatical rules of appropriate XML language.

An XML DTD can be either specified inside the document, or it can be kept in a separate document and then liked separately.

A Document Type Definition, or DTD, is a set of rules you define for your XML documents. Why would this be important, you ask? Well, let's say you are using XML to exchange financial information between your bank and your home computer. If the format of your XML document is not consistent throughout, your savings account data could be interpreted as your checking account data, causing all kinds of confusion. DTD enables us to define the components of an XML document and put them where we want them to be. But wait, there's more! DTDs can also be used to define *entities*, or common elements for XML documents.

Syntax

Basic syntax of a DTD is as follows −

```
<!DOCTYPE element DTD identifier
[
   declaration1
   declaration2
   ........
]>
```

In the above syntax,

- The DTD starts with <!DOCTYPE delimiter.
- An element tells the parser to parse the document from the specified root element.

- DTD identifier is an identifier for the document type definition, which may be the path to a file on the system or URL to a file on the internet. If the DTD is pointing to external path, it is called External Subset.
- The square brackets [ ] enclose an optional list of entity declarations called *Internal Subset*.

Internal DTD

A DTD is referred to as an internal DTD if elements are declared within the XML files. To refer it as internal DTD, *standalone* attribute in XML declaration must be set to yes. This means, the declaration works independent of an external source.

Syntax

Following is the syntax of internal DTD −

<!DOCTYPE root-element [element-declarations]>

where *root-element* is the name of root element and *element-declarations* is where you declare the elements.

Example

Following is a simple example of internal DTD −

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

<!DOCTYPE address [

<!ELEMENT address (name,company,phone)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT phone (#PCDATA)>

]>

<address>

<name>Tanmay Patil</name>

<company>XYZ</company>

<phone>(011) 123-4567</phone>

</address>

Let us go through the above code –

Start Declaration: Begin the XML declaration with the following statement.

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

DTD: Immediately after the XML header, the *document type declaration* follows, commonly referred to as the DOCTYPE –

<!DOCTYPE address [

The DOCTYPE declaration has an exclamation mark (!) at the start of the element name. The DOCTYPE informs the parser that a DTD is associated with this XML document.

DTD Body: The DOCTYPE declaration is followed by body of the DTD, where you declare elements, attributes, entities, and notations.

<!ELEMENT address (name,company,phone)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

<!ELEMENT phone_no (#PCDATA)>

Several elements are declared here that make up the vocabulary of the <name> document. <!ELEMENT name (#PCDATA)> defines the element *name* to be of type "#PCDATA". Here #PCDATA means parse-able text data.

End Declaration: Finally, the declaration section of the DTD is closed using a closing bracket and a closing angle bracket (**]>**). This effectively ends the definition, and thereafter, the XML document follows immediately.

Rules

- The document type declaration must appear at the start of the document (preceded only by the XML header) – it is not permitted anywhere else within the document.
- Similar to the DOCTYPE declaration, the element declarations must start with an exclamation mark.
- The Name in the document type declaration must match the element type of the root element.

External DTD

In external DTD elements are declared outside the XML file. They are accessed by specifying the system attributes which may be either the legal *.dtd* file or a valid URL. To refer it as external DTD, *standalone* attribute in the XML declaration must be set as **no**. This means, declaration includes information from the external source.

Syntax

Following is the syntax for external DTD –

```
<!DOCTYPE root-element SYSTEM "file-name">
```

where *file-name* is the file with *.dtd* extension.

Example

The following example shows external DTD usage –

```
<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>
<!DOCTYPE address SYSTEM "address.dtd">
<address>
   <name>Tanmay Patil</name>
   <company>XYZ</company>
   <phone>(011) 123-4567</phone>
</address>
```

The content of the DTD file address.dtd is as shown −

```
<!ELEMENT address (name,company,phone)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT company (#PCDATA)>
<!ELEMENT phone (#PCDATA)>
```

Types

You can refer to an external DTD by using either system identifiers or public identifiers.

System Identifiers

A system identifier enables you to specify the location of an external file containing DTD declarations. Syntax is as follows −

```
<!DOCTYPE name SYSTEM "address.dtd" [...]>
```

As you can see, it contains keyword SYSTEM and a URI reference pointing to the location of the document.

Public Identifiers

Public identifiers provide a mechanism to locate DTD resources and is written as follows −

```
<!DOCTYPE name PUBLIC "-//Beginning XML//DTD Address Example//EN">
```

As you can see, it begins with keyword PUBLIC, followed by a specialized identifier. Public identifiers are used to identify an entry in a catalog. Public identifiers can follow any format, however, a commonly used format is called Formal Public Identifiers, or FPIs.

XML Usage

XML stands for **E**xtensible **M**arkup **L**anguage. It is a text-based markup language derived from Standard Generalized Markup Language (SGML).

XML tags identify the data and are used to store and organize the data, rather than specifying how to display it like HTML tags, which are used to display the data. XML is not going to replace HTML in the near future, but it introduces new possibilities by adopting many successful features of HTML.

A short list of XML usage says it all −

- XML can work behind the scene to simplify the creation of HTML documents for large web sites.
- XML can be used to exchange the information between organizations and systems.
- XML can be used for offloading and reloading of databases.
- XML can be used to store and arrange the data, which can customize your data handling needs.
- XML can easily be merged with style sheets to create almost any desired output.
- Virtually, any type of data can be expressed as an XML document.

Markup

XML is a markup language that defines set of rules for encoding documents in a format that is both human-readable and machine-readable. So *what exactly is a markup language?* Markup is information added to a document that enhances its meaning in certain ways, in that it identifies the parts and how they relate to each other. More specifically, a markup language is a set of symbols that can be placed in the text of a document to demarcate and label the parts of that document.

Following example shows how XML markup looks, when embedded in a piece of text −

```
<message>
   <text>Hello, world!</text>
</message>
```

This snippet includes the markup symbols, or the tags such as <message>...</message> and <text>... </text>. The tags <message> and </message> mark the start and the end of the XML code

fragment. The tags <text> and </text> surround the text Hello, world!.

Declaration of XML

XML declaration contains details that prepare an XML processor to parse the XML document. It is optional, but when used, it must appear in the first line of the XML document.

Syntax

Following syntax shows XML declaration –

```
<?xml
   version = "version_number"
   encoding = "encoding_declaration"
   standalone = "standalone_status"
?>
```

Each parameter consists of a parameter name, an equals sign (=), and parameter value inside a quote.

Rules

An XML declaration should abide with the following rules –

- If the XML declaration is present in the XML, it must be placed as the first line in the XML document.
- If the XML declaration is included, it must contain version number attribute.
- The Parameter names and values are case-sensitive.
- The names are always in lower case.
- The order of placing the parameters is important. The correct order is: *version, encoding and standalone.*
- Either single or double quotes may be used.
- The XML declaration has no closing tag i.e. </?xml>

XML Declaration Examples

Following are few examples of XML declarations –

XML declaration with no parameters –

<?xml >

XML declaration with version definition –

<?xml version = "1.0">

XML declaration with all parameters defined –

<?xml version = "1.0" encoding = "UTF-8" standalone = "no" ?>

XML declaration with all parameters defined in single quotes –

<?xml version = '1.0' encoding = 'iso-8859-1' standalone = 'no' ?>

XML Tags

Let us learn about one of the most important part of XML, the XML tags. XML tags form the foundation of XML. They define the scope of an element in XML. They can also be used to insert comments, declare settings required for parsing the environment, and to insert special instructions.

We can broadly categorize XML tags as follows –

Start Tag

The beginning of every non-empty XML element is marked by a start-tag. Following is an example of start-tag –

<address>

End Tag

Every element that has a start tag should end with an end-tag. Following is an example of end-tag –

</address>

Note, that the end tags include a solidus ("/") before the name of an element.

Empty Tag

The text that appears between start-tag and end-tag is called content. An element which has no content is termed as empty. An empty element can be represented in two ways as follows −

A start-tag immediately followed by an end-tag as shown below −

<hr></hr>

A complete empty-element tag is as shown below −

<hr />

Empty-element tags may be used for any element which has no content.

XML Tags Rules

Following are the rules that need to be followed to use XML tags −

Rule 1

XML tags are case-sensitive. Following line of code is an example of wrong syntax </Address>, because of the case difference in two tags, which is treated as erroneous syntax in XML.

<address>This is wrong syntax</Address>

Following code shows a correct way, where we use the same case to name the start and the end tag.

<address>This is correct syntax</address>

Rule 2

XML tags must be closed in an appropriate order, i.e., an XML tag opened inside another element must be closed before the outer element is closed. For example −

<outer_element>

<internal_element>

This tag is closed before the outer_element

```
    </internal_element>
</outer_element>
```

Character Entities

"The document entity serves as the root of the entity tree and a starting-point for an XML processor".

This means, entities are the placeholders in XML. These can be declared in the document prolog or in a DTD.

Both, HTML and XML, have some symbols reserved for their use, which cannot be used as content in XML code. For example, < and > signs are used for opening and closing XML tags. To display these special characters, the character entities are used.

There are few special characters or symbols which are not available to be typed directly from the keyboard. Character Entities can also be used to display those symbols/special characters.

Types of Character Entities

There are three types of character entities –

- Predefined Character Entities
- Numbered Character Entities
- Named Character Entities

Predefined Character Entities

They are introduced to avoid the ambiguity while using some symbols. For example, an ambiguity is observed when less than (<) or greater than (>) symbol is used with the angle tag (<>). Character entities are basically used to delimit tags in XML. Following is a list of pre-defined character entities from XML specification. These can be used to express characters without ambiguity.

- Ampersand – &
- Single quote – '
- Greater than – >

- Less than – <
- Double quote – "

Numeric Character Entities

The numeric reference is used to refer to a character entity. Numeric reference can either be in decimal or hexadecimal format. As there are thousands of numeric references available, these are a bit hard to remember. Numeric reference refers to the character by its number in the Unicode character set.

General syntax for decimal numeric reference is –

&# decimal number;

General syntax for hexadecimal numeric reference is –

&#x Hexadecimal number;

The following table lists some predefined character entities with their numeric values –

Entity name	Character	Decimal reference	Hexadecimal reference
quot	"	"	"
amp	&	&	&
apos	'	'	'
lt	<	<	<
gt	>	>	>

Named Character Entity

As it is hard to remember the numeric characters, the most preferred type of character entity is the named character entity. Here, each entity is identified with a name.

For example –

- 'Aacute' represents capital Á character with acute accent.
- 'ugrave' represents the small ù with grave accent.

Validation of XML

Validation is a process by which an XML document is validated. An XML document is said to be valid if its contents match with the elements, attributes and associated document type declaration (DTD), and if the document complies with the constraints expressed in it. Validation is dealt in two ways by the XML parser. They are −

- Well-formed XML document
- Valid XML document

Well-formed XML Document

An XML document is said to be **well-formed** if it adheres to the following rules −

- Non DTD XML files must use the predefined character entities for amp(&), apos(single quote), gt(>), lt(<), quot(double quote).
- It must follow the ordering of the tag. i.e., the inner tag must be closed before closing the outer tag.
- Each of its opening tags must have a closing tag or it must be a self-ending tag.(<title>....</title> or <title/>).
- It must have only one attribute in a start tag, which needs to be quoted.
- amp(&), apos(single quote), gt(>), lt(<), quot(double quote)entities other than these must be declared.

Example

Following is an example of a well-formed XML document −

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

<!DOCTYPE address

[

<!ELEMENT address (name,company,phone)>

<!ELEMENT name (#PCDATA)>

<!ELEMENT company (#PCDATA)>

```
    <!ELEMENT phone (#PCDATA)>
]>


<address>
    <name>Tanmay Patil</name>
    <company>XYZ</company>
    <phone>(011) 123-4567</phone>
</address>
```

The above example is said to be well-formed as –

- It defines the type of document. Here, the document type is elementtype.
- It includes a root element named as address.
- Each of the child elements among name, company and phone is enclosed in its self-explanatory tag.
- Order of the tags is maintained.

Valid XML Document

If an XML document is well-formed and has an associated Document Type Declaration (DTD), then it is said to be a valid XML document.

XML Schemas

An XML Schema is a language for expressing constraints about XML documents. There are several different schema languages in widespread use, but the main ones are Document Type Definitions (DTDs), Relax-NG, Schematron and W3C XSD (XML Schema Definitions). From this page you can find out more about DTDs and W3C XSD, since those are the primary schema languages defined at W3C.

XML Schema Standards:

- XML Schema Definition (XSD) is currently the de facto standard for describing XML documents and is the XML Schema standard we will concentrate. XSD is controlled

by the World Wide Web Consortium (W3C). An XSD is itself an XML document, and there is even an XSD to describe the XSD standard.

- Document Type Definition (DTD) was the first formalized standard but has now, in most cases, been superseded by XSD.
- XML Data Reduced (XDR) was an early attempt but Microsoft to provide a more comprehensive standard than DTD. This standard has been phased out in the Microsoft products in favor of XSD.
- There are also a number of other schema standards such as Schematron and RELAX NG.

XML Schema is commonly known as XML Schema Definition (XSD). It is used to describe and validate the structure and the content of XML data. XML schema defines the elements, attributes and data types. Schema element supports Namespaces. It is similar to a database schema that describes the data in a database.

Syntax

You need to declare a schema in your XML document as follows –

Example

The following example shows how to use schema −

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

<xs:schema xmlns:xs = "http://www.w3.org/2001/
XMLSchema">

   <xs:element name = "contact">

      <xs:complexType>

         <xs:sequence>

            <xs:element name = "name" type = "xs:string" />

            <xs:element name = "company" type = "xs:string" />

            <xs:element name = "phone" type = "xs:int" />

         </xs:sequence>
```

```
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The basic idea behind XML Schemas is that they describe the legitimate format that an XML document can take.

Using for XML Schema

A Schema can be used:

- to provide a list of elements and attributes in a vocabulary;
- to associate types, such as integer, string, etc., or more specifically such as hatsize, sock_colour, etc., with values found in documents;
- to provide documentation that is both human-readable and machine-processable;
- to give a formal description of one or more documents.

Information in schema documents is often used by XML-aware editing systems so that they can offer users the most likely elements to occur at any given location in a document.

Checking a document against a Schema is known as validating against that schema; for a DTD, this is just *validating*, but for any other type of schema the type is mentioned, such as XSD Validation or Relax-NG validation.

Validating against a schema is an important component of quality assurance. The Service Modeling Language (SML) provides a framework for relating multiple XSD documents to one or more documents in a single validation episode.

Since XSD supports associating data types with element and attribute content, it is also used for *data binding*, that is, for software components that read and write XML representations of **computer** programming-language objects.

Advantages of Schemas

There are some advantages of schemas, such as:

- Define the characteristics and syntax of a set of documents.
- Independent groups can have a common format for interchanging XML documents.
- Software applications that process the XML documents know what to expect if the documents adhere to a formal schema.
- XML documents can be validated to verify that they conform to a given schema.
- Validation can be used as a debugging tool, directing the designer to items in a document that violate the schema.
- A schema can act as documentation for users defining or reading some set of XML documents.
- A schema can increase the reliability, consistency, and accuracy of exchanged documents.

Elements

As we saw in the XML elements are the building blocks of XML document. An element can be defined within an XSD as follows −

```
<xs:element name = "x" type = "y"/>
```

Definition Types

You can define XML schema elements in the following ways −

- *Simple Type*: Simple type element is used only in the context of the text. Some of the predefined simple types are: xs:integer, xs:boolean, xs:string, xs:date. For example −

```
<xs:element name = "phone_number" type = "xs:int" />
```

- *Complex Type*: A complex type is a container for other element definitions. This allows you to specify which child elements an element can contain and to provide some structure within your XML documents. For example −

```
<xs:element name = "Address">
   <xs:complexType>
      <xs:sequence>
```

```
        <xs:element name = "name" type = "xs:string" />
        <xs:element name = "company" type = "xs:string" />
        <xs:element name = "phone" type = "xs:int" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

In the above example, *Address* element consists of child elements. This is a container for other <xs:element> definitions, that allows to build a simple hierarchy of elements in the XML document.

- *Global Types:* With the global type, you can define a single type in your document, which can be used by all other references. For example, suppose you want to generalize the *person* and *company* for different addresses of the company. In such case, you can define a general type as follows −

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

Now let us use this type in our example as follows −

```
<xs:element name = "Address1">
  <xs:complexType>
    <xs:sequence>
      <xs:element name = "address" type = "AddressType" />
      <xs:element name = "phone1" type = "xs:int" />
```

```
        </xs:sequence>
    </xs:complexType>
</xs:element>
<xs:element name = "Address2">
    <xs:complexType>
        <xs:sequence>
        <xs:element name = "address" type = "AddressType" />
            <xs:element name = "phone2" type = "xs:int" />
        </xs:sequence>
    </xs:complexType>
</xs:element>
```

Instead of having to define the name and the company twice (once for *Address1* and once for *Address2*), we now have a single definition. This makes maintenance simpler, i.e., if you decide to add "Postcode" elements to the address, you need to add them at just one place.

Attributes

Attributes in XSD provide extra information within an element. Attributes have *name* and *type* property as shown below −

```
<xs:attribute name = "x" type = "y"/>
```

Applications of XML

Extensible Markup Language (XML) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications, all gratis open standards.

The design goals of XML emphasize simplicity, generality, and usability over the Internet. It is a textual data format with strong support via Unicode for the languages of the world. Although the design of XML focuses on documents, it is widely used for the

representation of arbitrary data structures, for example in web services. Many application programming interfaces (APIs) have been developed to aid software developers with processing XML data, and several schema systems exist to aid in the definition of XML-based languages.

As of 2009, hundreds of document formats using XML syntax have been developed, including RSS, Atom, SOAP, and XHTML. XML-based formats have become the default for many office-productivity tools, including Microsoft Office (Office Open XML), OpenOffice.org and LibreOffice (OpenDocument), and Apple's iWork. XML has also been employed as the base language for communication protocols, such as XMPP.

Well-formedness and error-handling

The XML specification defines an XML document as a well-formed text - meaning that it satisfies a list of syntax rules provided in the specification. Some key points in the fairly lengthy list include:

The document contains only properly encoded legal Unicode characters

- None of the special syntax characters such as < and & appear except when performing their markup-delineation roles
- The begin, end, and empty-element tags that delimit the elements are correctly nested, with none missing and none overlapping
- The element tags are case-sensitive; the beginning and end tags must match exactly. Tag names cannot contain any of the characters !"#$%&'()*+,/;<=>?@[\]^`{|}~, nor a space character, and cannot start with -, ., or a numeric digit.
- A single "root" element contains all the other elements

The definition of an XML document excludes texts that contain violations of well-formedness rules; they are simply not XML. An XML processor that encounters such a violation is required to report such errors and to cease normal processing. This policy, occasionally referred to as draconian, stands in notable contrast to the behavior of programs that process HTML, which are designed to produce a reasonable result even in the

presence of severe markup errors. XML's policy in this area has been criticized as a violation of Postel's law ("Be conservative in what you send; be liberal in what you accept").

The XML specification defines a valid XML document as a well-formed XML document which also conforms to the rules of a Document Type Definition (DTD). By extension, the term can also refer to documents that conform to rules in other schema languages, such as XML Schema (XSD). This term should not be confused with a well-formed XML document, which is defined as an XML document that has correct XML syntax according to W3C standards.

SIMPLE API FOR XML

SAX (Simple API for XML) is an application program interface (API) that allows a programmer to interpret a Web file that uses the Extensible Markup Language (XML) - that is, a Web file that describes a collection of data. SAX is an alternative to using the Document Object Model (DOM) to interpret the XML file. As its name suggests, it is a simpler interface than DOM and is appropriate where many or very large files are to be processed, but it contains fewer capabilities for manipulating the data content.

SAX is an *event-driven* interface. The programmer specifies an event that may happen and, if it does, SAX gets control and handles the situation. SAX works directly with an XML parser.

SAX was developed collaboratively by members of the XML-DEV mailing list (currently hosted by OASIS). The original version of SAX, which was specific to Java, was the first API for XML in Java to gain broad industry support.

Benefits of SAX

SAX parsers have some benefits over DOM-style parsers. A SAX parser only needs to report each parsing event as it happens, and normally discards almost all of that information once reported (it does, however, keep some things, for example a list of all elements that have not been closed yet, in order to catch later errors such as end-tags in the wrong order). Thus,

the minimum memory required for a SAX parser is proportional to the maximum depth of the XML file (i.e., of the XML tree) and the maximum data involved in a single XML event (such as the name and attributes of a single start-tag, or the content of a processing instruction, etc.).

This much memory is usually considered negligible. A DOM parser, in contrast, typically builds a tree representation of the entire document in memory to begin with, thus using memory that increases with the entire document length. This takes considerable time and space for large documents (memory allocation and data-structure construction take time). The compensating advantage, of course, is that once loaded any part of the document can be accessed in any order.

Because of the event-driven nature of SAX, processing documents is generally far faster than DOM-style parsers, so long as the processing can be done in a start-to-end pass. Many tasks, such as indexing, conversion to other formats, very simple formatting, and the like, can be done that way. Other tasks, such as sorting, rearranging sections, getting from a link to its target, looking up information on one element to help process a later one, and the like, require accessing the document structure in complex orders and will be much faster with DOM than with multiple SAX passes.

Some implementations do not neatly fit either category: a DOM approach can keep its persistent data on disk, cleverly organized for speed (editors such as SoftQuad Author/Editor and large-document browser/indexers such as DynaText do this); while a SAX approach can cleverly cache information for later use (any validating SAX parser keeps more information than described above). Such implementations blur the DOM/SAX tradeoffs, but are often very effective in practice.

Due to the nature of DOM, streamed reading from disk requires techniques such as lazy evaluation, caches, virtual memory, persistent data structures, or other techniques. Processing XML documents larger than main memory is sometimes thought impossible because some DOM parsers do not allow it. However, it is no less possible than sorting a dataset larger than main memory using disk space as memory to sidestep this limitation.

Drawbacks of SAX

The event-driven model of SAX is useful for XML parsing, but it does have certain drawbacks. Virtually any kind of XML validation requires access to the document in full. The most trivial example is that an attribute declared in the DTD to be of type IDREF, requires that there be an element in the document that uses the same value for an ID attribute. To validate this in a SAX parser, one must keep track of all ID attributes (any one of them might end up being referenced by an IDREF attribute at the very end); as well as every IDREF attribute until it is resolved. Similarly, to validate that each element has an acceptable sequence of child elements, information about what child elements have been seen for each parent, must be kept until the parent closes.

Additionally, some kinds of XML processing simply require having access to the entire document. XSLT and XPath, for example, need to be able to access any node at any time in the parsed XML tree. Editors and browsers likewise need to be able to display, modify, and perhaps re-validate at any time. While a SAX parser may well be used to construct such a tree initially, SAX provides no help for such processing as a whole.

XML processing with SAX

A parser that implements SAX (i.e., a SAX Parser) functions as a stream parser, with an event-driven API. The user defines a number of callback methods that will be called when events occur during parsing. The SAX events include (among others):

- XML Text nodes
- XML Element Starts and Ends
- XML Processing Instructions
- XML Comments

Some events correspond to XML objects that are easily returned all at once, such as comments. However, XML elements can contain many other XML objects, and so SAX represents them as does XML itself: by one event at the beginning, and another at the end. Properly speaking, the SAX interface does not deal in elements, but in events that largely correspond to tags. SAX parsing is unidirectional; previously parsed data cannot be re-read without starting the parsing operation again.

There are many SAX-like implementations in existence. In practice, details vary, but the overall model is the same. For example, XML attributes are typically provided as name and value arguments passed to element events, but can also be provided as separate events, or via a hash or similar collection of all the attributes. For another, some implementations provide "Init" and "Fin" callbacks for the very start and end of parsing; others do not. The exact names for given event types also vary slightly between implementations.

DOCUMENT OBJECT MODEL (DOM)

Document Object Model (DOM), a programming interface specification being developed by the World Wide Web Consortium (W3C), lets a programmer create and modify HTML pages and XML documents as full-fledged program objects. Currently, HTML (Hypertext Markup Language) and XML (Extensible Markup Language) are ways to express a document in terms of a data structure. As program objects, such documents will be able to have their contents and data "hidden" within the object, helping to ensure control over who can manipulate the document. As objects, documents can carry with them the object-oriented procedures called methods. DOM is a strategic and open effort to specify how to provide programming control over documents. It was inspired in part by the advent of the new HTML capabilities generally called dynamic HTML and as a way to encourage consistent browser behavior with Web page and their elements.

DOM defines the objects and properties and methods (interface) to access all XML elements. It is separated into 3 different parts / levels –

- Core DOM – standard model for any structured document
- XML DOM – standard model for XML documents
- HTML DOM – standard model for HTML documents

XML DOM is a standard object model for XML. XML documents have a hierarchy of informational units called *nodes*; DOM is a standard programming interface of describing those nodes and the relationships between them.

As XML DOM also provides an API that allows a developer to add, edit, move or remove nodes at any point on the tree in order to create an application.

Following is the diagram for the DOM structure. The diagram depicts that parser evaluates an XML document as a DOM structure by traversing through each node.

Advantages of XML DOM

The following are the advantages of XML DOM.

- XML DOM is language and platform independent.
- XML DOM is traversable - Information in XML DOM is organized in a hierarchy which allows developer to navigate around the hierarchy looking for specific information.
- XML DOM is modifiable - It is dynamic in nature providing the developer a scope to add, edit, move or remove nodes at any point on the tree.

Disadvantages of XML DOM

- It consumes more memory (if the XML structure is large) as program written once remains in memory all the time until and unless removed explicitly.
- Due to the extensive usage of memory, its operational speed, compared to SAX is slower.

Model of DOM

Now that we know what DOM means, let's see what a DOM structure is. A DOM document is a collection of *nodes* or pieces of information, organized in a hierarchy. Some types of *nodes* may have *child* nodes of various types and others are leaf nodes that cannot have anything under them in the document structure. Following is a list of the node types, with a list of node types that they may have as children −

- **Document** − Element (maximum of one), ProcessingInstruction, Comment, DocumentType (maximum of one)

- **DocumentFragment** – Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- **EntityReference** – Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- **Element** – Element, Text, Comment, ProcessingInstruction, CDATASection, EntityReference
- **Attr** – Text, EntityReference
- **ProcessingInstruction** – No children
- **Comment** – No children
- **Text** – No children
- **CDATASection** – No children
- **Entity** – Element, ProcessingInstruction, Comment, Text, CDATASection, EntityReference
- **Notation** – No children

Example

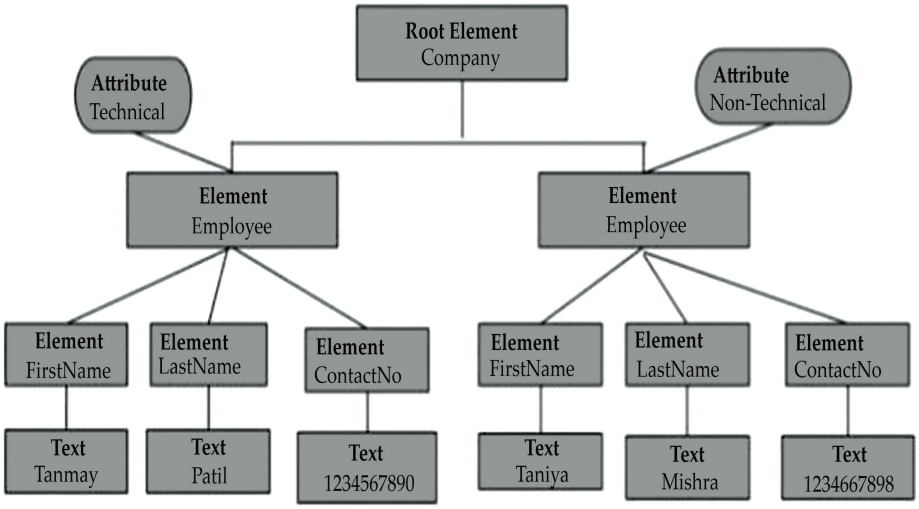
Consider the DOM representation of the following XML document node.xml.

```
<?xml version = "1.0"?>
<Company>
   <Employee category = "technical">
      <FirstName>Tanmay</FirstName>
      <LastName>Patil</LastName>
      <ContactNo>1234567890</ContactNo>
   </Employee>

   <Employee category = "non-technical">
      <FirstName>Taniya</FirstName>
      <LastName>Mishra</LastName>
      <ContactNo>1234667898</ContactNo>
   </Employee>
```

</Company>

The Document Object Model of the above XML document would be as follows –



From the above flowchart, we can infer –

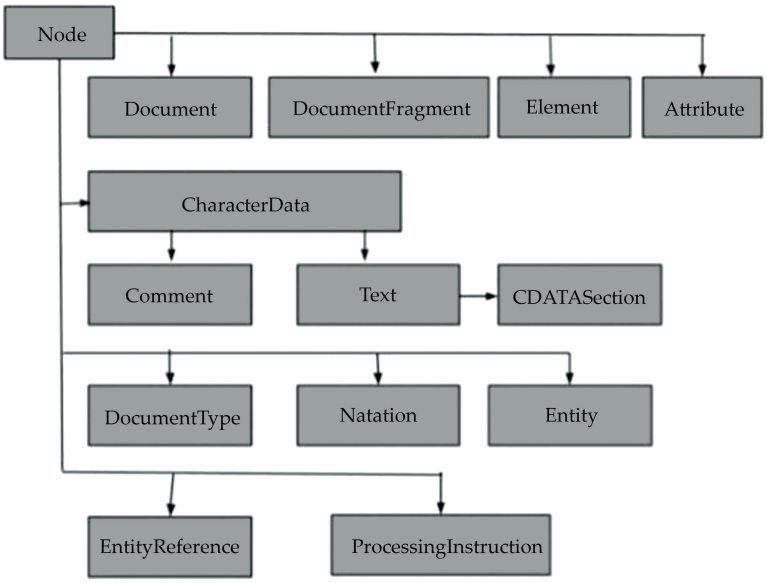
- *Node* object can have only one *parent node* object. This occupies the position above all the nodes. Here it is *Company*.
- The *parent node* can have multiple nodes called the *child* nodes. These *child* nodes can have additional nodes called the *attribute* nodes. In the above example, we have two attribute nodes *Technical* and *Non-technical*. The *attribute* node is not actually a child of the element node, but is still associated with it.
- These *child* nodes in turn can have multiple child nodes. The text within the nodes is called the *text* node.
- The node objects at the same level are called as siblings.
- The DOM identifies –
  - the objects to represent the interface and manipulate the document.
  - the relationship among the objects and interfaces.

Nodes of DOM

Every XML DOM contains the information in hierarchical units called *Nodes* and the DOM describes these nodes and the relationship between them.

Node Types

The following flowchart shows all the node types –



The most common types of nodes in XML are –

- **Document Node** – Complete XML document structure is a *document node*.
- **Element Node** – Every XML element is an *element node*. This is also the only type of node that can have attributes.
- **Attribute Node** – Each attribute is considered an *attribute node*. It contains information about an element node, but is not actually considered to be children of the element.
- **Text Node** – The document texts are considered as *text node*. It can consist of more information or just white space.

Some less common types of nodes are –

- CData Node – This node contains information that should

not be analyzed by the parser. Instead, it should just be passed on as plain text.

- Comment Node – This node includes information about the data, and is usually ignored by the application.
- Processing Instructions Node – This node contains information specifically aimed at the application.
- Document Fragments Node
- Entities Node
- Entity reference nodes
- Notations Node

Node Tree

In an XML document, the information is maintained in hierarchical structure; this hierarchical structure is referred to as the *Node Tree*. This hierarchy allows a developer to navigate around the tree looking for specific information, thus nodes are allowed to access. The content of these nodes can then be updated.

The structure of the node tree begins with the root element and spreads out to the child elements till the lowest level.

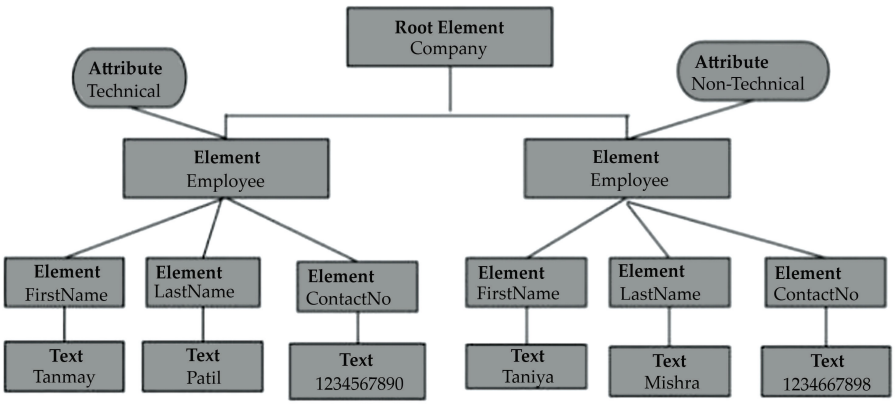
Example

Following example demonstrates a simple XML document, whose node tree is structure is shown in the diagram below –

```
<?xml version = "1.0"?>
<Company>
   <Employee category = "Technical">
      <FirstName>Tanmay</FirstName>
      <LastName>Patil</LastName>
      <ContactNo>1234567890</ContactNo>
   </Employee>
   <Employee category = "Non-Technical">
      <FirstName>Taniya</FirstName>
```

```
    <LastName>Mishra</LastName>
    <ContactNo>1234667898</ContactNo>
  </Employee>
</Company>
```

As can be seen in the above example whose pictorial representation (of its DOM) is as shown below –



Citation: https://www.tutorialspoint.com/dom/images/xml_dom_model.jpg

- The topmost node of a tree is called the root. The root node is <Company> which in turn contains the two nodes of <Employee>. These nodes are referred to as child nodes.
- The child node <Employee> of root node <Company>, in turn consists of its own child node (<FirstName>, <LastName>, <ContactNo>).
- The two child nodes, <Employee> have attribute values Technical and Non-Technical, are referred as *attribute nodes*.
- The text within every node is called the *text node*.

Methods of DOM

DOM as an API contains interfaces that represent different types of information that can be found in an XML document, such as elements and text.

Following table lists the DOM classes and interfaces –

S.No.	Interface & Description
1	**DOMImplementation** It provides a number of methods for performing operations that are independent of any particular instance of the document object model.
2	**DocumentFragment** It is the "lightweight" or "minimal" document object, and it (as the superclass of Document) anchors the XML/HTML tree in a full-fledged document.
3	**Document** It represents the XML document's top-level node, which provides access to all the nodes in the document, including the root element.
4	**Node** It represents XML node.
5	**NodeList** It represents a read-only list of *Node* objects.
6	**NamedNodeMap** It represents collections of nodes that can be accessed by name.
7	**Data** It extends *Node* with a set of attributes and methods for accessing character data in the DOM.
8	**Attribute** It represents an attribute in an Element object.
9	**Element** It represents the element node. Derives from Node.
10	**Text** It represents the text node. Derives from CharacterData.
11	**Comment** It represents the comment node. Derives from CharacterData.
12	**ProcessingInstruction** It represents a "processing instruction". It is used in XML as a way to keep processor-specific information in the text of the document.
13	**CDATA Section** It represents the CDATA Section. Derives from Text.
14	**Entity** It represents an entity. Derives from Node.
15	**EntityReference** This represent an entity reference in the tree. Derives from Node.

These interfaces include the methods and properties necessary to work with these objects. Properties define the characteristic of the node whereas methods give the way to manipulate the nodes.

Loading of DOM

In order to describe the interfaces provided by the API, the W3C uses an abstract language called the Interface Definition Language (IDL). The advantage of using IDL is that the developer learns how to use the DOM with his or her favorite language and can switch easily to a different language.

The disadvantage is that, since it is abstract, the IDL cannot be used directly by Web developers. Due to the differences between programming languages, they need to have mapping — or binding — between the abstract interfaces and their concrete languages. DOM has been mapped to programming languages such as Javascript, JScript, Java, C, C++, PLSQL, Python, and Perl.

Parser

A *parser* is a software application that is designed to analyze a document, in our case XML document and do something specific with the information. Some of the DOM based parsers are listed in the following table −

S.No	Parser & Description
1	**JAXP** Sun Microsystem's Java API for XML Parsing (JAXP)
2	**XML4J** IBM's XML Parser for Java (XML4J)
3	**msxml** Microsoft's XML parser (msxml) version 2.0 is built-into Internet Explorer 5.5
4	**4DOM** 4DOM is a parser for the Python programming language

5	**XML::DOM** XML::DOM is a Perl module to manipulate XML documents using Perl
6	**Xerces** Apache's Xerces Java Parser

In a tree-based API like DOM, the parser traverses the XML file and creates the corresponding DOM objects. Then you can traverse the DOM structure back and forth.

Loading and Parsing XML

While loading an XML document, the XML content can come in two forms –

- Directly as XML file
- As XML string

Content as XML file

Following example demonstrates how to load XML (node.xml) data using Ajax and Javascript when the XML content is received as an XML file. Here, the Ajax function gets the content of an xml file and stores it in XML DOM. Once the DOM object is created, it is then parsed.

```
<!DOCTYPE html>
<html>
   <body>
      <div>
      <b>FirstName:</b> <span id = "FirstName"></span><br>
      <b>LastName:</b> <span id = "LastName"></span><br>
      <b>ContactNo:</b> <span id = "ContactNo"></span><br>
         <b>Email:</b> <span id = "Email"></span>
      </div>
      <script>
```

```
//if browser supports XMLHttpRequest

if (window.XMLHttpRequest) { // Create an instance
of XMLHttpRequest object.
code for IE7+, Firefox, Chrome, Opera, Safari
xmlhttp = new XMLHttpRequest();
} else { // code for IE6, IE5
xmlhttp = new ActiveXObject("Microsoft.
XMLHTTP");
}

// sets and sends the request for calling "node.xml"
xmlhttp.open("GET","/dom/node.xml",false);
xmlhttp.send();

// sets and returns the content as XML DOM
xmlDoc = xmlhttp.responseXML;

//parsing the DOM object
document.getElementById("FirstName").innerHTML =
xmlDoc.getElementsByTagName("FirstName")
[0].childNodes[0].nodeValue;
document.getElementById("LastName").innerHTML =
xmlDoc.getElementsByTagName("LastName")
[0].childNodes[0].nodeValue;
document.getElementById("ContactNo").innerHTML =
xmlDoc.getElementsByTagName("ContactNo")
[0].childNodes[0].nodeValue;
document.getElementById("Email").innerHTML =
```

```
                    xmlDoc.getElementsByTagName("Email")[0].
childNodes[0].nodeValue;
        </script>
    </body>
</html>

node.xml
<Company>
    <Employee category = "Technical" id = "firstelement">
        <FirstName>Tanmay</FirstName>
        <LastName>Patil</LastName>
        <ContactNo>1234567890</ContactNo>
        <Email>tanmaypatil@xyz.com</Email>
    </Employee>

    <Employee category = "Non-Technical">
        <FirstName>Taniya</FirstName>
        <LastName>Mishra</LastName>
        <ContactNo>1234667898</ContactNo>
        <Email>taniyamishra@xyz.com</Email>
    </Employee>

    <Employee category = "Management">
        <FirstName>Tanisha</FirstName>
        <LastName>Sharma</LastName>
        <ContactNo>1234562350</ContactNo>
        <Email>tanishasharma@xyz.com</Email>
```

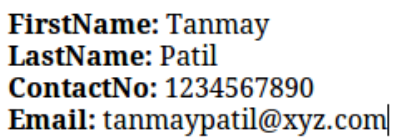
```
    </Employee>
</Company>
```

Most of the details of the code are in the script code.

- Internet Explorer uses the *ActiveXObject("Microsoft.XMLHTTP")* to create an instance of XMLHttpRequest object, other browsers use the *XMLHttpRequest()* method.
- the *responseXML* transforms the XML content directly in XML DOM.
- Once the XML content is transformed into JavaScript XML DOM, you can access any XML element by using the JS DOM methods and properties. We have used the DOM properties such as *childNodes*, *nodeValue* and DOM methods such as getElementsById(ID), getElementsByTagName(tags_name).

Execution

Save this file as loadingexample.html and open it in your browser. You will receive the following output –



Content as XML string

Following example demonstrates how to load XML data using Ajax and Javascript when XML content is received as XML file. Here, the Ajax function, gets the content of an xml file and stores it in XML DOM. Once the DOM object is created it is then parsed.

```
<!DOCTYPE html>
<html>
   <head>
```

```
<script>

// loads the xml string in a dom object
function loadXMLString(t) { // for non IE browsers
if (window.DOMParser) {
// create an instance for xml dom object parser = new DOMParser();
xmlDoc = parser.parseFromString(t,"text/xml");
}
// code for IE
else { // create an instance for xml dom object
xmlDoc = new ActiveXObject("Microsoft.XMLDOM");
xmlDoc.async = false;
xmlDoc.loadXML(t);
}
return xmlDoc;
}
</script>
</head>
<body>
<script>

// a variable with the string
var text = "<Employee>";
text = text+"<FirstName>Tanmay</FirstName>";
text = text+"<LastName>Patil</LastName>";
```

```
        text = text+"<ContactNo>1234567890</ContactNo>";
        text = text+"<Email>tanmaypatil@xyz.com</Email>";
        text = text+"</Employee>";
      // calls the loadXMLString() with "text" function and store the
xml dom in a variable
        var xmlDoc = loadXMLString(text);

       //parsing the DOM object
        y = xmlDoc.documentElement.childNodes;
        for (i = 0;i<y.length;i++) {
          document.write(y[i].childNodes[0].nodeValue);
          document.write("<br>");
        }
    </script>
  </body>
</html>
```

Most of the details of the code are in the script code.

- Internet Explorer uses the *ActiveXObject("Microsoft.XMLDOM")* to load XML data into a DOM object, other browsers use the *DOMParser()*function and *parseFromString(text, 'text/xml')* method.
- The variable *text* shall contain a string with XML content.
- Once the XML content is transformed into JavaScript XML DOM, you can access any XML element by using JS DOM methods and properties. We have used DOM properties such as *childNodes*, *nodeValue*.

Execution

Save this file as loadingexample.html and open it in your browser. You will see the following output –

Tanmay
Patil
1234567890
tanmaypatil@xyz.com

Now that we saw how the XML content transforms into JavaScript XML DOM, you can now access any XML element by using the XML DOM methods.

Traversing of DOM

This parsed DOM object can be traversed. Traversing is a process in which looping is done in a systematic manner by going across each and every element step by step in a node tree.

Example

The following example (traverse_example.htm) demonstrates DOM traversing. Here we traverse through each child node of <Employee> element.

```
<!DOCTYPE html>
<html>
   <style>
      table,th,td {
         border:1px solid black;
         border-collapse:collapse
      }
   </style>
   <body>
      <div id = “ajax_xml”></div>
      <script>
         //if browser supports XMLHttpRequest
         if (window.XMLHttpRequest) {// Create an instance of XMLHttpRequest object.
```

```
        code for IE7+, Firefox, Chrome, Opera, Safari
        var xmlhttp = new XMLHttpRequest();
    } else {// code for IE6, IE5
        var xmlhttp = new ActiveXObject("Microsoft.
XMLHTTP");
    }
    // sets and sends the request for calling "node.xml"
        xmlhttp.open("GET","/dom/node.xml",false);
        xmlhttp.send();
    // sets and returns the content as XML DOM
        var xml_dom = xmlhttp.responseXML;

    // this variable stores the code of the html table
        var html_tab = '<table id = "id_tabel" align =
"center">
        <tr>
            <th>Employee Category</th>
            <th>FirstName</th>
            <th>LastName</th>
            <th>ContactNo</th>
            <th>Email</th>
        </tr>';
        var arr_employees = xml_dom.getElementsByTag
Name("Employee");
    // traverses the "arr_employees" array
        for(var i = 0; i<arr_employees.length; i++) {
            var employee_cat = arr_employees[i].
getAttribute('category');
```

```
                    // gets the value of 'category' element of current
"Element" tag

                  // gets the value of first child-node of 'FirstName'

                    // element of current "Employee" tag

                      var employee_firstName =

                        arr_employees[i].getElementsByTagName(
'FirstName')[0].childNodes[0].nodeValue;


                  // gets the value of first child-node of 'LastName'

                    // element of current "Employee" tag

                      var employee_lastName =

               arr_employees[i].getElementsByTagName('LastName')
[0].childNodes[0].nodeValue;


                  // gets the value of first child-node of 'ContactNo'

                    // element of current "Employee" tag

                      var employee_contactno =

                        arr_employees[i].getElementsByTagName(
'ContactNo')[0].childNodes[0].nodeValue;


                    // gets the value of first child-node of 'Email'

                    // element of current "Employee" tag

                      var employee_email =

                  arr_employees[i].getElementsByTagName('Email')
[0].childNodes[0].nodeValue;


                    // adds the values in the html table
```

```
                html_tab += '<tr>
                    <td>'+ employee_cat+ '</td>
            <td>'+ employee_firstName+ '</td>
                    <td>'+ employee_lastName+ '</td>
                    <td>'+ employee_contactno+ '</td>
                    <td>'+ employee_email+ '</td>
                </tr>';
              }
            html_tab += '</table>';
      // adds the html table in a html tag, with id = "ajax_xml"
            document.getElementById('ajax_xml').innerHTML =
html_tab;
        </script>
    </body>
</html>
```

- This code loads node.xml.
- The XML content is transformed into JavaScript XML DOM object.
- The array of elements (with tag Element) using the method getElementsByTagName() is obtained.
- Next, we traverse through this array and display the child node values in a table.

Execution

Save this file as *traverse_example.html* on the server path (this file and node.xml should be on the same path in your server). You will receive the following output –

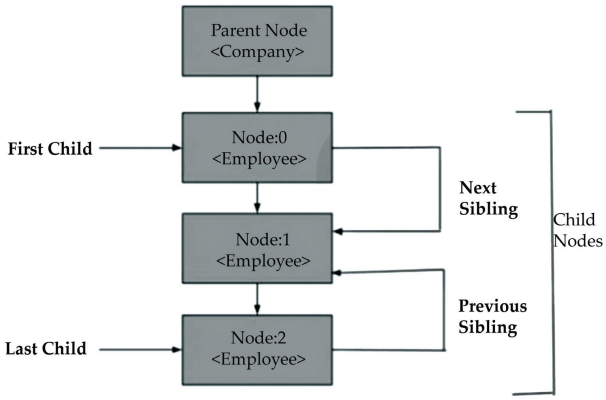
Employee Categ ory	FirstName	LastName	ContactNo	Email
Technical	Tanmay	Patil	1234367890	tanmaypatil@xyz.com
Non-Technical	Taniya	Mishra	1234667898	taniyamishra@xyz.com
Management	Tanisha	Sharma	1234362350	tanishasharma@xyz.com

Navigation of DOM

Until now we studied DOM structure, how to load and parse XML DOM object and traverse through the DOM objects. Here we will see how we can navigate between nodes in a DOM object. The XML DOM consist of various properties of the nodes which help us navigate through the nodes, such as –

- parentNode
- childNodes
- firstChild
- lastChild
- nextSibling
- previousSibling

Following is a diagram of a node tree showing its relationship with the other nodes.



DOM - Parent Node

This property specifies the parent node as a node object.

Example

The following example (navigate_example.htm) parses an XML document (node.xml) into an XML DOM object. Then the DOM object is navigated to the parent node through the child node –

```
<!DOCTYPE html>
```

```
<html>
  <body>
    <script>
      if (window.XMLHttpRequest) {
        xmlhttp = new XMLHttpRequest();
      } else {
      xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      xmlhttp.open("GET","/dom/node.xml",false);
      xmlhttp.send();
      xmlDoc = xmlhttp.responseXML;

      var y = xmlDoc.getElementsByTagName("Employee")
[0];
      document.write(y.parentNode.nodeName);
    </script>
  </body>
</html>
```

As you can see in the above example, the child node *Employee* navigates to its parent node.

Execution

Save this file as *navigate_example.html* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the parent node of *Employee*, i.e., *Company*.

First Child

This property is of type *Node* and represents the first child name present in the NodeList.

Example

The following example (first_node_example.htm) parses an XML document (node.xml) into an XML DOM object, then navigates to the first child node present in the DOM object.

```
<!DOCTYPE html>
<html>
  <body>
    <script>
      if (window.XMLHttpRequest) {
        xmlhttp = new XMLHttpRequest();
      } else {
      xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      xmlhttp.open("GET","/dom/node.xml",false);
      xmlhttp.send();
      xmlDoc = xmlhttp.responseXML;
      function get_firstChild(p) {
        a = p.firstChild;

        while (a.nodeType != 1) {
          a = a.nextSibling;
        }
        return a;
      }
          var firstchild = get_firstChild(xmlDoc.
getElementsByTagName("Employee")[0]);
      document.write(firstchild.nodeName);
```

```
    </script>
  </body>
</html>
```

- Function *get_firstChild(p)* is used to avoid the empty nodes. It helps to get the firstChild element from the node list.
- x = get_firstChild(xmlDoc.getElementsByTagName("Employee")[0]) fetches the first child node for the tag name *Employee*.

Execution

Save this file as *first_node_example.htm* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the first child node of *Employee* i.e. *FirstName*.

Last Child

This property is of type *Node* and represents the last child name present in the NodeList.

Example

The following example (last_node_example.htm) parses an XML document (node.xml) into an XML DOM object, then navigates to the last child node present in the xml DOM object.

```
<!DOCTYPE html>
  <body>
    <script>
      if (window.XMLHttpRequest) {
        xmlhttp = new XMLHttpRequest();
      } else {
      xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
```

```
xmlhttp.open("GET","/dom/node.xml",false);
xmlhttp.send();
xmlDoc = xmlhttp.responseXML;

function get_lastChild(p) {
   a = p.lastChild;

   while (a.nodeType! = 1){
      a = a.previousSibling;
   }
   return a;
}
        var lastchild = get_lastChild(xmlDoc.
getElementsByTagName("Employee")[0]);
document.write(lastchild.nodeName);
</script>
</body>
</html>
```

Execution

Save this file as *last_node_example.htm* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the last child node of *Employee,* i.e, *Email*.

Next Sibling

This property is of type *Node* and represents the next child, i.e, the next sibling of the specified child element present in the NodeList.

Example

The following example (nextSibling_example.htm) parses an XML document (node.xml) into an XML DOM object which navigates immediately to the next node present in the xml document.

```
<!DOCTYPE html>
  <body>
    <script>
      if (window.XMLHttpRequest) {
        xmlhttp = new XMLHttpRequest();
      }
      else {
      xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      xmlhttp.open("GET","/dom/node.xml",false);
      xmlhttp.send();
      xmlDoc = xmlhttp.responseXML;

      function get_nextSibling(p) {
        a = p.nextSibling;

        while (a.nodeType != 1) {
          a = a.nextSibling;
        }
        return a;
      }
      var nextsibling = get_nextSibling(xmlDoc.getElemen
tsByTagName("FirstName")[0]);
      document.write(nextsibling.nodeName);
```

```
    </script>
  </body>
</html>
```

Execution

Save this file as *nextSibling_example.htm* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the next sibling node of *FirstName,* i.e, *LastName*.

Previous Sibling

This property is of type *Node* and represents the previous child, i.e, the previous sibling of the specified child element present in the NodeList.

Example

The following example (previoussibling_example.htm) parses an XML document (node.xml) into an XML DOM object, then navigates the before node of the last child node present in the xml document.

```
<!DOCTYPE html>
  <body>
    <script>
      if (window.XMLHttpRequest)
      {
        xmlhttp = new XMLHttpRequest();
      } else {
     xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
      }
      xmlhttp.open("GET","/dom/node.xml",false);
      xmlhttp.send();
```

```
            xmlDoc = xmlhttp.responseXML;
            function get_previousSibling(p) {
               a = p.previousSibling;
               while (a.nodeType != 1) {
                  a = a.previousSibling;
               }
               return a;
            }
                  prevsibling = get_previousSibling(xmlDoc.
getElementsByTagName("Email")[0]);
            document.write(prevsibling.nodeName);
         </script>
      </body>
   </html>
```

Execution

Save this file as *previoussibling_example.htm* on the server path (this file and node.xml should be on the same path in your server). In the output, we get the previous sibling node of *Email*, i.e, *ContactNo*.

Accessing of DOM

We will study about how to access the XML DOM nodes which are considered as the informational units of the XML document. The node structure of the XML DOM allows the developer to navigate around the tree looking for specific information and simultaneously access the information.

Accessing Nodes

Following are the three ways in which you can access the nodes −

- By using the getElementsByTagName () method

- By looping through or traversing through nodes tree
- By navigating the node tree, using the node relationships

getElementsByTagName ()

This method allows accessing the information of a node by specifying the node name. It also allows accessing the information of the Node List and Node List Length.

Syntax

The getElementByTagName() method has the following syntax –

```
node.getElementByTagName("tagname");
```

Where,

- *node* – is the document node.
- *tagname* – holds the name of the node whose value you want to get.

Example

Following is a simple program which illustrates the usage of method getElementByTagName.

```
<!DOCTYPE html>
<html>
   <body>
      <div>
      <b>FirstName:</b> <span id = "FirstName"></span><br>
      <b>LastName:</b> <span id = "LastName"></span><br>
       <b>Category:</b> <span id = "Employee"></span><br>
      </div>
      <script>
      if (window.XMLHttpRequest) {// code for IE7+, Firefox,
```

```
Chrome, Opera, Safari
            xmlhttp = new XMLHttpRequest();
          } else {// code for IE6, IE5
         xmlhttp = new ActiveXObject("Microsoft.XMLHTTP");
          }
          xmlhttp.open("GET","/dom/node.xml",false);
          xmlhttp.send();
          xmlDoc = xmlhttp.responseXML;

       document.getElementById("FirstName").innerHTML =
            xmlDoc.getElementsByTagName("FirstName")[0].
childNodes[0].nodeValue;
       document.getElementById("LastName").innerHTML =
            xmlDoc.getElementsByTagName("LastName")[0].
childNodes[0].nodeValue;
       document.getElementById("Employee").innerHTML =
            xmlDoc.getElementsByTagName("Employee")[0].
attributes[0].nodeValue;
      </script>
    </body>
  </html>
```

- In the above example, we are accessing the information of the nodes *FirstName*, *LastName* and *Employee*.
- *xmlDoc.getElementsByTagName("FirstName")[0].childNodes[0].nodeValue;* This line accesses the value for the child node *FirstName* using the getElementByTagName() method.
- *xmlDoc.getElementsByTagName("Employee")[0].attributes[0].nodeValue;* This line accesses the attribute value of the node *Employee* getElementByTagName() method.

KEY TERMS

Data
Application program interface (API)
Extensible Markup Language (XML)
Document Type Declaration (DTD)
Document Object Model (DOM)
Computer
Web page
Document entity
Service Modeling Language (SML)
Specific information

DISCUSSION QUESTIONS

1. What is XML? Explain.
2. Discuss the XML schemas.
3. What are the applications of XML?
4. Discuss about the document object model (DOM).
5. Explain simple API for XML.
6. Focus on traversing of DOM.
7. Discuss about navigation of DOM.
8. How to access of DOM?

REFERENCES

1. Adams N, Cannon E, Murray-Rust P: ChemAxiom – an ontological framework for chemistry in science. Nature Proceedings. 2009, 10.1038/npre.2009.3714.1.
2. Adams S, de Castro P, Echenique P, Estrada J, Hanwell MD, Murray-Rust P, Sherwood P, Thomas J, Townsend J: The Quixote project: Collaborative and Open Quantum Chemistry data managementin the Internet age. J Cheminfo. 2011, 3: 38-10.1186/1758-2946-3-38.
3. Hanwell MD, Curtis DE, Lonie DC, Vandermeersch T, Zurek E, Hutchison GR: Avogadro: An advanced semantic chemical editor, visualization, and analysisplatform. J Cheminfo. 2012, 4: 17-10.1186/1758-2946-4-17.
4. Hummelshoj F, Abild-Pedersen F, Studt F, Bligaard T, Norskov J: CatApp: A Web application for surface chemistry and heterogeneouscatalysis. Angew Chem Int Ed. 2012, 51: 272-274. 10.1002/anie.201107947.
5. Jain A, Hautier G, Moore C, Ong SP, Fischer C, Mueller T, Persson KA, Ceder G: A high-throughput infrastructure for density functional theorycalculations. Comp Mat Sci. 2011, 50: 2295-2310. 10.1016/j.commatsci.2011.02.023.
6. Kleese-van Dam K, James M, Walker AM: Integrating data management and collaborative sharing with computationalscience processes. Handbook of Research on Computational Science and Engineering: Theory andPractice Volume 1. Edited by: Leng J, Sharrok W. 2011, Hershey, Pennsylvania: IGI Global, 506-538.
7. Maddison DR, Swofford DL, Maddison WP: Nexus: an extensible file format for systematic information. Syst Biol. 1997, 46: 590-621. 10.1093/sysbio/46.4.590.
8. Marcial LH, Hemminger BM: Scientific data repositories on the Web: An initial survey. J Am Soc Inf Sci. 2010, 61: 2029-2048. 10.1002/asi.21339.
9. McMahon B: Applied and implied semantics in crystallographic publishing. J Cheminfo. 2012, 4: 19-10.1186/1758-2946-4-19.
10. Murray-Rust P, Hanwell MD, Hutchison GR, Neylon C, Spjuth O, Townsend J, Willighagen E, Walker AM: Building a CML code library. J Cheminfo. 2012, 4: 14-10.1186/1758-2946-4-14.
11. Murray-Rust P, Townsend J, Adams SE, Phadungsukanan W, Thomas J: The semantics of Chemical Markup Language (CML): dictionaries andconventions. J Cheminf. 2011, 3: 43-10.1186/1758-2946-3-43.
12. Murray-Rust P, Townsend JA, Adams SE, Phadungsukanan W, Thomas J: The semantics of Chemical Markup Language (CML): dictionaries andconventions. J Cheminfo. 2011, 3: 43-10.1186/1758-2946-3-43.

CHAPTER 4

CASCADING STYLE SHEETS (CSS)



LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Give the overview of CSS
2. Focus on inline styles
3. How to conflicting styles
4. Learn about linking external style sheets
5. Focus on positioning elements
6. Focus on backgrounds
7. Learn about element dimensions
8. Describe the text flow and the box model
9. Focus on user style sheets

INTRODUCTION

Cascading Style Sheets (CSS) is a style sheet language used for describing the presentation of a document written in a markup language such as HTML or XML (including XML dialects such as SVG, MathML or XHTML). CSS is a cornerstone technology of the World Wide Web, alongside HTML and JavaScript. CSS is designed to enable the separation of presentation and content, including layout, colors, and fonts. This separation can improve content accessibility; provide more flexibility and control in the specification of presentation characteristics; enable multiple web pages to share formatting by specifying the relevant CSS in a separate .css file, which reduces complexity and repetition in the structural content; and enable the .css file to be cached to improve the page load speed between the pages that share the file and its formatting. Separation of formatting and content also makes it feasible to present the same markup page in different styles for different rendering methods, such as on-screen, in print, by voice (via speech-based browser or screen reader), and on Braille-based tactile devices. CSS also has rules for alternate formatting if the content is accessed on a mobile device.



Cascading style sheets (CSS) is a standard defined by the World Wide Web Consortium that offers designers more flexibility and accuracy when defining the appearance of text and formats than standard HTML. Essentially, CSS allows designers to manipulate the appearance of the webpage without affecting its HTML structure. For example, if you wanted to change all the text in your document to blue, and all the headlines to green, with standard HTML one would have to manually change the elements on the page one by one. By using CSS, it's possible to redefine the body elements in the entire document to turn blue with just one instruction, and then perform another step to change the headlines to green.

OVERVIEW OF CSS

Cascading Style Sheets, fondly referred to as CSS, is a simple design language intended to simplify the process of making web pages presentable. CSS handles the look and feel part of a web page. Using CSS, you can control the color of the text, the style of fonts, the spacing between paragraphs, how columns are sized and laid out, what background images or colors are used, as well as a variety of other effects. CSS is easy to learn and understand but it provides a powerful control over the presentation of an HTML document. Most commonly, CSS is combined with the markup languages HTML or XHTML.



History of CSS

"Like HTML, CSS is standardized by the W3 consortium. CSS is an open, independent and freely usable standard. As with HTML, a working group exists for CSS who concerns itself with development of the standard. This is being done in accordance with W3C rules.

CSS comes in different versions. CSS level 1 was released in 1996, and republished with corections in 1999. CSS level 2 appeared in 1998 and built on CSS level 1. CSS level 2 adds support for different output media. CSS 3 is currently in development and will be split up in modules.

The idea behind it was to seperate presentation and content of a web site. The advantage of this approach is that updates and changes to the presentation need only be applied once (to the stylesheet) to affect all pages. Especially large sites can achieve significant time savings. In order to enforce this separation of

concerns, the W3C has deprecated the <font> tag in HTML 4. Newer (X)HTML versions will deprecate other tags in a similar vein.

CSS stylesheets are an immediate addition to HTML. CSS is a language to define format properties of HTML or XML tags. HTML has reached the end of it's shelf life and is not being developed further. XHTML is the new standard instead. XHTML reformulates HTML with XML syntax while at the same time getting rid of presentational markup. XHTML only contains content, presentation is the domain of a CSS stylesheet. This allows for different stylesheets for different output formats such as print, aural or small devices. Search machines on the other hand have an easier job if they are only presented with content markup."

Significance of Cascading Style Sheets

If used appropriately, cascading style sheets can be quite beneficial to the web designer. In addition to text, cascading style sheets can define spacing between lines, the size of the type in pixels instead of points, and define specific fonts within pages. Formatting each individual page becomes very cumbersome for the designer, especially if their site contains hundreds of pages. With style sheets, one only needs to specify such preferences once, and the style can be applied to an entire site. And if the designer decides to change the width of the page, then he or she only needs to change this preference in one place, rather than having to search through all of the pages to change the HTML.

Style sheets offer flexibility in terms of the presentation effects that they provide. Properties such as color, background, margin, border, and many more can be applied to all elements. With just HTML and its proprietary extensions, one must rely on attributes like BGCOLOR, which are only available for a few elements. Style sheets give the flexibility of applying a style to all paragraphs, or all level-two headings, or all emphasized text.

However, there is a down side to using CSS. Only Netscape 4.0+, 6.0+ and Explorer 4 and 5 support CSS. Some of these browsers don't offer full support for the entire CSS1 specification (Green, 2002). For instance, if someone were to look at stylebased pages in an older browser, they would not see any formatting

beyond default colors, sizes, fonts and positions. However, Dreamweaver is one of the few programs that can convert styles to HTML tags automatically

Advantages of CSS

- CSS saves time - You can write CSS once and then reuse the same sheet in multiple HTML pages. You can define a style for each HTML element and apply it to as many web pages as you want.
- Pages load faster - If you are using CSS, you do not need to write HTML tag attributes every time. Just write one CSS rule of a tag and apply it to all the occurrences of that tag. So, less code means faster download times.
- Easy maintenance - To make a global change, simply change the style, and all the elements in all the web pages will be updated automatically.
- Superior styles to HTML - CSS has a much wider array of attributes than HTML, so you can give a far better look to your HTML page in comparison to HTML attributes.
- Multiple Device Compatibility - Style sheets allow content to be optimized for more than one type of device. By using the same HTML document, different versions of a website can be presented for handheld devices such as PDAs and cellphones or for printing.
- Global web standards – Now HTML attributes are being deprecated and it is being recommended to use CSS. So it's a good idea to start using CSS in all the HTML pages to make them compatible with future browsers.

Who Creates and Maintains CSS?

CSS is created and maintained through a group of people within the W3C called the CSS Working Group. The CSS Working Group creates documents called specifications. When a specification has been discussed and officially ratified by the W3C members, it becomes a recommendation. These ratified specifications are called recommendations because the W3C has no control over the actual implementation of the language. Independent companies and organizations create that software.

INLINE STYLES

A Web developer can declare document styles in many ways. This method presents inline styles that declare an individual element's format using the XHTML attribute style. Figure 1 applies inline styles to p elements to alter their font size and color.

```
1 <?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5 <!-- Figure 1: inline.html -->
6 <!-- Using inline styles -->
7
8 <html xmlns = "http://www.w3.org/1999/xhtml">
9 <head>
10 <title>Inline Styles</title>
11 </head>
12
13 <body>
14
15 <p>This text does not have any style applied to it.</p>
16
17 <!-- The style attribute allows you to declare -->
18 <!-- inline styles. Separate multiple styles -->
19 <!-- with a semicolon.
20 <p style = "font-size: 20pt"> This text has the
21 <em>font-size</em> style applied to it, making it 20pt.
22 </p>
24 <p style = "font-size: 20pt; color: #0000ff">
25 This text has the <em>font-size</em> and
26 <em>color</em> styles applied to it, making it
27 20pt. and blue.</p>
28
29 </body>
30 </html>
```

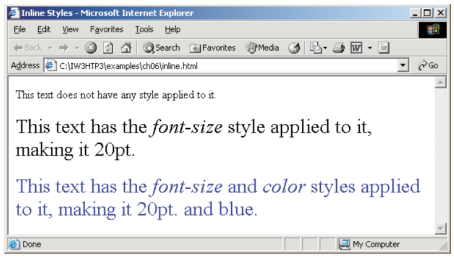


Figure 1: Inline styles.

The first **inline style** declaration appears in line 20. Attribute style specifies the style for an element. Each CSS property (the font-size property in this case) is followed by a colon and a value. In line 20, we declare this particular p element to use 20-point font size. Line 21 uses element em to "emphasize" text, which most browsers do by making the font italic.

Line 24 specifies the two properties, font-size and color, separated by a semicolon. In this line, we set the given paragraph's color to blue, using the hexadecimal code #0000ff. Color names may be used in place of hexadecimal codes.

Embedded Style Sheets

A second technique for using style sheets is embedded style sheets. Embedded style sheets enable a Web-page author to embed an entire CSS document in an XHTML document's head section. Figure 2 creates an embedded style sheet containing four styles.

```
1 <?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5 <!-- Figure 2: declared.html -->
6 <!-- Declaring a style sheet in the header section. -->
7
8 <html xmlns = "http://www.w3.org/1999/xhtml">
9 <head>
```

```
10 <title>Style Sheets</title>
11
12 <!-- this begins the style sheet section -->
13 <style type = "text/css">
14
15 em { background-color: #8000ff;
16 color: white }
17
18 h1 { font-family: arial, sans-serif }
19
20 p { font-size: 14pt }
21
22 .special { color: blue }
23
24 </style>
25 </head>
26
27 <body>
28
29 <!-- this class attribute applies the .special style -->
30 <h1 class = "special">Deitel & Associates, Inc.</h1>
31
32 <p>Deitel &amp;amp; Associates, Inc. is an internationally
33 recognized corporate training and publishing organization
34 specializing in programming languages, Internet/World
35 Wide Web technology and object technology education.
```

```
36 Deitel &amp;amp; Associates, Inc. is a member of the World
Wide
37 Web Consortium. The company provides courses on Java,
38 C++, Visual Basic, C, Internet and World Wide Web
39 programming, and Object Technology.</p>
40
41 <h1>Clients</h1>
42 <p class = “special”> The company’s clients include many
43 <em>Fortune 1000 companies</em>, government agencies,
44 branches of the military and business organizations.
45 Through its publishing partnership with Prentice Hall,
46 Deitel &amp;amp; Associates, Inc. publishes leading-edge
47 programming textbooks, professional books, interactive
48 CD-ROM-based multimedia Cyber Classrooms, satellite
49 courses and World Wide Web courses.</p>
50
51 </body>
52 </html>
```

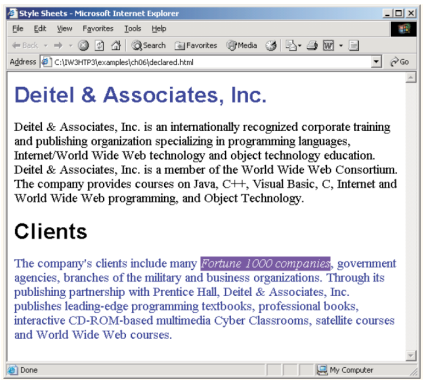


Figure 2: Embedded style sheets.

The style element (lines 13–24) defines the embedded style sheet. Styles placed in the head apply to matching elements wherever they appear in the entire document. The style element's type attribute specifies the Multipurpose Internet Mail Extensions (MIME) type that describes a file's content. CSS documents use the MIME type text/ css. Other MIME types include image/ gif (for GIF images) and text/javascript. The body of the style sheet (lines 15–22) declares the CSS rules for the style sheet. We declare rules for em (lines 15–16), h1 (line 18) and p (line 20) elements. When the browser renders this document, it applies the properties defined in these rules to every element to which the rule applies. For example, the rule in lines 15–16 will be applied to all em elements (in this example, there is one in line 43). The body of each rule is enclosed in curly braces ({and}).

Line 22 declares a style class named special. Style classes define styles that can be applied to any type of element. In this example, we declare class special, which sets color to blue. We can apply this style to elements of any type, whereas the other rules in this style sheet apply only to specific element types (i.e., em, h1 or p). Style class declarations are preceded by a period.

CSS rules in embedded style sheets use the same syntax as inline styles; the property name is followed by a colon (:) and the value of the property. Multiple properties are separated by semicolons (;). In the rule for em elements, the color property specifies the color of the text, and property background-color specifies the background color of the element.

The font-family property (line 18) specifies the name of the font to use. We use the arial font. The second value, sans-serif, is a generic font family. Not all users have the same fonts installed on their computers, so Web-page authors often specify a comma-separated list of fonts to use for a particular style. The browser attempts to use the fonts in the order they appear in the list. Many Web-page authors end a font list with a generic font family name in case the other fonts are not installed on the user's computer. In this example, if the arial font is not found on the system, the browser instead will display a generic sans-serif font, such as helvetica or verdana. Other generic font families include serif (e.g., times new roman, Georgia), cursive (e.g., script), fantasy (e.g., critter) and monospace (e.g., courier, fixedsys).

The font-size property (line 20) specifies a 14-point font. Relative values— xxsmall, x-small, small, smaller, medium, large, larger, x-large and xx-large— also can be used. Generally, relative values for font-size are preferred over point sizes because an author does not know the specific measurements of the display for each client. Relative font-size values permit moreflexible viewing of Web pages. For example, a user may wish to view a Web page on a handheld device with a small screen. Specifying an 18- point font size in a style sheet will prevent such a user from seeing more than one or two characters at a time. However, if a relative font size is specified, such as large or larger, the actual size is determined by the browser that displays the font. Using relative sizes also makes pages more accessible to users with disabilities. Users with impaired vision, for example, may configure their browser to use a larger default font, upon which all relative sizes are based. Text that the author specifies to be smaller than the main text still displays in a smaller size font, yet it is clearly visible to each user.

Line 30 uses attribute class in an h1 element to apply a style class—in this case class special (declared as .special in the style sheet). When the browser renders the h1 element, note that the text appears on screen with the properties of both an h1 element (arial or sans-serif font defined in line 18) and the .special style class applied (the color blue defined in line 22).

The formatting for the p element and the .special class are applied to the text in lines 42–49. All the styles applied to an element (the parent or ancestor element) also apply to the element's nested elements (child or descendant elements). The em element nested in the p element in line 43 inherits the style from the p element (namely, the 14-point font size in line 20), but retains its italic style. The em element has its own color property, so it overrides the color property of the special class.

CONFLICTING STYLES

Cascading style sheets are "cascading" because styles may be defined by a user, an author or a user agent (e.g., a Web browser). Styles "cascade," or flow together, such that the ultimate appearance of elements on a page results from combining styles defined in several ways. Styles defined by the user take

precedence over styles defined by the user agent, and styles defined by authors take precedence over styles defined by the user. Styles defined for parent elements are also inherited by child (nested) elements. We discuss the rules for resolving conflicts between styles defined for elements and styles inherited from parent and ancestor elements.

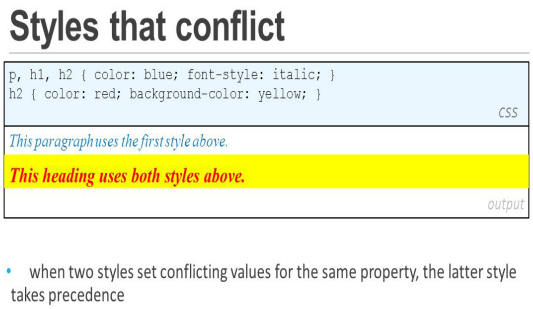


Figure 2 presented an example of inheritance in which a child em element inherited the font-size property from its parent p element. However, in Figure 2, the child em element had a color property that conflicted with (i.e., had a different value than) the color property of its parent p element. Properties defined for child and descendant elements have a greater specificity than properties defined for parent and ancestor elements.

```
1 <?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5 <!-- Figure 3: advanced.html -->
6 <!-- More advanced style sheets -->
7
8 <html xmlns = "http://www.w3.org/1999/xhtml">
9 <head>
10 <title>More Styles</title>
11
```

```
12 <style type = “text/css”>
13
14 a.nodec { text-decoration: none }
15
16 a:hover { text-decoration: underline;
17 color: red;
18 background-color: #ccffcc }
19
20 li em { color: red;
21 font-weight: bold }
22
23 ul { margin-left: 75px }
24
25 ul ul { text-decoration: underline;
26 margin-left: 15px }
27
28 </style>
29 </head>
30
31 <body>
32
33 <h1>Shopping list for <em>Monday</em>:</h1>
34
35 <ul>
36 <li>Milk</li>
37 <li>Bread
38 <ul>
```

```
39 <li>White bread</li>
40 <li>Rye bread</li>
41 <li>Whole wheat bread</li>
42 </ul>
43 </li>
44 <li>Rice</li>
45 <li>Potatoes</li>
46 <li>Pizza <em>with mushrooms</em></li>
47 </ul>
48
49 <p><a class = “nodec” href = “http://www.food.com”>
50 Go to the Grocery store</a></p>
51
52 </body>
53 </html>
```

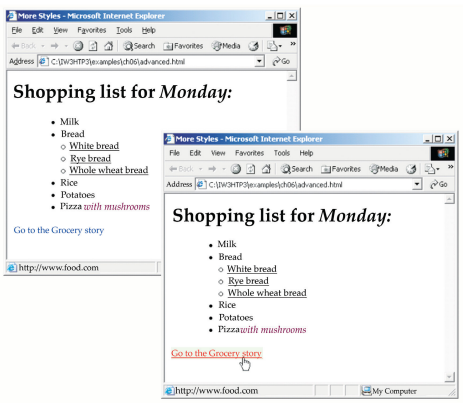


Figure 3: Inheritance in style sheets.

Line 14 applies property text-decoration to all an elements whose class attribute is set to nodec. The text-decoration property applies decorations to text within an element. By default, browsers underline the text of an a (anchor) element. Here, we set the

text-decoration property to none to indicate that the browser should not underline **hyperlink**. Other possible values for text-decoration include overline, linethrough, underline and blink. [Note: blink is not supported by Internet Explorer.] The .nodec appended to a is an extension of class styles; this style will apply only to an elements that specify nodec in their class attribute.

LINKING EXTERNAL STYLE SHEETS

Style sheets are a convenient way to create a document with a uniform theme. With external style sheets (i.e., separate documents that contain only CSS rules), Web-page authors can provide a uniform look and feel to an entire Web site. Different pages on a site can all use the same style sheet. When changes to the styles are required, the Web-page author needs to modify only a single CSS file to make style changes across the entire Web site. Figure 4 presents an external style sheet. Lines 1–2 are CSS comments. Like XHTML comments, CSS comments describe the content of a CSS document. Comments may be placed in any type of CSS code (i.e., inline styles, embedded style sheets and external style sheets) and always start with /* and end with */. Text between these delimiters is ignored by the browser.

```
1 /* Figure 4: styles.css */
2 /* An external stylesheet */
3
4 a { text-decoration: none }
5
6 a:hover { text-decoration: underline;
7 color: red;
8 background-color: #ccffcc }
9
10 li em { color: red;
11 font-weight: bold;
12 background-color: #ffffff }
```

```
13
14 ul { margin-left: 2cm }
15
16 ul ul { text-decoration: underline;
17 margin-left: .5cm }
```

Figure 4: External style sheet (styles.css).

Figure 5 contains an XHTML document that references the external style sheet in Figure 4. Lines 11–12 (Figure 5) show a link element that uses the rel attribute to specify a relationship between the current document and another document. We declare the linked document to be a stylesheet for this document. The type attribute specifies the MIME type as text/css. The href attribute provides the URL for the document containing the style sheet. Styles.css is in the same directory as external.html.

```
1 <?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5 <!-- Figure 5: external.html -->
6 <!-- Linking external style sheets -->
7
8 <html xmlns = "http://www.w3.org/1999/xhtml">
9 <head>
10 <title>Linking External Style Sheets</title>
11 <link rel = "stylesheet" type = "text/css"
12 href = "styles.css" />
13 </head>
14
15 <body>
```

```
16
17 <h1>Shopping list for <em>Monday</em>:</h1>
18 <ul>
19 <li>Milk</li>
20 <li>Bread
21 <ul>
22 <li>White bread</li>
23 <li>Rye bread</li>
24 <li>Whole wheat bread</li>
25 </ul>
26 </li>
27 <li>Rice</li>
28 <li>Potatoes</li>
29 <li>Pizza <em>with mushrooms</em></li>
30 </ul>
31
32 <p>
33 <a href = "http://www.food.com">Go to the Grocery store</a>
34 </p>
35
36 </body>
37 </html>
```

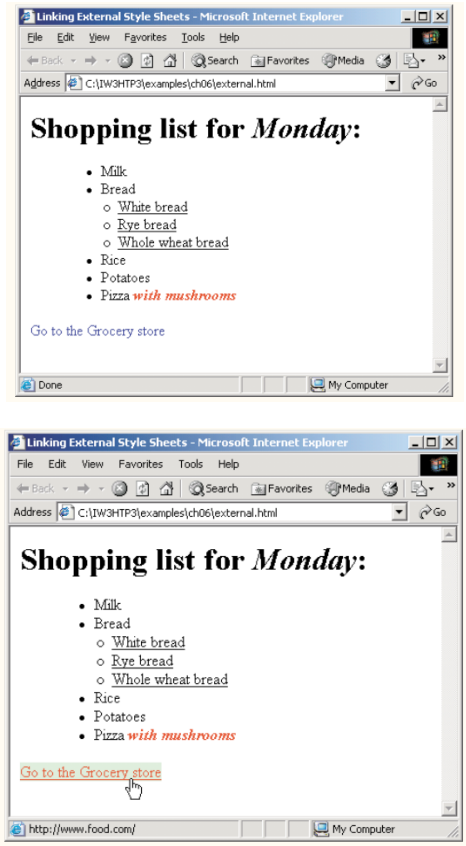


Figure 5: Linking an external style sheet.

W3C CSS Validation Service

The W3C provides a validation service (jigsaw.w3.org/css-validator) that validates external CSS documents to ensure that they conform to the W3C CSS Recommendation. Like XHTML validation, CSS validation ensures that style sheets are syntactically correct. The validator provides the option of either entering the CSS document's URL, pasting the CSS document's contents into a text area or uploading a CSS document.

Figure 6 illustrates uploading a CSS document, using the file upload feature available at jigsaw.w3.org/css-validator/validator-upload.html.

To validate the document, click the Browse… button to locate the file on your computer. Like many W3C Recommendations, the CSS Recommendation is being developed in stages (or versions). The current version under development is Version 3, so select

CSS version 3 in the Profile drop-down list. This field indicates to the validator the CSS Recommendation against which the uploaded file should be validated. Click Submit this CSS file for validation to upload the file for validation. Figure 7 shows the results of validating styles.css (Figure 4).

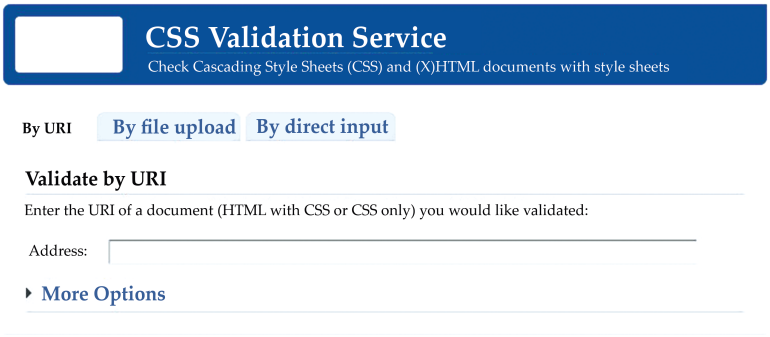


Figure 6: Validating a CSS document.

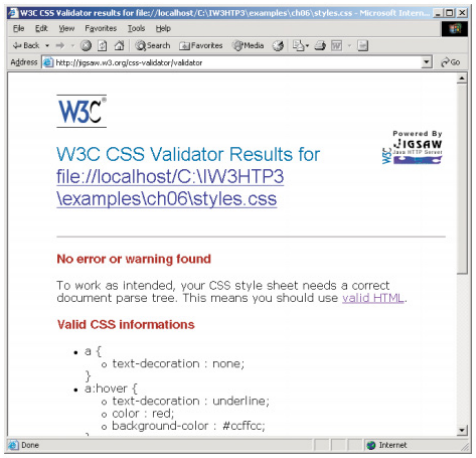


Figure 7: CSS validation results.

POSITIONING ELEMENTS

Before CSS, controlling the positioning of elements in an XHTML document was difficult—the browser determined positioning. CSS introduced the position property and a capability called absolute positioning, which gives authors greater control over how document elements are displayed. Figure 8 demonstrates absolute positioning.

```
1 <?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5 <!-- Figure 8: positioning.html -->
6 <!-- Absolute positioning of elements -->
7
8 <html xmlns = "http://www.w3.org/1999/xhtml">
9 <head>
10 <title>Absolute Positioning</title>
11 </head>
12
13 <body>
14
15 <p><img src = "i.gif" style = "position: absolute;
16 top: 0px; left: 0px; z-index: 1"
17 alt = "First positioned image" /></p>
18 <p style = "position: absolute; top: 50px; left: 50px;
19 z-index: 3; font-size: 20pt">Positioned Text</p>
20 <p><img src = "circle.gif" style = "position: absolute;
21 top: 25px; left: 100px; z-index: 2" alt =
22 "Second positioned image" /></p>
23
24 </body>
25 </html>
```

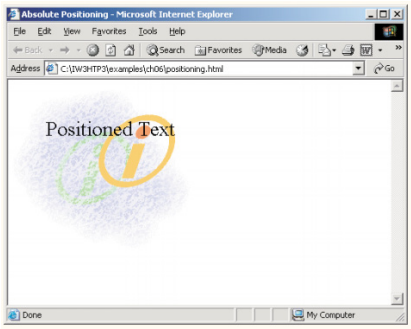


Figure 8: Absolute positioning of elements with CSS.

Lines 15–17 position the first img element (i.gif) on the page. Specifying an element's position as absolute removes the element from the normal flow of elements on the page, instead positioning it according to the distance from the top, left, right or bottom margins of its containing block-level element (i.e., an element such as body or p). Here, we position the element to be 0 pixels away from both the top and left margins of the p element (lines 15–17).

The z-index attribute allows you to layer overlapping elements properly. Elements that have higher z-index values are displayed in front of elements with lower z-index values. In this example, i.gif has the lowest z-index (1), so it displays in the background. The img element in lines 20–22 (circle.gif) has a z-index of 2, so it displays in front of i.gif. The p element in lines 18–19 (Positioned Text) has a z-index of 3, so it displays in front of the other two. If you do not specify a z-index or if elements have the same z-index value, the elements are placed from background to foreground in the order they are encountered in the document.

Absolute positioning is not the only way to specify page layout. Figure 9 demonstrates relative positioning, in which elements are positioned relative to other elements.

```
1 <?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5 <!-- Figure 9: positioning2.html -->
```

```
6 <!-- Relative positioning of elements -->
7
8 <html xmlns = “http://www.w3.org/1999/xhtml”>
9 <head>
10 <title>Relative Positioning</title>
11
12 <style type = “text/css”>
13
14 p { font-size: 1.3em;
15 font-family: verdana, arial, sans-serif }
16
17 span { color: red;
18 font-size: .6em;
19 height: 1em }
20
21 .super { position: relative;
22 top: -1ex }
23
24.sub { position: relative;
25 bottom: -1ex }
26
27 .shiftleft { position: relative;
28 left: -1ex }
29
30 .shiftright { position: relative;
31 right: -1ex }
32
```

```
33 </style>
34 </head>
35
36 <body>
37
38 <p>The text at the end of this sentence
39 <span class = “super”>is in superscript</span>. </p>
40
41 <p>The text at the end of this sentence
42 <span class = “sub”>is in subscript</span>. </p>
43
44 <p>The text at the end of this sentence
45 <span class = “shiftleft”>is shifted left</span>. </p>
46
47 <p>The text at the end of this sentence
48 <span class = “shiftright”>is shifted right</span>. </p>
49
50 </body>
51 </html>
```

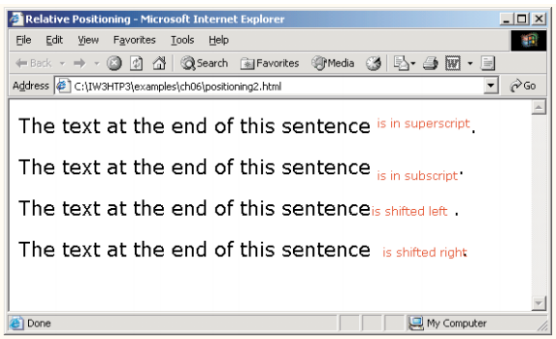


Figure 9: Relative positioning of elements with CSS.

Setting the position property to relative, as in class super (lines 21–22),lays out the element on the page and offsets it by the specified top, bottom, left or right value. Unlike absolute positioning, relative positioning keeps elements in the general flow of elements on the page, so positioning is relative to other elements in the flow. Recall that ex (line 22) is the x-height of a font, a relative length measurement typically equal to the height of a lowercase x.

We introduce the span element in line 39. Element span is a grouping element—it does not apply any inherent formatting to its contents. Its primary purpose is to apply CSS rules or id attributes to a block of text. Element span is an inline-level element—it is displayed inline with other text and with no line breaks. Lines 17–19 define the CSS rule for span. A similar element is the div element, which also applies no inherent styles but is displayed on its own line, with margins above and below (a block-level element).

BACKGROUNDS

CSS provides control over the element backgrounds. We introduced the background-color property. CSS also can add background images to documents. Figure 10 adds a corporate logo to the bottom-right corner of the document. This logo stays fixed in the corner even when the user scrolls up or down the screen.

```
1 <?xml version = “1.0”?>
2 <!DOCTYPE html PUBLIC “-//W3C//DTD XHTML 1.1//EN”
3 “http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd”>
4
5 <!—Figure 10: background.html -->
6 <!-- Adding background images and indentation -->
7
8 <html xmlns = “http://www.w3 .org/1999/xhtml”>
9 <head>
```

```
10 <title>Background Images</title>
11
12 <style type = “text/css”>
13
14 body { background-image: url(logo.gif);
15 background-position: bottom right;
16 background-repeat: no-repeat;
17 background-attachment: fixed; }
18
19 p { font-size: 18pt;
20 color: #aa5588;
21 text-indent: 1em;
22 font-family: arial, sans-serif; }
23
24 .dark { font-weight: bold }
25
26 </style>
27 </head>
28
29 <body>
30
31 <p>
32 This example uses the background-image,
33 background-position and background-attachment
34 styles to place the <span class = “dark”>Deitel
35 &amp;amp; Associates, Inc.</span> logo in the bottom,
36 right corner of the page. Notice how the logo
```

```
37 stays in the proper position when you resize the
38 browser window.
39 </p>
40
41 </body>
42 </html>
```

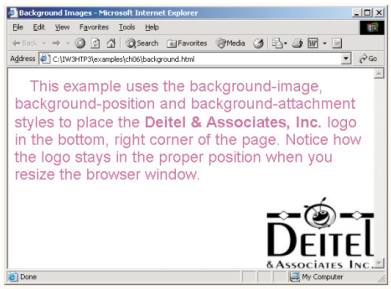


Figure 10: Background image added with CSS.

The background-image property (line 14) specifies the image URL for the image logo.gif in the format url(fileLocation). The Web-page author also can set the background-color property in case the image is not found. The background-position property (line 15) places the image on the page. The keywords top, bottom, center, left and right are used individually or in combination for vertical and horizontal positioning. An image can be positioned using lengths by specifying the horizontal length followed by the vertical length. For example, to position the image as horizontally centered (positioned at 50% of the distance across the screen) and 30 pixels from the top, use

```
background-position: 50% 30px;
```

The background-repeat property (line 16) controls the tiling of the background image. Tiling places multiple copies of the image next to each other to fill the background. Here, we set the tiling to no-repeat to display only one copy of the background image. The background-repeat property can be set to repeat (the default) to tile the image vertically and horizontally, repeat-x to tile the image only horizontally or repeat-y to tile the image only vertically.

The final property setting, background-attachment: fixed (line 17), fixes the image in the position specified by background-position. Scrolling the browser window will not move the image from its position. The default value, scroll, moves the image as the user scrolls through the document.

Line 21 indents the first line of text in the element by the specified amount, in this case 1em. An author might use this property to create a Web page that reads more like a novel, in which the first line of every paragraph is indented.

Line 24 uses the font-weight property to specify the "boldness" of text. Possible values are bold, normal (the default), bolder (bolder than bold text) and lighter (lighter than normal text). Boldness also can be specified with multiples of 100, from 100 to 900 (e.g., 100, 200, …, 900). Text specified as normal is equivalent to 400, and bold text is equivalent to 700. However, many systems do not have fonts that can scale with this level of precision, so using the values from 100 to 900 might not display the desired effect. Another CSS property that formats text is the font-style property, which allows the developer to set text to none, italic or oblique (oblique will default to italic if the system does not support oblique text).

ELEMENT DIMENSIONS

In addition to positioning elements, CSS rules can specify the actual dimensions of each page element. Figure 11 demonstrates how to set the dimensions of elements.

```
1 <?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5 <!-- Figure 11: width.html -->
6 <!-- Setting box dimensions and aligning text -->
7
8 <html xmlns = "http://www.w3.org/1999/xhtml">
```

```
9 <head>
10 <title>Box Dimensions</title>
11
12 <style type = “text/css”>
13
14 div { background-color: #ffccff;
15 margin-bottom: .5em }
16 </style>
17
18 </head>
19
20 <body>
21
22 <div style = “width: 20%”> Here is some
23 text that goes in a box which is
24 set to stretch across twenty percent
25 of the width of the screen.</div>
26
27 <div style = “width: 80%; text-align: center”>
28 Here is some CENTERED text that goes in a box
29 which is set to stretch across eighty percent of
30 the width of the screen.</div>
31
32 <div style = “width: 20%; height: 30%; overflow: scroll”>
33 This box is only twenty percent of
34 the width and thirty percent of the height.
35 What do we do if it overflows? Set the
```

```
36 overflow property to scroll!</div>
37
38 </body>
39 </html>
```

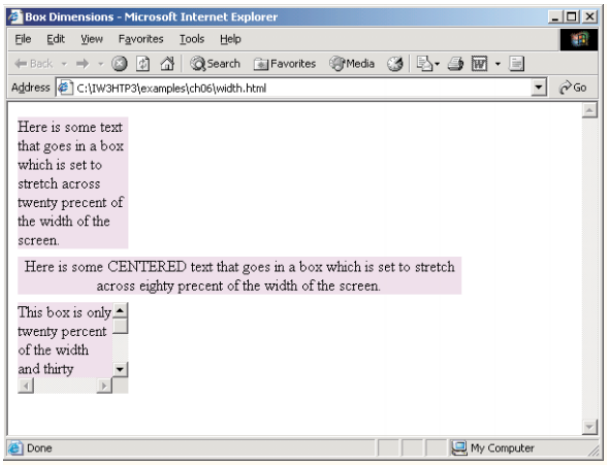


Figure 11: Element dimensions and text alignment.

The inline style in line 22 illustrates how to set the width of an element on screen; here, we indicate that the div element should occupy 20% of the screen width. Most elements are left-aligned by default; however, this alignment can be altered to position the element elsewhere. The height of an element can be set similarly, using the height property. The height and width values also can be specified as relative or absolute lengths. For example

```
width: 10em
```

sets the element's width to be equal to 10 times the font size. Line 27 sets text in the element to be center aligned; other values for the text-align property include left and right. One problem with setting both dimensions of an element is that the content inside the element can exceed the set boundaries, in which case the element is simply made large enough for all the content to fit. However, in line 32, we set the overflow property to scroll, a setting that adds scrollbars if the text overflows the boundaries.

TEXT FLOW AND THE BOX MODEL

A browser normally places text and elements on screen in the order in which they appear in the XHTML document. However, as we have seen with absolute positioning, it is possible to remove elements from the normal flow of text. Floating allows you to move an element to one side of the screen; other content in the document then flows around the floated element. In addition, each block-level element has a virtual box drawn around it, based on what is known as the box model. The properties of this box can be adjusted to control the amount of padding inside the element and the margins outside the element (Figure 12).

```
1 <?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5 <!-- Figure 12: floating.html -->
6 <!-- Floating elements and element boxes -->
7
8 <html xmlns = "http://www.w3.org/1999/xhtml">
9 <head>
10 <title>Flowing Text Around Floating Elements</title>
11
12 <style type = "text/css">
13
14 div { background-color: #ffccff;
15 margin-bottom: .5em;
16 font-size: 1.5em;
17 width: 50% }
18
19 p { text-align: justify }
```

```
20
21 </style>
22
23 </head>
24
25 <body>
26
27 <div style = “text-align: center”>
28 Deitel &amp;amp; Associates, Inc.</div>
29
30 <div style = “float: right; margin: .5em;
31 text-align: right”>
32 Corporate Training and Publishing</div>
33
34 <p>Deitel &amp;amp; Associates, Inc. is an internationally
35 recognized corporate training and publishing organization
36 specializing in programming languages, Internet/World
37 Wide Web technology and object technology education.
38 The company provides courses on Java, C++, Visual Basic,
C,
39 Internet and World Wide Web programming, and Object
Technology.</p>
40
41 <div style = “float: right; padding: .5em;
42 text-align: right”>
43 Leading-Edge Programming Textbooks</div>
44
```

```
45 <p>The company's clients include many Fortune 1000
46 companies, government agencies, branches of the military
47 and business organizations.</p>
48
49 <p style = "clear: right">Through its publishing
50 partnership with Prentice Hall, Deitel &amp;amp; Associates,
51 Inc. publishes leading-edge programming textbooks,
52 professional books, interactive CD-ROM-based multimedia
53 Cyber Classrooms, satellite courses and World Wide Web
54 courses.</p>
55
56 </body>
57 </html>
```

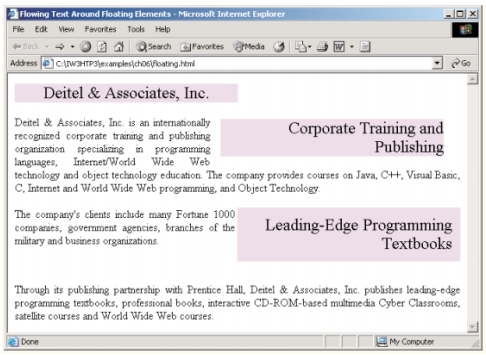


Figure 12: Floating elements, aligning text and setting box dimensions.

In addition to text, whole elements can be floated to the left or right of content. This means that any nearby text will wrap around the floated element. For example, in lines 30–32 we float a div element to the right side of the screen. As you can see from the sample screen capture, the text from lines 34–39 flows cleanly to the left and underneath the **div element**. The second property in line 30, margin, specifies the distance between the edge of the element and any other element on the

page. When the browser renders elements using the box model, the content of each element is surrounded by padding, a border and a margin (Figure 13).

Margins for individual sides of an element can be specified by using the properties margin-top, margin-right, margin-left and margin-bottom.

Lines 41–43 specify a div element that floats at the right side of the content. Property padding for the div element is set to .5em (half a line's height). Padding is the distance between the content inside an element and the element's border. Like the margin, the padding can be set for each side of the box, with padding-top, padding-right, padding-left and padding-bottom.

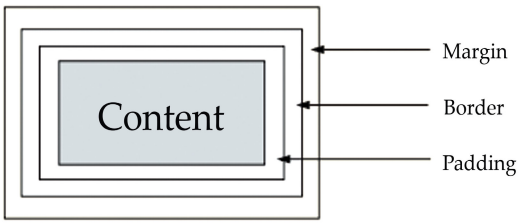


Figure 13: Box model for block-level elements.

Lines 49–54 show that you can interrupt the flow of text around a floated element by setting the clear property to the same direction as that in which the element is floated— right or left. Notice in the screen capture that the text from lines 45–47 flows to the left of the floated div element (lines 43–45), but the text from lines 49–54 does not. Setting the clear property to all interrupts the flow on both sides of the document.

Another property of every block-level element on screen is the border, which lies between the padding space and the margin space, and has numerous properties for adjusting its appearance, as shown in Figure 14.

```
1 <?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
```

```
5 <!-- Figure 14: borders.html -->
6 <!-- Setting borders of an element -->
7
8 <html xmlns = “http://www.w3.org/1999/xhtml”>
9 <head>
10 <title>Borders</title>
11
12 <style type = “text/css”>
13
14 body { background-color: #ccffcc }
15
16 div { text-align: center;
17 margin-bottom: 1em;
18 padding: .5em }
19
20 .thick { border-width: thick }
21
22 .medium { border-width: medium }
23
24.thin { border-width: thin }
25
26.groove { border-style: groove }
27
28 .inset { border-style: inset }
29
30 .outset { border-style: outset }
31
```

```
32 .red { border-color: red }
33
34 .blue { border-color: blue }
35
36 </style>
37 </head>
38
39 <body>
40
41 <div class = “thick groove”>This text has a border</div>
42 <div class = “medium groove”>This text has a border</div>
43 <div class = “thin groove”>This text has a border</div>
44
45 <p class = “thin red inset”>A thin red line...</p>
46 <p class = “medium blue outset”>
47 And a thicker blue line</p>
48
49 </body>
50 </html>
```

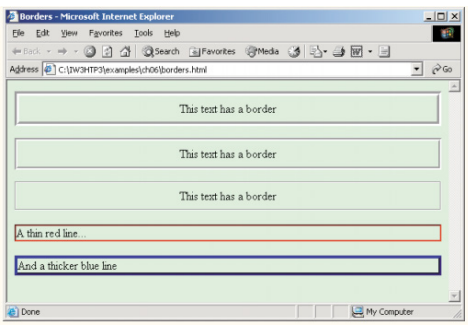


Figure 14: Borders of block-level elements.

We set three properties—border-width, border-color and border-style. The border-width property may be set to any valid CSS length or to the predefined value of thin, medium or thick. The border-color property sets the color.

As with padding and margins, each of the border properties may be set for an individual side of the box (e.g., border-top-style or border-left-color). A developer can assign more than one class to an XHTML element by using the class attribute, as shown in line 41.

The border-styles are none, hidden, dotted, dashed, solid, double, groove, ridge, inset and outset. Borders groove and ridge have opposite effects, as do inset and outset. Figure 15 illustrates these border styles.

```
1 <?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5 <!-- Figure 15: borders2.html -->
6 <!-- Various border-styles -->
7
8 <html xmlns = "http://www.w3.org/1999/xhtml">
9 <head>
10 <title>Borders</title>
11
12 <style type = "text/css">
13
14 body { background-color: #ccffcc }
15
16 div { text-align: center;
17 margin-bottom: .3em;
```

```
18 width: 50%;
19 position: relative;
20 left: 25%;
21 padding: .3em }
22 </style>
23 </head>
24
25 <body>
26
27 <div style = “border-style: solid”>Solid border</div>
28 <div style = “border-style: double”>Double border</div>
29 <div style = “border-style: groove”>Groove border</div>
30 <div style = “border-style: ridge”>Ridge border</div>
31 <div style = “border-style: inset”>Inset border</div>
32 <div style = “border-style: outset”>Outset border</div>
33
34 </body>
35 </html>
```

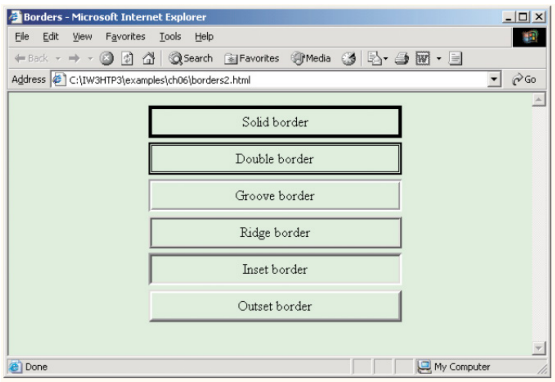


Figure 15: border-style property of the box model.

USER STYLE SHEETS

Users can define their own user style sheets to format pages based on their preferences. For example, people with visual impairments may want to increase the page's text size. Webpage authors need to be careful not to inadvertently override user preferences with defined styles.

Figure 16 contains an author style. The font-size is set to 9pt for all tags that have class note applied to them.

```
1 <?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5 <!—Figure 16: user_absolute.html -->
6 <!-- User styles -->
7
8 <html xmlns = "http://www.w3.org/1999/xhtml">
9 <head>
10 <title>User Styles</title>
11
12 <style type = "text/css">
13
14 .note { font-size: 9pt }
15
16 </style>
17 </head>
18
19 <body>
20
```

```
21 <p>Thanks for visiting my Web site. I hope you enjoy it.
22 </p><p class = “note”>Please Note: This site will be
23 moving soon. Please check periodically for updates.</p>
24
25 </body>
26 </html>
```

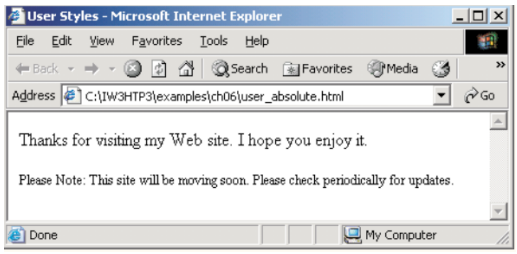


Figure 16: pt measurement for text size.

User style sheets are external style sheets. Figure 17 shows a user style sheet that sets the body's font-size to 20pt, color to yellow and background-color to #000080.

```
1 /* Figure 17: userstyles.css */
2 /* A user stylesheet */
3
4 body { font-size: 20pt;
5 color: yellow;
6 background-color: #000080 }
```

Figure 17: User style sheet.

User style sheets are not linked to a document; rather, they are set in the browser's options. To add a user style sheet in Internet Explorer 6, select Internet Options..., located in the Tools menu. In the Internet Options dialog (Figure 18) that appears, click Accessibility..., check the Format documents using my style sheet checkbox, and type the location of the user style sheet. Internet Explorer 6 applies the user style sheet to any document it loads.

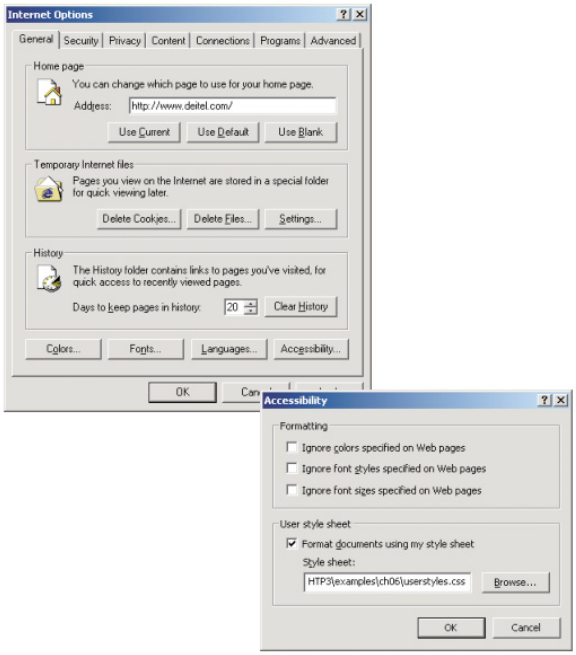


Figure 18: User style sheet in Internet Explorer 6.

The Web page from Figure 16 is displayed in Figure 19, with the user style sheet from Figure 17 applied.

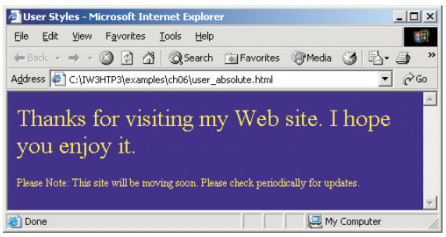


Figure 19: User style sheet applied with pt measurement.

In this example, if users define their own font-size in a user style sheet, the author style has a higher precedence and overrides the user style. The 9pt font specified in the author style sheet overrides the 20pt font specified in the user style sheet. This small font may make pages difficult to read, especially for individuals with visual impairments. A developer can avoid this problem by using relative measurements (e.g., em or ex) instead of absolute measurements, such as pt. Figure 20 changes the font-size property to use a relative measurement (line 14) that does not override the user style set in Figure 17. Instead, the

font size displayed is relative to the one specified in the user style sheet. Text enclosed in the tag displays as 20pt ,and tags that have class note applied to them are displayed in 15pt (.75 times 20pt).

```
1 <?xml version = "1.0"?>
2 <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5 <!-- Figure 20: user_relative.html -->
6 <!-- User styles -->
7
8 <html xmlns = "http://www.w3.org/1999/xhtml">
9 <head>
10 <title>User Styles</title>
11
12 <style type = "text/css">
13
14 .note { font-size: .75em }
15
16 </style>
17 </head>
18
19 <body>
20
21 <p>Thanks for visiting my Web site. I hope you enjoy it.
22 </p><p class = "note">Please Note: This site will be
23 moving soon. Please check periodically for updates.</p>
```

```
24
25 </body>
26 </html>
```

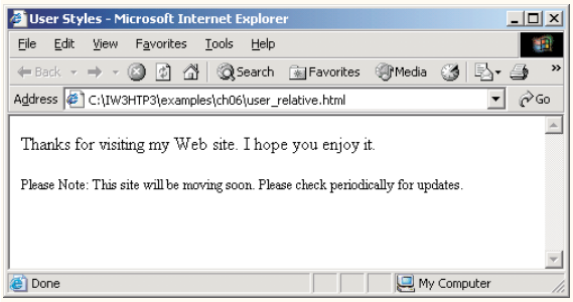


Figure 20: em measurement for text size.

Figure 21 displays the Web page from Figure 20 with the user style sheet from Figure 16 applied. Note that the second line of text displayed is larger than the same line of text in Figure 19.

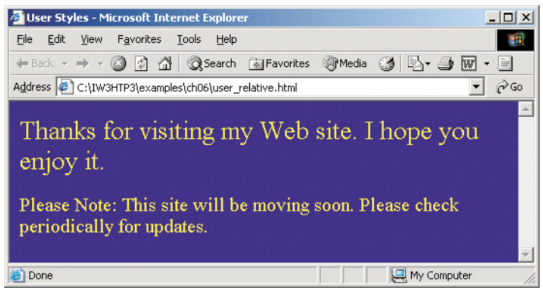


Figure 21: User style sheet applied with em measurement.

KEY TERMS

Inline style
External style
div element
Stylesheets
Absolute positioning
Hyperlink
Padding
Complexity
Multipurpose Internet Mail Extensions (MIME)

DISCUSSION QUESTIONS

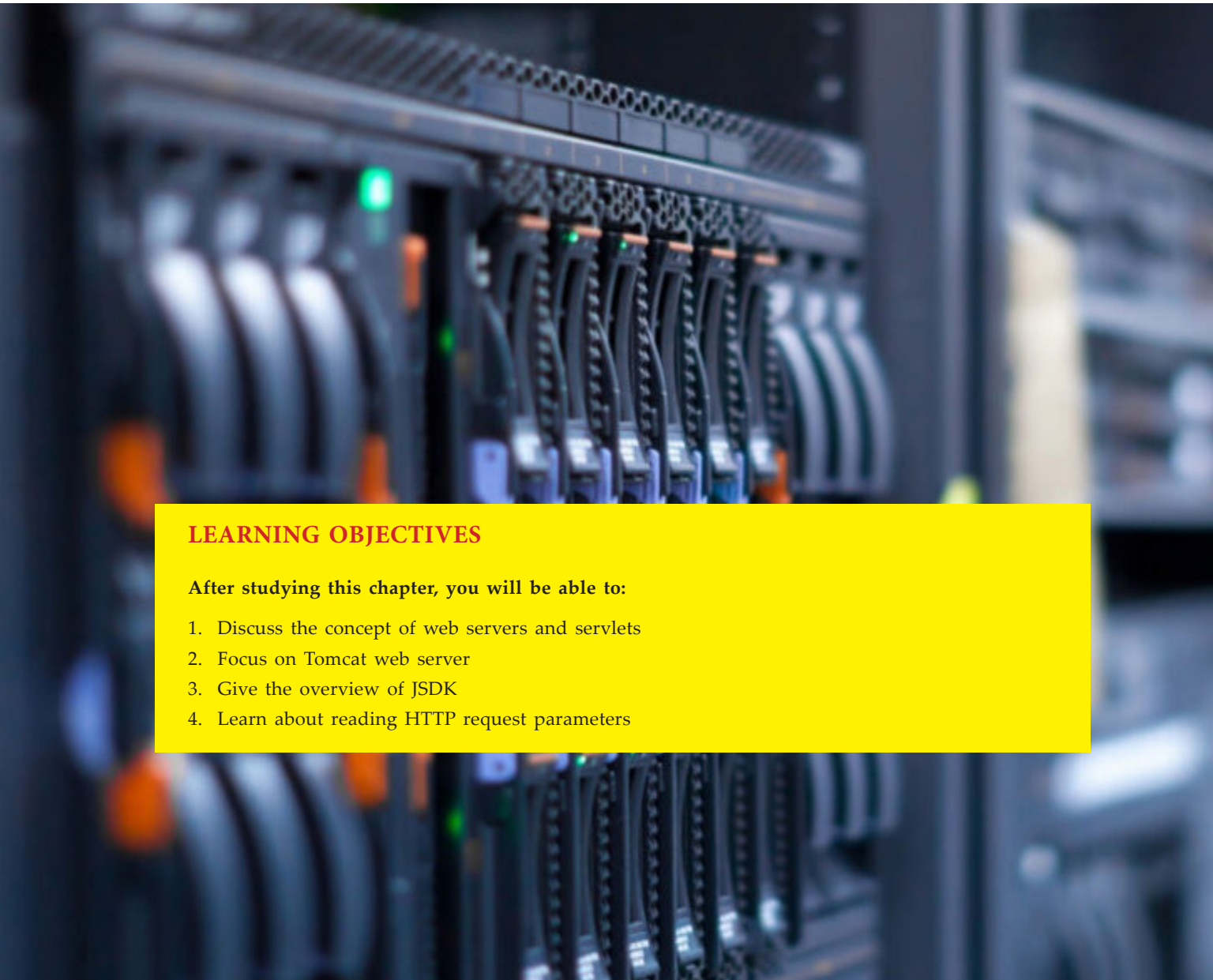
1. Explain the significance of cascading style sheets.
2. Discuss about history of CSS.
3. Define the inline styles.
4. Describe the embedded style sheets.
5. What is the W3C CSS validation service?
6. Write short notes on text flow and the box model.

REFERENCES

1. Anderson, Kareem (13 September 2017). "Microsoft's newest browser gets a significant boost with EdgeHTML 16". Retrieved 7 October 2017.
2. Atkins Jr., Tab; J. Etemad, Elika; Atanassov, Rossen; Brufau, Oriol; Mogilevsky, Alex; Cupp, Phil; Mielke, Markus, eds. (2021-03-11). "CSS Grid Layout Module Level 2". W3C. CSS Working Group. Retrieved 2021-04-09.
3. Bos, Bert (26 February 2011). "CSS current work". World Wide Web Consortium. Retrieved 3 March 2011.
4. Bos, Bert; et al. (7 December 2010). "9.3 positioning schemes". Cascading Style Sheets Level 2 Revision 1 (CSS 2.1) Specification. W3C. Retrieved 16 February 2011.
5. Cascading Style Sheets-Wikipedia, the free encyclopedia. (). In . Retrieved Sep. 26, 2011, from Web site: http://en.wikipedia.org/wiki/Cascading_Style_Sheets#History
6. Cederholm, Dan; Ethan Marcotte (2009). Handcrafted CSS: More Bulletproof Web Design. New Riders. p. 114. Retrieved 19 June 2010.
7. Cederholm, Dan; Ethan Marcotte (2009). *Handcrafted CSS: More Bulletproof Web Design*. New Riders. p. 114. Retrieved 19 June 2010.
8. CSS (Cascading Style Sheet) Definition. Retrieved Sep. 26, 2011, from Web site: http://www.techterms.com/definition/css
9. CSS Grid – Table layout is back. Be there and be square. Google Developers. January 2017. Retrieved 2021-04-09.
10. CSS3 Solutions for Internet Explorer – Smashing Magazine". Smashing Magazine. 2010-04-28. Retrieved 2016-10-12.
11. Goetter, Raphael (2017-02-16). "Flexbox grids and frameworks". GitHub Gist. Archived from the original on 2017-02-16. Retrieved 2021-04-09.
12. Holzschlag, Molly E (2005). Spring into HTML and CSS. Pearson Education, Inc. ISBN 0-13-185586-7.
13. HTML - A Word Definition From the Webopedia Computer Dictionary. Retrieved Sep. 26, 2011, from Web site: http://www.webopedia.com/TERM/H/HTML.html
14. Lou Montulli; Brendan Eich; Scott Furman; Donna Converse; Troy Chevalier (22 August 1996). "JavaScript-Based Style Sheets". World Wide Web Consortium. Retrieved 23 June 2010.
15. Marcotte, Ethan (2018-07-18). "Fractional. — Ethan Marcotte". Ethan Marcotte. Retrieved 2021-04-09.

CHAPTER 5

WEB SERVERS AND SERVLETS



LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Discuss the concept of web servers and servlets
2. Focus on Tomcat web server
3. Give the overview of JSDK
4. Learn about reading HTTP request parameters

INTRODUCTION

A web server is computer software and underlying hardware that accepts requests via HTTP (the network protocol created to distribute web content) or its secure variant HTTPS. A user agent, commonly a web browser or web crawler, initiates communication by making a request for a web page or other resource using HTTP, and the server responds with the content of that resource or an error message. A web server can also accept and store resources sent from the user agent if configured to do so. The hardware used to run a web server can vary according to the volume of requests that it needs to handle. At the low end of the range are embedded systems, such as a router that runs a small web server as its configuration interface. A high-traffic Internet website might handle requests with hundreds of servers that run on racks of high-speed computers. A web server is a computer system that processes requests via HTTP, the basic network protocol used to distribute information on the World Wide Web. The term can refer to the entire system, or specifically to the software that accepts and supervises the HTTP requests. A web server is a computer that stores websites on the Internet and delivers web pages to viewers upon request. This service is referred to as web hosting. Every web server has a unique address, called an Internet Protocol address, that tells other computers connected to the Internet where to find the server on the vast network.

Servlets provide a component-based, platform-independent method for building Web-based applications, without the performance limitations of CGI programs. Servlets have access to the entire family of Java APIs, including the JDBC API to access enterprise databases.

A servlet is a Java programming language class that is used to extend the capabilities of servers that host applications accessed by means of a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by web servers. For such applications, Java Servlet technology defines HTTP-specific servlet classes. The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing servlets. All servlets must implement the Servlet interface, which defines life-cycle methods. When implementing a generic service, you can use or

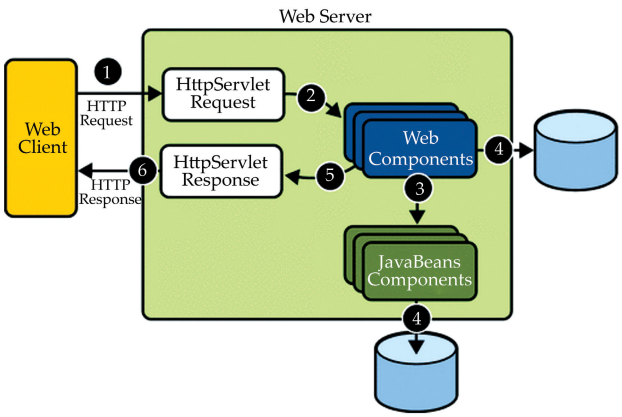
extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet and doPost, for handling HTTP-specific services.

CONCEPT OF WEB SERVERS AND SERVLETS

A Web server is a program that uses HTTP (Hypertext Transfer Protocol) to serve the files that form Web pages to users, in response to their requests, which are forwarded by their computers' HTTP clients. Dedicated computers and appliances may be referred to as Web servers as well.

The process is an example of the client/server model. All computers that host Web sites must have Web server programs. Leading Web servers include Apache (the most widely-installed Web server), Microsoft's Internet Information Server (IIS) and nginx (pronounced *engine X*) from NGNIX. Other Web servers include Novell's NetWare server, Google Web Server (GWS) and IBM's family of Domino servers

Web servers often come as part of a larger package of Internet- and intranet-related programs for serving email, downloading requests for File Transfer Protocol (FTP) files, and building and publishing Web pages. Considerations in choosing a Web server include how well it works with the operating system and other servers, its ability to handle server-side programming, security characteristics.

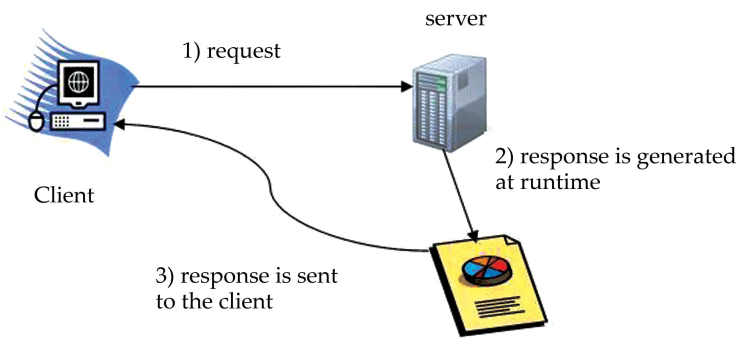


Servlet technology is robust and scalable because of java language. Before Servlet, CGI (Common Gateway Interface) scripting language was popular as a server-side programming language. But there was many disadvantages of this technology. We have discussed these disadvantages below.

There are many interfaces and classes in the servlet API such as Servlet, GenericServlet, HttpServlet, ServletRequest, ServletResponse etc.

Servlet can be described in many ways, depending on the context.

- Servlet is a technology i.e. used to create web application.
- Servlet is an API that provides many interfaces and classes including documentations.
- Servlet is an interface that must be implemented for creating any servlet.
- Servlet is a class that extend the capabilities of the servers and respond to the incoming request. It can respond to any type of requests.
- Servlet is a web component that is deployed on the server to create dynamic web page.

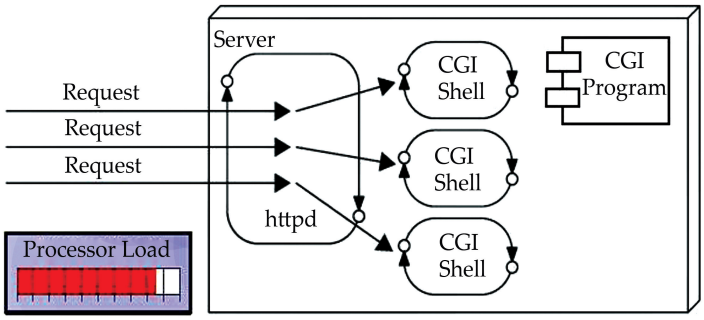


Web application

A web application is an application accessible from the web. A web application is composed of web components like Servlet, JSP, Filter etc. and other components such as HTML. The web components typically execute in Web Server and respond to HTTP request.

CGI (Common Gateway Interface)

CGI technology enables the web server to call an external program and pass HTTP request information to the external program to process the request. For each request, it starts a new process.



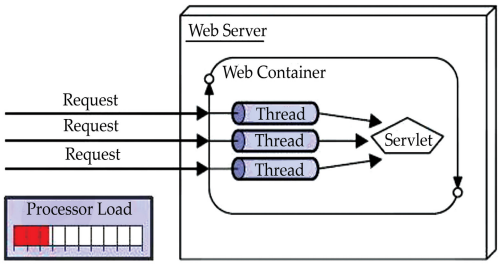
Disadvantages of CGI

There are many problems in CGI technology:

1. If number of clients increases, it takes more time for sending response.
2. For each request, it starts a process and Web server is limited to start processes.
3. It uses platform dependent language e.g. C, C++, perl.

Advantage of Servlet

There are many advantages of servlet over CGI. The web container creates threads for handling the multiple requests to the servlet. Threads have a lot of benefits over the Processes such as they share a common memory area, lightweight, cost of communication between the threads are low.



The basic benefits of servlet are as follows:

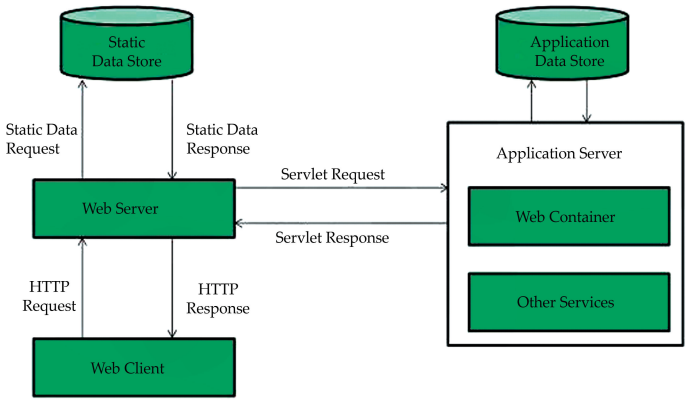
1. *Better performance*: because it creates a thread for each request not process.
2. *Portability*: because it uses java language.
3. *Robust*: Servlets are managed by JVM so we do not need to worry about memory leak, garbage collection etc.
4. *Secure*: because it uses java language.

Web Server Working

Web server is a computer where the web content is stored. Basically web server is used to host the web sites but there exists other web servers also such as gaming, storage, FTP, email etc.

Web server respond to the client request in either of the following two ways:

- Sending the file to the client associated with the requested URL.
- Generating response by invoking a script and communicating with database



Key Points

- When client sends request for a web page, the web server search for the requested page if requested page is found then it will send it to client with an HTTP response.
- If the requested web page is not found, web server will the send an HTTP response: Error 404 Not found.

- If client has requested for some other resources then the web server will contact to the application server and data store to construct the HTTP response.

Architecture

Web Server Architecture follows the following two approaches:

1. Concurrent Approach
2. Single-Process-Event-Driven Approach.

Concurrent Approach

Concurrent approach allows the web server to handle multiple client requests at the same time. It can be achieved by following methods:

- Multi-process
- Multi-threaded
- Hybrid method.

Multi-processing

In this a single process (parent process) initiates several single-threaded child processes and distribute incoming requests to these child processes. Each of the child processes are responsible for handling single request. It is the responsibility of parent process to monitor the load and decide if processes should be killed or forked.

Multi-threaded

Unlike Multi-process, it creates multiple single-threaded process.

Hybrid

It is combination of above two approaches. In this approach multiple process are created and each process initiates multiple threads. Each of the threads handles one connection. Using multiple threads in single process results in less load on system resources.

Examples

Following table describes the most leading web servers available today:

S.N.	Web Server Description
1	**Apache HTTP Server** This is the most popular web server in the world developed by the Apache Software Foundation. Apache web server is an open source software and can be installed on almost all operating systems including Linux, UNIX, Windows, FreeBSD, Mac OS X and more. About 60% of the web server machines run the Apache Web Server.
2.	**Internet Information Services (IIS)** The Internet Information Server (IIS) is a high performance Web Server from Microsoft. This web server runs on Windows NT/2000 and 2003 platforms (and may be on upcoming new Windows version also). IIS comes bundled with Windows NT/2000 and 2003; Because IIS is tightly integrated with the operating system so it is relatively easy to administer it.
3.	**Lighttpd** The lighttpd, pronounced lighty is also a free web server that is distributed with the FreeBSD operating system. This open source web server is fast, secure and consumes much less CPU power. Lighttpd can also run on Windows, Mac OS X, Linux and Solaris operating systems.
4.	**Sun Java System Web Server** This web server from Sun Microsystems is suited for medium and large web sites. Though the server is free it is not open source. It however, runs on Windows, Linux and UNIX platforms. The Sun Java System web server supports various languages, scripts and technologies required for Web 2.0 such as JSP, Java Servlets, PHP, Perl, Python, and Ruby on Rails, ASP and Coldfusion etc.
5.	**Jigsaw Server** Jigsaw (W3C's Server) comes from the World Wide Web Consortium. It is open source and free and can run on various platforms like Linux, UNIX, Windows, and Mac OS X Free BSD etc. Jigsaw has been written in Java and can run CGI scripts and PHP programs.

TOMCAT WEB SERVER

Tomcat is an application server from the Apache Software Foundation that executes Java servlets and renders Web pages that include Java Server Page coding. Described as a "reference implementation" of the Java Servlet and the Java Server Page specifications, Tomcat is the result of an open collaboration of developers and is available from the Apache Web site in both binary and source versions. Tomcat can be used as either a standalone product with its own internal Web server or together with other Web servers, including Apache, Netscape Enterprise Server, Microsoft Internet Information Server (IIS), and Microsoft Personal Web Server.

Apache Tomcat is an open-source Java Servlet Container developed by the Apache Software Foundation (ASF). Tomcat implements several Java EE specifications including

Java Servlet, Java Server Pages (JSP), Java EL, and WebSocket, and provides a "pure Java" HTTP web **server** environment in which Java code can run. Tomcat is developed and maintained by an open community of developers under the auspices of the Apache Software Foundation, released under the Apache License 2.0 license, and is open-source software.

Servelets

Java Servlets are programs that run on a Web or Application server and act as a middle layer between a requests coming from a Web browser or other HTTP client and databases or applications on the HTTP server.

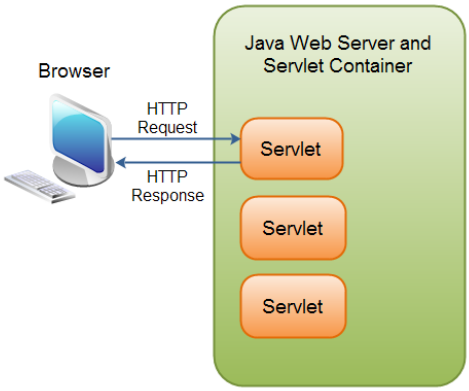
Using Servlets, you can collect input from users through web page forms, present records from a database or another source, and create web pages dynamically.

Java Servlets often serve the same purpose as programs implemented using the Common Gateway Interface (CGI).

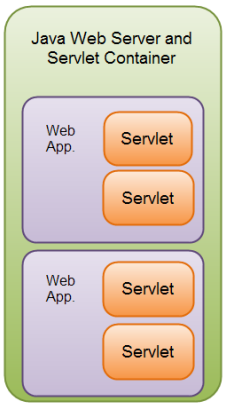
But Servlets offer several advantages in comparison with the CGI.

- Performance is significantly better.
- Servlets execute within the address space of a Web server. It is not necessary to create a separate process to handle each client request.
- Servlets are platform-independent because they are written in Java.
- Java security manager on the server enforces a set of restrictions to protect the resources on a server machine. So servlets are trusted.
- The full functionality of the Java class libraries is available to a servlet. It can communicate with applets, databases, or other software via the sockets and RMI mechanisms that you have seen already.

A Java Servlet is a Java object that responds to HTTP requests. It runs inside a Servlet container. Here is an illustration of that:



A Servlet is part of a Java web application. A Servlet container may run multiple web applications at the same time, each having multiple servlets running inside. Here is an illustration of that:



A Java web application can contain other components than servlets. It can also contain Java Server Pages (JSP), Java Server Faces (JSF) and Web Services. This tutorial is about Java Servlets only, though.

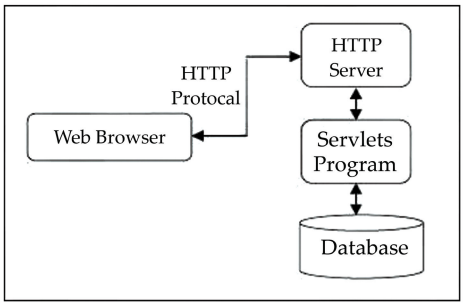
HTTP Request and Response

The browser sends an HTTP request to the Java web server. The **web server** checks if the request is for a servlet. If it is, the servlet container is passed the request. The servlet container will then find out which servlet the request is for, and activate that servlet. The servlet is activated by calling the Servlet.service () method.

Once the servlet has been activated via the service () method, the servlet processes the request, and generates a response. The response is then sent back to the browser.

Servlets Architecture

The following diagram shows the position of Servlets in a Web Application.



Servlets Tasks

Servlets perform the following major tasks –

- Read the explicit data sent by the clients (browsers). This includes an HTML form on a Web page or it could also come from an applet or a custom HTTP client program.
- Read the implicit HTTP request data sent by the clients (browsers). This includes cookies, media types and compression schemes the browser understands, and so forth.
- Process the data and generate the results. This process may require talking to a database, executing an RMI or CORBA call, invoking a Web service, or computing the response directly.
- Send the explicit data (i.e., the document) to the clients (browsers). This document can be sent in a variety of formats, including text (HTML or XML), binary (GIF images), Excel, etc.
- Send the implicit HTTP response to the clients (browsers). This includes telling the browsers or other clients what type of document is being returned (e.g., HTML), setting cookies and caching parameters, and other such tasks.

Servlets Packages

Java Servlets are Java classes run by a web server that has an interpreter that supports the Java Servlet specification.

Servlets can be created using the javax.servlet and javax.servlet.http packages, which are a standard part of the Java's enterprise edition, an expanded version of the Java class library that supports large-scale development projects.

These classes implement the Java Servlet and JSP specifications. At the time of writing this tutorial, the versions are Java Servlet 2.5 and JSP 2.1.

Java servlets have been created and compiled just like any other Java class. After you install the servlet packages and add them to your computer's Classpath, you can compile servlets with the JDK's Java compiler or any other current compiler.

What is Next?

You would take you step by step to set up your environment to start with Servlets. So fasten your belt for a nice drive with Servlets.

Servlets-Environment Setup

A development environment is where you would develop your Servlet, test them and finally run them.

Like any other Java program, you need to compile a servlet by using the Java compiler **javac** and after compilation the servlet application, it would be deployed in a configured environment to test and run.

This development environment setup involves the following steps –

Setting up Java Development Kit

This step involves downloading an implementation of the Java Software Development Kit (SDK) and setting up PATH environment variable appropriately.

Once you download your Java implementation, follow the given instructions to install and configure the setup. Finally set PATH and JAVA_HOME environment variables to refer to the directory that contains java and javac, typically java_install_dir/bin and java_install_dir respectively.

If you are running Windows and installed the SDK in C:\jdk1.8.0_65, you would put the following line in your C:\autoexec.bat file.

set PATH = C:\jdk1.8.0_65\bin;%PATH%

set JAVA_HOME = C:\jdk1.8.0_65

Alternatively, on Windows NT/2000/XP, you could also right-click on My Computer, select Properties, then Advanced, then Environment Variables. Then, you would update the PATH value and press the OK button.

On Unix (Solaris, Linux, etc.), if the SDK is installed in /usr/local/jdk1.8.0_65 and you use the C shell, you would put the following into your .cshrc file.

setenv PATH /usr/local/jdk1.8.0_65/bin:$PATH

setenv JAVA_HOME /usr/local/jdk1.8.0_65

Alternatively, if you use an Integrated Development Environment (IDE) like Borland JBuilder, Eclipse, IntelliJ IDEA, or Sun ONE Studio, compile and run a simple program to confirm that the IDE knows where you installed Java.

Setting up Web Server – Tomcat

A number of Web Servers that support servlets are available in the market. Some web servers are freely downloadable and Tomcat is one of them. Apache Tomcat is an open source software implementation of the Java Servlet and Java Server Pages technologies and can act as a standalone server for testing servlets and can be integrated with the Apache Web Server.

Here are the steps to setup tomcat on your machine –

- Once you downloaded the installation, unpack the binary distribution into a convenient location. For example in C:\apache-tomcat-8.0.28 on windows, or /

usr/local/apache-tomcat-8.0.289 on Linux/Unix and create CATALINA_HOME environment variable pointing to these locations.

Tomcat can be started by executing the following commands on windows machine −

%CATALINA_HOME%\bin\startup.bat

or

C:\apache-tomcat-8.0.28\bin\startup.bat

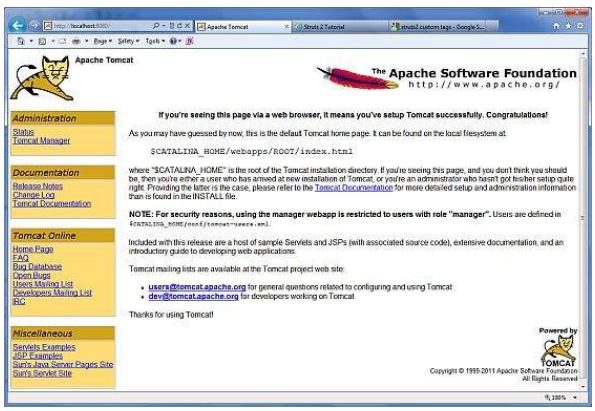
Tomcat can be started by executing the following commands on Unix (Solaris, Linux, etc.) machine −

$CATALINA_HOME/bin/startup.sh

or

/usr/local/apache-tomcat-8.0.28/bin/startup.sh

After startup, the default web applications included with Tomcat will be available by visiting http://localhost:8080/. If everything is fine then it should display following result −



Tomcat can be stopped by executing the following commands on windows machine −

C:\apache-tomcat-8.0.28\bin\shutdown

Tomcat can be stopped by executing the following commands on Unix (Solaris, Linux, etc.) machine

/usr/local/apache-tomcat-8.0.28/bin/shutdown.sh

Setting Up the CLASSPATH

Since servlets are not part of the Java Platform, Standard Edition, you must identify the servlet classes to the compiler.

If you are running Windows, you need to put the following lines in your C:\autoexec.bat file.

set CATALINA = C:\apache-tomcat-8.0.28

set CLASSPATH = %CATALINA%\common\lib\servlet-api.jar;%CLASSPATH%

Alternatively, on Windows NT/2000/XP, you could go to My Computer –> Properties –> Advanced –> Environment Variables. Then, you would update the CLASSPATH value and press the OK button.

On Unix (Solaris, Linux, etc.), if you are using the C shell, you would put the following lines into your .cshrc file.

setenv CATALINA = /usr/local/apache-tomcat-8.0.28

setenv CLASSPATH $CATALINA/common/lib/servlet-api.jar:$CLASSPATH

Lifecycle of a Servlet

A servlet life cycle can be defined as the entire process from its creation till the destruction. The following are the paths followed by a servlet.

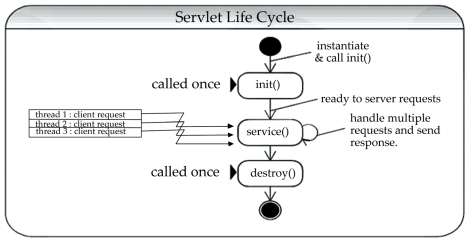
- The servlet is initialized by calling the **init()** method.
- The servlet calls service() method to process a client's request.
- The servlet is terminated by calling the destroy() method.
- Finally, servlet is garbage collected by the garbage collector of the JVM.

Now let us discuss the life cycle methods in detail.

The init() Method

The init method is called only once. It is called only when the servlet is created, and not called for any user requests afterwards. So, it is used for one-time initializations, just as with the init method of applets.

The servlet is normally created when a user first invokes a URL corresponding to the servlet, but you can also specify that the servlet be loaded when the server is first started.



When a user invokes a servlet, a single instance of each servlet gets created, with each user request resulting in a new thread that is handed off to doGet or doPost as appropriate. The init() method simply creates or loads some data that will be used throughout the life of the servlet.

The init method definition looks like this −

```
public void init() throws ServletException {
   // Initialization code...
}
```

The service() Method

The service() method is the main method to perform the actual task. The servlet container (i.e. web server) calls the service() method to handle requests coming from the client(browsers) and to write the formatted response back to the client.

Each time the server receives a request for a servlet, the server spawns a new thread and calls service. The service() method checks the HTTP request type (GET, POST, PUT, DELETE, etc.) and calls doGet, doPost, doPut, doDelete, etc. methods as appropriate.

Here is the signature of this method −

```
public void service(ServletRequest request, ServletResponse
response)
    throws ServletException, IOException {
}
```

The service () method is called by the container and service method invokes doGe, doPost, doPut, doDelete, etc. methods as appropriate. So you have nothing to do with service() method but you override either doGet() or doPost() depending on what type of request you receive from the client.

The doGet() and doPost() are most frequently used methods with in each service request. Here is the signature of these two methods.

The doGet() Method

A GET request results from a normal request for a URL or from an HTML form that has no METHOD specified and it should be handled by doGet() method.

```
public void doGet(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
    // Servlet code
}
```

The doPost() Method

A POST request results from an HTML form that specifically lists POST as the METHOD and it should be handled by doPost() method.

```
public void doPost(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
```

```
    // Servlet code
}
```

The destroy() Method

The destroy() method is called only once at the end of the life cycle of a servlet. This method gives your servlet a chance to close database connections, halt background threads, write cookie lists or hit counts to disk, and perform other such cleanup activities.

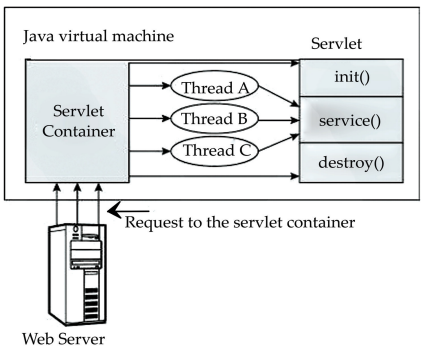
After the destroy() method is called, the servlet object is marked for **garbage collection**. The destroy method definition looks like this −

```
public void destroy() {
    // Finalization code...
}
```

Architecture Diagram

The following figure depicts a typical servlet life-cycle scenario.

- First the HTTP requests coming to the server are delegated to the servlet container.
- The servlet container loads the servlet before invoking the service() method.
- Then the servlet container handles multiple requests by spawning multiple threads, each thread executing the service() method of a single instance of the servlet.



Servlets - Examples

Servlets are Java classes which service HTTP requests and implement the **javax.servlet.Servlet** interface. Web application developers typically write servlets that extend javax.servlet. http.HttpServlet, an abstract class that implements the Servlet interface and is specially designed to handle HTTP requests.

Sample Code

Following is the sample source code structure of a servlet example to show Hello World −

```
// Import required java libraries
import java.io.*;
import javax.servlet.*;
import javax.servlet.http.*;

// Extend HttpServlet class
public class HelloWorld extends HttpServlet {

  private String message;

  public void init() throws ServletException {
    // Do required initialization
    message = “Hello World”;
  }

  public void doGet(HttpServletRequest request,
HttpServletResponse response)
    throws ServletException, IOException {
```

```
        // Set response content type
        response.setContentType("text/html");

        // Actual logic goes here.
        PrintWriter out = response.getWriter();
        out.println("<h1>" + message + "</h1>");
    }

    public void destroy() {
        // do nothing.
    }
}
```

Compiling a Servlet

Let us create a file with name HelloWorld.java with the code shown above. Place this file at C:\ServletDevel (in Windows) or at /usr/ServletDevel (in Unix). This path location must be added to CLASSPATH before proceeding further.

Assuming your environment is setup properly, go in **ServletDevel** directory and compile HelloWorld.java as follows –

```
$ javac HelloWorld.java
```

If the servlet depends on any other libraries, you have to include those JAR files on your CLASSPATH as well. You have included only servlet-api.jar JAR file because I'm not using any other library in Hello World program.

This command line uses the built-in javac compiler that comes with the Sun Microsystems Java Software Development Kit (JDK). For this command to work properly, you have to include the location of the Java SDK that you are using in the PATH environment variable. If everything goes fine, above compilation would produce **HelloWorld.class** file in the same directory.

Servlet Deployment

By default, a servlet application is located at the path <Tomcat-installationdirectory>/webapps/ROOT and the class file would reside in <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes.

If you have a fully qualified class name of **com.myorg.MyServlet**, then this servlet class must be located in WEB-INF/classes/com/myorg/MyServlet.class.

For now, let us copy HelloWorld.class into <Tomcat-installationdirectory>/webapps/ROOT/WEB-INF/classes and create following entries in **web.xml** file located in <Tomcat-installation-directory>/webapps/ROOT/WEB-INF/

```
<servlet>
   <servlet-name>HelloWorld</servlet-name>
   <servlet-class>HelloWorld</servlet-class>
</servlet>

<servlet-mapping>
   <servlet-name>HelloWorld</servlet-name>
   <url-pattern>/HelloWorld</url-pattern>
</servlet-mapping>
```

Above entries to be created inside <web-app>...</web-app> tags available in web.xml file. There could be various entries in this table already available, but never mind.

You are almost done, now let us start tomcat server using <Tomcat-installationdirectory>\bin\startup.bat (on Windows) or <Tomcat-installationdirectory>/bin/startup.sh (on Linux/Solaris etc.) and finally type **http://localhost:8080/HelloWorld** in the browser's address box. If everything goes fine, you would get the following result



JSDK

JSDK is the Java Servlet Developers Kit. This is an add on to the regular JDK (Java Developers Kit). The JSDK has the additional files needed in order to compile Java servlets. The latest version of the JSDK is version 2.0. Included in the JSDK is a Java Servlet Runner program. The Servlet Runner is a program that runs on your workstation, and allows you to test servlets you have written without running a web server. Other files included are the several Java Servlet examples. Included are the .java and .class files for testing purposes and to help you understand how the Java code is implemented. Another important file that is included is the jsdk.jar file. This file includes the class information necessary to compile the servlets.

JSDK (Java Servlet Development Kit) is a package containing all the classes and interfaces needed to develop servlets. JSDK also contains a web server and servlet engine to test your creations. The servlet engine provided in JSDK is a basic one(but free). There are many other servlet engines much more robust and can be interfaced with most major web servers of the market.

Installing JSDK

Let me preface the JSDK installation procedures by saying that the JSDK is an add on to the JDK. If you are trying to compile servlets using JSDK, you will be missing software. The JDK includes the javac.exe program and other java library and class files that are necessary to compile java code. Before you start to install the JSDK, install the JDK, then begin JSDK install.

Installing the JSDK is a very simple procedure. When you run the .exe that you download from the http://java.sun.com web site, it will create some temporary web files on your hard drive. It will then launch the installation script. You will be asked to

specify the directory you wish to have the files copied to. As in the JDK, the recommended directory includes a 'period' in the directory name (ex. C:\JSDK.20). Change this to C:\JSDK20 instead to make it easier when searching for the directory. After the software is installed, you will have a new subdirectory. In order to use both the JDK and JSDK together, the java compiler (javac.exe) needs to know where the class files are located. I found the easiest way to make this work, is to first put the bin directory for the JDK in the path. This will make it easy to find the javac.exe program when compiling code. Second, add the jsdk.jar file to the classpath. This can be done by adding a SET statement to the autoexec.bat file on your workstation (for Windows 95). The SET statement should read SET CLASSPATH = *drive:JSDK install path*\lib\jsdk.jar (ex. SET CLASSPATH = C:\jsdk20\lib\jsdk.jar). Once this is done, you will have no problem compiling your java servlets.

The Servelet AP

Before creating the first servlet, you need to understand the Servlet API and Tomcat Servlet container. The Servlet API provides interfaces and classes that are required to built servlets. These interfaces and classes are group into the following two packages :

- javax.servlet
- javax.servlet.http

One should remember that these packages are not part of Java core packages, instead they are standard extension provides by Tomcat. Therefore, they are not included with Java SE 6. First, we shall discuss in detail javax.servlet package, its classes and interfaces and then javax.servlet.http package. The javax.servlet and javax.servlet.http packages represent interfaces and classes for servlet api.

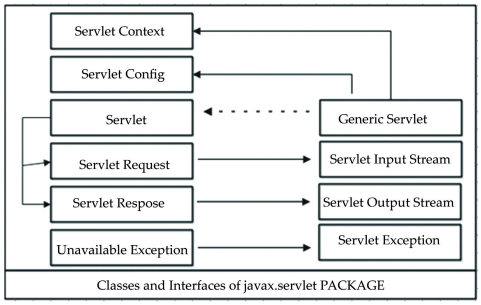
The **javax.servlet** package contains many interfaces and classes that are used by the servlet or web container. These are not specific to any protocol.

The **javax.servlet.http** package contains interfaces and classes that are responsible for http requests only.

Let's see what are the interfaces of javax.servlet package.

javax.servlet PACKAGE

The javax.Servlet package contains a number of interfaces and classes that establish the framework in which servlets operate. The classes and interface in this package are protocol independent. The Fig. shows classes and interfaces in this package.



The javax. servlet package contains many interfaces and classes. Some of the interfaces and classes are listed in table

Interfaces	Description
Servlet	Declares life cycle methods that all servlets must implement.
ServletConfig	Allows servlets to get initialization parameters
ServletContext	Allows servlets to communicate with its servlet container.
ServletRequest	Provides client request information to a servlet.
ServletResponse	Assist a servlet in sending a response to the client.

Classes	Description
GenericServlet	Provides a basic implementation of the Servlet interface for protocol independent servlets
ServletInputStream	Provides an input stream for reading binary data from a client request.
ServletOutputStream	Provides an output stream for sending binary data to the client.
ServletException	Defines a general exception, a servlet can throw when it encounters difficulty.
UnavailableException	Indicates that a servlet is not available to service client request.

Interfaces in javax.servlet package

There are many interfaces in javax.servlet package. They are as follows:

1. Servlet
2. ServletRequest
3. ServletResponse
4. RequestDispatcher
5. ServletConfig
6. ServletContext
7. SingleThreadModel
8. Filter
9. FilterConfig
10. FilterChain
11. ServletRequestListener
12. ServletRequestAttributeListener
13. ServletContextListener
14. ServletContextAttributeListener

Classes in javax.servlet package

There are many classes in javax.servlet package. They are as follows:

1. GenericServlet
2. ServletInputStream
3. ServletOutputStream
4. ServletRequestWrapper
5. ServletResponseWrapper
6. ServletRequestEvent
7. ServletContextEvent
8. ServletRequestAttributeEvent
9. ServletContextAttributeEvent
10. ServletException
11. UnavailableException

Interfaces in javax.servlet.http package

There are many interfaces in javax.servlet.http package. They are as follows:

1. HttpServletRequest
2. HttpServletResponse
3. HttpSession
4. HttpSessionListener
5. HttpSessionAttributeListener
6. HttpSessionBindingListener
7. HttpSessionActivationListener
8. HttpSessionContext (deprecated now)

Classes in javax.servlet.http package

There are many classes in javax.servlet.http package. They are as follows:

1. HttpServlet
2. Cookie
3. HttpServletRequestWrapper
4. HttpServletResponseWrapper
5. HttpSessionEvent
6. HttpSessionBindingEvent
7. HttpUtils (deprecated now)

READING HTTP REQUEST PARAMETERS

Even if you are new to web programming, you have probably seen HTTP request parameters. These are the bits sometimes "tacked on" to the end of a URL when you submit a form on a web page. For example, if an HTML form was defined as follows with a username field:

```
<form action="http://www.examplesite.com/login">

<input type=text name="username">

<input type=submit>

</form>
```

Then submitting the form will make the browser request http://www.examplesite.com/login, but with the username parameter tacked on to the end:

http://www.examplesite.com/login**?username=mcjones**

When parameters are used

As well as being used for HTML forms as in the above example, HTTP parameters are also used in cases such as:

- to pass simple data in AJAX web applications;
- in some APIs designed for the programmer to call a web service, passing in parameters directly via the URL.

Essentially, any time a simple, short piece of data needs to be passed from the client to the server, they are a reasonable choice.

Reading parameters from the Servlet

Instead of adding request parameters to the end of the URL, it is also possible to POST them as actual data. To the Servlet, it makes no difference which request method (GET or POST) was used to send the parameters. In either case, we read the parameter using the getParameter() method of HttpServletRequest:

```
public void doGet(HttpServletRequest req,
HttpServletResponse res)
    throws ServletException, IOException {
  String userName = req.getParameter("username");
}
```

If the given parameter was not set, then null was returned. A couple of methods are available to help us enumerate the parameters passed in:

- the HttpServletRequest.getParameterNames() method returns an Enumeration of all defined parameters;
- the rarely used HttpServletRequest.getParameterMap() method returns a map of parameters; howver, the map returned the maps string parameter names to a string *array* of values.

It may sound odd that getParameterMap() returns mappings to *arrays* of strings. The reason is that strictly speaking, more than one value can be assigned to the same parameter name.

Parsing POSTed parameters "manually"

Usually, the most convenient way of reading request parameters is via the getParameter() method. It should be noted that if you use this method to read parameters that are POSTed, you *should not* generally use the getReader() method to obtain and read the POSTed data "manually".

Parameters vs request headers

HTTP **parameters** are generally used to pass *user-supplied* data. They should not be confused with HTTP request headers, which are generally used to carry "meta" information, such as the character encoding, referring page, type of browser etc.

Reading Initialization Parameters

The servlet container provides the initialization parameters for a servlet or filter within the configuration object the container passes to the init method.

The configuration object provides agetInitParameter() function that takes a ***string*** name and returns the contents of the initialization parameter by that name. A filter uses the configuration object to access initialization parameters. A servlet uses the convenience functions provided by ***rwsf::GenericServlet*** to access initialization parameters directly.

The code sample below shows a servlet retrieving the initialization parameters from the servlet element

- std::string greeting = getInitParameter("greeting");
- std::string fontsize = getInitParameter("fontsize");

If the instance definition does not contain an initialization parameter with the name provided, the getInitParameter() function returns an empty string.

Example Using Initialization Parameters

This program shows a servlet that reads the message and repeats initialization parameters when initialized. It shows the result when message is Shibboleth, repeats is 5, and the servlet is registered under the name ShowMsg. Remember that, although servlets *read* init parameters in a standard way, developers *set* init parameters in a server-specific manner. Please refer to your server documentation for authoritative details. The second program shows the configuration file used with Tomcat to obtain the result, the third program shows the configuration file used with the JSWDK, and it show how to set the parameters interactively with the Java Web Server. The result is identical with all three cases. Because the process of setting init parameters is server-specific, it is a good idea to minimize the number of separate initialization entries that have to be specified. This will limit the work you need to do when moving servlets that use init parameters from one server to another. If you need to read a large amount of data, you recommend that the init parameter itself merely give the location of a parameter file, and that the real data go in that file. An example of this approach is given in article (Restricting Access to Web Pages), where the initialization parameter specifies nothing more than the location of the password file.

ShowMessage.java

```
package coreservlets;

import java.io.*;

import javax.servlet.*;

import javax.servlet.http.*;

public class ShowMessage extends HttpServlet {

private String message;

private String defaultMessage = “No message.”;

private int repeats = 1;

public void init(ServletConfig config)

throws ServletException {
```

```
// Always call super.init
super.init(config);
message = config.getInitParameter("message");
if (message == null) {
message = defaultMessage;
}
try {
String repeatString = config.getInitParameter("repeats");
repeats = Integer.parseInt(repeatString);
} catch(NumberFormatException nfe) {
}
}
public void doGet(HttpServletRequest request,
HttpServletResponse response)
throws ServletException, IOException {
response.setContentType("text/html");
PrintWriter out = response.getWriter();
String title = "The ShowMessage Servlet";
out.println(ServletUtilities.headWithTitle(title) +
"<BODY BGCOLOR=\"#FDF5E6\">\n" +
"<H1 ALIGN=CENTER>" + title + "</H1>");
```

This second program shows the setup file used to supply initialization parameters to servlets used with Tomcat 3.0. The idea is that you first associate a name with the servlet class file, and then associate initialization parameters with that name (not with the actual class file). The setup file is located in *install_dir*/webpages/WEB-INF.

The third program shows the properties file used to supply initialization parameters to servlets in the JSWDK. As with

Tomcat, you first associate a name with the servlet class, then associate the initialization parameters with the name. The properties file is located in *install_dir*/webpages/WEB-INF.

```
for(int i=0; i<repeats; i++) {
out.println(message + "<BR>");
}
out.println("</BODY></HTML>");
}
}
```

web.xml (for Tomcat)

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE web-app
PUBLIC "-//Sun Microsystems, Inc.//DTD Web Application
2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2.2.dtd">
<web-app>
<servlet>
<servlet-name>
ShowMsg
</servlet-name>
<servlet-class>
coreservlets.ShowMessage
</servlet-class>
<init-param>
<param-name>
message
</param-name>
```

<param-value>

Shibboleth

</param-value>

</init-param>

<init-param>

<param-name>

repeats

</param-name>

<param-value>

5

</param-value>

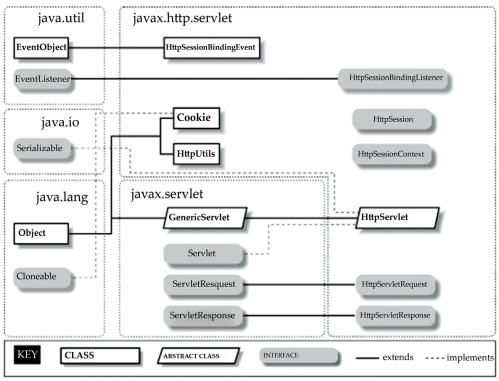
</init-param>

</servlet>

</web-app>

The javax.servelet HTTP Package

The javax.servlet.http package supports the development of servlets that use the HTTP protocol. The classes in this package extend the basic servlet functionality to support various HTTP specific features, including request and response headers, different request methods, and cookies.



The abstract HttpServlet class extendsjavax.servlet. GenericServlet and serves as the base class for HTTP servlets. HttpServlet-Request and HttpServletResponse allow additional interaction with the client. Finally, since the HTTP protocol is inherently stateless, the package also includes HttpSession and some related classes to support session tracking.

javax.servlet.http (cloneable)

The Cookie class provides servlets with an easy way to read, create, and manipulate HTTP-style cookies. Cookies provide a way to store a small amount of information on the client and are typically used for session tracking or storing user-configuration information. The getCookies() method of HttpServletRequest returns an array of Cookie objects. To set a new cookie on the client, a servlet creates a new Cookie object and uses the addCookie() method of HttpServletResponse. This must be done before sending any other content, since cookies are created within the HTTP header stream. The various methods of the Cookie class allow a servlet to set and get various attributes of a Cookie object, such as its path and domain.

```
public class Cookie implements Cloneable {

//Public Constructors

public Cookie (String name, String value);

//Public Instance Methods

public String getComment ();

public String getDomain ();

public int getMaxAge ();

public String getName ();

public String getPath ();

public boolean getSecure ();

public String getValue ();

public int getVersion ();

public void setComment (String purpose);
```

public void setDomain (String pattern);

public void setMaxAge (int expiry);

public void setPath (String uri);

public void setSecure (boolean flag);

public void setValue (String newValue);

public void setVersion (int v);

//Public methods overriding Object

public Object clone ();

}

Hierarchy: Object-->Cookie(Cloneable)

Passed To: HttpServletResponse.addCookie()

Returned By: HttpServletRequest.getCookies()

javax.servlet.http (serializable)

The abstract HttpServlet class serves as a framework for servlets that generate content for the World Wide Web using the HTTP protocol. Rather than overriding the service() method, you should override one or more of the method-specific request handlers (doGet(), doPost(), doPut(), etc.). The default service() implementation dispatches incoming requests to the appropriate methods, and so should not be overridden. The default implementations of doGet(), doPost(), doDelete(), and doPut() all return an HTTP BAD_REQUEST error, so if you want to handle one of these kinds of requests, you must override the appropriate method.

A web server calls getLastModified() in response to conditional GET requests. The default implementation returns -1. If you know when the output of your servlet last changed, you can return that time, specified in milliseconds since midnight, January 1, 1970 GMT, instead. This allows web browsers to cache your servlet's response.

public abstract class HttpServlet extends GenericServletimplements Serializable {

```
//Public Constructors

public HttpServlet ();

//Public methods overriding GenericServlet

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

//Protected Instance Methods

protected void doDelete (HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException;

protected void doGet (HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException;

protected void doOptions (HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException;

protected void doPost (HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException;

protected void doPut (HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException;

protected void doTrace (HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException;

protected long getLastModified (HttpServletRequest req);

protected void service (HttpServletRequest req,
HttpServletResponse resp) throws ServletException, IOException;

}
```

Hierarchy: Object-->GenericServlet(Servlet,ServletConfig,Serializable)-->HttpServlet(Serializable)

Servlets - Client HTTP Request

When a browser requests for a web page, it sends lot of information to the web server which cannot be read directly because this information travel as a part of header of HTTP request. Following is the important header information which comes from browser side and you would use very frequently in web programming −

Sr.No.	Header & Description
1	**Accept** This header specifies the MIME types that the browser or other clients can handle. Values of **image/png** or **image/jpeg** are the two most common possibilities.
2	**Accept-Charset** This header specifies the character sets the browser can use to display the information. For example ISO-8859-1.
3	**Accept-Encoding** This header specifies the types of encodings that the browser knows how to handle. Values of **gzip** or **compress** are the two most common possibilities.
4	**Accept-Language** This header specifies the client's preferred languages in case the servlet can produce results in more than one language. For example en, en-us, ru, etc
5	**Authorization** This header is used by clients to identify themselves when accessing password-protected Web pages.
6	**Connection** This header indicates whether the client can handle persistent HTTP connections. Persistent connections permit the client or other browser to retrieve multiple files with a single request. A value of **Keep-Alive** means that persistent connections should be used.
7	**Content-Length** This header is applicable only to POST requests and gives the size of the POST data in bytes.
8	**Cookie** This header returns cookies to servers that previously sent them to the browser.
9	**Host** This header specifies the host and port as given in the original URL.
10	**If-Modified-Since** This header indicates that the client wants the page only if it has been changed after the specified date. The server sends a code, 304 which means **Not Modified** header if no newer result is available.
11	**If-Unmodified-Since** This header is the reverse of If-Modified-Since; it specifies that the operation should succeed only if the document is older than the specified date.

12	**Referer** This header indicates the URL of the referring Web page. For example, if you are at Web page 1 and click on a link to Web page 2, the URL of Web page 1 is included in the Referrer header when the browser requests Web page 2.
13	**User-Agent** This header identifies the browser or other client making the request and can be used to return different content to different types of browsers.

Methods to read HTTP Header

There are following methods which can be used to read HTTP header in your servlet program. These methods are available with *HttpServletRequest* object

Sr.No.	Method & Description
1	**Cookie** **. getCookies()** Returns an array containing all of the Cookie objects the client sent with this request.
2	**Enumeration getAttributeNames()** Returns an Enumeration containing the names of the attributes available to this request.
3	**Enumeration getHeaderNames()** Returns an enumeration of all the header names this request contains.
4	**Enumeration getParameterNames()** Returns an Enumeration of String objects containing the names of the parameters contained in this request
5	**HttpSession getSession()** Returns the current session associated with this request, or if the request does not have a session, creates one.
6	**HttpSession getSession(boolean create)** Returns the current HttpSession associated with this request or, if if there is no current session and value of create is true, returns a new session.
7	**Locale getLocale()** Returns the preferred Locale that the client will accept content in, based on the Accept-Language header.

8	**Object getAttribute(String name)** Returns the value of the named attribute as an Object, or null if no attribute of the given name exists.
9	**ServletInputStream getInputStream()** Retrieves the body of the request as binary data using a ServletInputStream.
10	**String getAuthType()** Returns the name of the authentication scheme used to protect the servlet, for example, "BASIC" or "SSL," or null if the JSP was not protected.
11	**String getCharacterEncoding()** Returns the name of the character encoding used in the body of this request.
12	**String getContentType()** Returns the MIME type of the body of the request, or null if the type is not known.
13	**String getContextPath()** Returns the portion of the request URI that indicates the context of the request.
14	**String getHeader(String name)** Returns the value of the specified request header as a String.
15	**String getMethod()** Returns the name of the HTTP method with which this request was made, for example, GET, POST, or PUT.
16	**String getParameter(String name)** Returns the value of a request parameter as a String, or null if the parameter does not exist.
17	**String getPathInfo()** Returns any extra path information associated with the URL the client sent when it made this request
18	**String getProtocol()** Returns the name and version of the protocol the request.
19	**String getQueryString()** Returns the query string that is contained in the request URL after the path.
20	**String getRemoteAddr()** Returns the Internet Protocol (IP) address of the client that sent the request.
21	**String getRemoteHost()** Returns the fully qualified name of the client that sent the request.

22	**String getRemoteUser()** Returns the login of the user making this request, if the user has been authenticated, or null if the user has not been authenticated.
23	**String getRequestURI()** Returns the part of this request's URL from the protocol name up to the query string in the first line of the HTTP request.
24	**String getRequestedSessionId()** Returns the session ID specified by the client.
25	**String getServletPath()** Returns the part of this request's URL that calls the JSP.
26	**String** **. getParameterValues(String name)** Returns an array of String objects containing all of the values the given request parameter has, or null if the parameter does not exist.
27	**boolean isSecure()** Returns a Boolean indicating whether this request was made using a secure channel, such as HTTPS.
28	**int getContentLength()** Returns the length, in bytes, of the request body and made available by the input stream, or -1 if the length is not known.
29	**int getIntHeader(String name)** Returns the value of the specified request header as an int.
30	**int getServerPort()** Returns the port number on which this request was received.

Servlets - Session Tracking

HTTP is a "stateless" protocol which means each time a client retrieves a Web page, the client opens a separate connection to the Web server and the server automatically does not keep any record of previous client request.

Still there are following three ways to maintain session between web client and web server –

Cookies

A webserver can assign a unique session ID as a cookie to each web client and for subsequent requests from the client they can be recognized using the recieved cookie.

This may not be an effective way because many time browser does not support a cookie, so you would not recommend to use this procedure to maintain the sessions.

Hidden Form Fields

A web server can send a hidden HTML form field along with a unique session ID as follows −

<input type = “hidden” name = “sessionid” value = “12345”>

This entry means that, when the form is submitted, the specified name and value are automatically included in the GET or POST data. Each time when web browser sends request back, then session_id value can be used to keep the track of different web browsers.

This could be an effective way of keeping track of the session but clicking on a regular (<A HREF...>) hypertext link does not result in a form submission, so hidden form fields also cannot support general session tracking.

URL Rewriting

You can append some extra data on the end of each URL that identifies the session, and the server can associate that session identifier with data it has stored about that session.

For example, with http://tutorialspoint.com/file.htm;sessionid = 12345, the session identifier is attached as sessionid = 12345 which can be accessed at the web server to identify the client.

URL rewriting is a better way to maintain sessions and it works even when browsers do not support cookies. The drawback of URL re-writing is that you would have to generate every URL dynamically to assign a session ID, even in case of a simple static HTML page.

The HttpSession Object

Apart from the above mentioned three ways, servlet provides HttpSession Interface which provides a way to identify a user across more than one page request or visit to a Web site and to store information about that user.

The servlet container uses this interface to create a session between an HTTP client and an HTTP server. The session persists for a specified time period, across more than one connection or page request from the user.

You would get HttpSession object by calling the public method **getSession()**of HttpServletRequest, as below –

```
HttpSession session = request.getSession();
```

You need to call *request.getSession()* before you send any document content to the client. Here is a summary of the important methods available through HttpSession object –

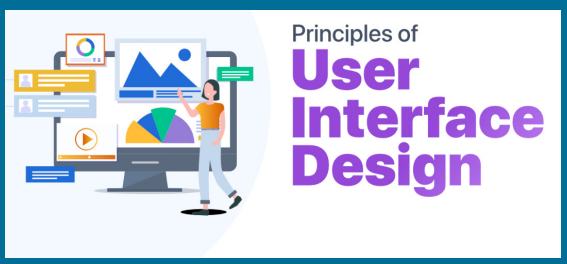
Sr.No.	Method & Description
1	**public Object getAttribute(String name)** This method returns the object bound with the specified name in this session, or null if no object is bound under the name.
2	**public Enumeration getAttributeNames()** This method returns an Enumeration of String objects containing the names of all the objects bound to this session.
3	**public long getCreationTime()** This method returns the time when this session was created, measured in milliseconds since midnight January 1, 1970 GMT.
4	**public String getId()** This method returns a string containing the unique identifier assigned to this session.
5	**public long getLastAccessedTime()** This method returns the last accessed time of the session, in the format of milliseconds since midnight January 1, 1970 GMT
6	**public int getMaxInactiveInterval()** This method returns the maximum time interval (seconds), that the servlet container will keep the session open between client accesses.

7	**public void invalidate()** This method invalidates this session and unbinds any objects bound to it.
8	**public boolean isNew(** This method returns true if the client does not yet know about the session or if the client chooses not to join the session.
9	**public void removeAttribute(String name)** This method removes the object bound with the specified name from this session.
10	**public void setAttribute(String name, Object value)** This method binds an object to this session, using the name specified.
11	**public void setMaxInactiveInterval(int interval)** This method specifies the time, in seconds, between client requests before the servlet container will invalidate this session.

CASE STUDY

USER INTERFACE DESIGN

Broad objective of this case study is to analyze User Interface Design. User interface design or engineering is the structure of websites, computer systems, appliances, machines, cell phone communication devices, and computer software with the give attention to the user's practical knowledge and interaction. The purpose of user interface design should be to make the customer's interaction as uncomplicated and efficient as it can be, in terms involving accomplishing user goals—what is usually called user-centered style. Other objectives are: Design graphical user interface (GUI) and printed computer outputs, Design GUI input screens that use the proper screen-based controls and Design a GUI interface that integrates the above outputs and inputs.



Questions

1. Give a detailed overview on design graphical user interface (GUI).
2. What is the purpose of user interface design?

KEY TERMS

HTTP (Hypertext Transfer Protocol)
Internet Information Server (IIS)
Web server
Garbage collection
Protocol
Network protocol
File Transfer Protocol
Server
URL
Web programming
Interface

DISCUSSION QUESTIONS

1. Define the concept of web servers and servlets.
2. How to setup a servlets-environment?
3. What do you understand by the tomcat web server?
4. Elaborate the JSDK.
5. Explain the lifecycle of a servlet.
6. Focus on servlets - client http request
7. Define the methods to read http header
8. Learn about servlets - session tracking

REFERENCES

1. C. D. Bustamante, A. Fledel-Alon, S. Williamson, R. Nielsen, M. Todd Hubisz, S. Glanowski, D. M. Tanenbaum, T. J. White, J. J. Sninsky, R. D. Hernandez et al. Natural selection on protein-coding genes in the human genome. Nature, 437(7062):1153–1157, 2005.
2. Chimpanzee Sequencing and Analysis Consortium. Initial sequence of the chimpanzee genome and comparison with the human genome. Nature, 437(7055):69– 87, 2005.
3. E. V. Ball, P. D. Stenson, S. S. Abeysinghe, M. Krawczak, D. N. Cooper, and N. A. Chuzhanova. Microdeletions and microinsertions causing human genetic disease: common mechanisms of mutagenesis and the role of local DNA sequence complexity. Hum Mutat, 26(3):205–13, 2005.
4. F.-C. Chen and W.-H. Li. Genomic divergences between humans and other hominoids and the effective population size of the common ancestor of humans and chimpanzees. Am J Hum Genet, 68(2):444–456, 2001.
5. F.-C. Chen, C.-J. Chen, W.-H. Li, and T.-J. Chuang. Human-specific insertions and deletions inferred from mammalian genome sequences. Genome Res, 17(1):16–22, 2007.
6. H. M. Berman, J. Westbrook, Z. Feng, G. Gilliland, T. N. Bhat, H. Weissig, I. N. Shindyalov, and P. E. Bourne. The Protein Data Bank. Nucleic Acids Res, 28(1):235–242, 2000.
7. J. A. Bailey and E. E. Eichler. Primate segmental duplications: crucibles of evolution, diversity and disease. Nat Rev Genet, 7(7):552–564, 2006.
8. M. Ashburner, C. A. Ball, J. A. Blake, D. Botstein, H. Butler, J. M. Cherry, A. P. Davis, K. Dolinski, S. S. Dwight, J. T. Eppig et al. Gene Ontology: tool for the unification of biology. The Gene Ontology Consortium. Nat Genet, 25(1):25–29, 2000.
9. M. Barluenga, K. N. Stolting, W. Salzburger, M. Muschick, and A. Meyer. Sympatric speciation in Nicaraguan crater lake cichlid fish. Nature, 439(7077):719–23, 2006.
10. M. Brudno, C. B. Do, G. M. Cooper, M. F. Kim, E. Davydov, NISC Comparative Sequencing Program, E. D. Green, A. Sidow, and S. Batzoglou. LAGAN and Multi-LAGAN: efficient tools for large-scale multiple alignment of genomic DNA. Genome Res, 13(4):721–731, 2003.
11. M. R. Barnes and I. C. Gray. Amino acid properties and consequences of subsitutions. In Bioinformatics for Geneticists. Wiley, 2003.

CHAPTER 6

JAVASERVER PAGES (JSP)



LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Give the overview of javaserver pages (JSP)
2. How to Setup the JSP environment
3. Define the JSP architecture
4. Learn about Syntax in JSP
5. Focus on JSP directives

INTRODUCTION

JavaServer Pages (JSP) is a technology that helps software developers create dynamically generated web pages based on HTML, XML, or other document types. Released in 1999 by Sun Microsystems, JSP is similar to PHP and ASP, but it uses the Java programming language.

JavaServer Pages (JSP) is a server-side programming technology that enables the creation of dynamic web pages and applications. This is accomplished by embedding Java code into HTML, XML, DHTML, or other document types. When a client such as a web browser makes a request to the Java application container, which is typically a web server, the static page is converted behind the scenes, and displayed as dynamic content to the viewer. It has been speculated that JSP was released to compete with PHP, a commonly used open-source programming language.

While JSP is sometimes associated with Active Server Pages (ASP), these are actually two different technologies. The biggest difference is that ASP is developed by Microsoft&reg, and largely considered as a framework. This is due to its ability to support a wide variety of programming languages enabled by Microsoft's&reg Component Object Model (COM**)**. Created by Sun Microsystems&reg, JSP is considered an object-orientated programming language.



JavaServer Pages technology allows Java code and select predefined actions to be inserted into static web page content. This code is compiled at runtime for each request made to the page. The server, Java code and/or any custom programming that may have been implemented, is controlled by a pre-installed software application known as a ***virtual machine*** **(VM)**. This VM

integrates with a host operating system. The most common VM used to run JSP applications is Java Virtual Machine™ (JVM).

From a technical aspect, JSP can be viewed as a high-level abstraction of Java servlets, because they essentially run on top of them. JavaServer Pages are usually compiled by a Java compiler. This compiler either generates servlets in the form of Java code, or byte code that can be directly executed. Since they are capable of being interpreted as soon as requests are made, the time it takes to convert a static page to dynamic content is often greatly reduced.

Many programmers prefer JSP because it simplifies development by allowing the insertion of Java code directly into standard text files. It is also an independent, cross-platform, portable language that falls in line with the Java motto of 'write once, run anywhere.' Boasting the ability to separate content from the foundation of the page, the language allows programmers to create new code independent of the existing work. JSP is largely considered to be a robust and efficient technology for both the server and client side aspects of programming.

OVERVIEW OF JAVASERVER PAGES (JSP)

JavaServer Pages (JSP) is a technology for developing Webpages that supports dynamic content. This helps developers insert java code in HTML pages by making use of special JSP tags, most of which start with <% and end with %>.

A JavaServer Pages component is a type of Java servlet that is designed to fulfill the role of a user interface for a Java web application. Web developers write JSPs as text files that combine HTML or XHTML code, XML elements, and embedded JSP actions and commands.

Using JSP, you can collect input from users through Webpage forms, present records from a database or another source, and create Webpages dynamically.

JSP tags can be used for a variety of purposes, such as retrieving information from a database or registering user preferences, accessing JavaBeans components, passing control between pages, and sharing information between requests, pages etc.



Why Use JSP?

JavaServer Pages often serve the same purpose as programs implemented using the Common Gateway Interface (CGI). But JSP offers several advantages in comparison with the CGI.

- Performance is significantly better because JSP allows embedding Dynamic Elements in HTML Pages itself instead of having separate CGI files.
- JSP are always compiled before they are processed by the server unlike CGI/Perl which requires the server to load an interpreter and the target script each time the page is requested.
- JavaServer Pages are built on top of the Java Servlets API, so like Servlets, JSP also has access to all the powerful Enterprise Java APIs, including **JDBC, JNDI, EJB, JAXP,** etc.
- JSP pages can be used in combination with servlets that handle the business logic, the model supported by **Java servlet** template engines.

Finally, JSP is an integral part of Java EE, a complete platform for enterprise class applications. This means that JSP can play a part in the simplest applications to the most complex and demanding.

Advantages of JSP

There are many advantages of JSP over servlet. They are as follows:

1. ***Extension to Servlet***: JSP technology is the extension to servlet technology. We can use all the features of servlet in JSP. In addition to, we can use implicit objects, predefined tags, expression language and Custom tags in JSP that makes JSP development easy.

2. ***Easy to maintain***: JSP can be easily managed because we can easily separate our business logic with presentation logic. In servlet technology, we mix our business logic with the presentation logic.
3. ***Fast Development: No need to recompile and redeploy***: If JSP page is modified, we don't need to recompile and redeploy the project. The servlet code needs to be updated and recompiled if we have to change the look and feel of the application.
4. ***Less code than Servlet***: In JSP, we can use a lot of tags such as action tags, jstl, custom tags etc. that reduces the code. Moreover, we can use EL, implicit objects etc.

Life Cycle of a JSP Page

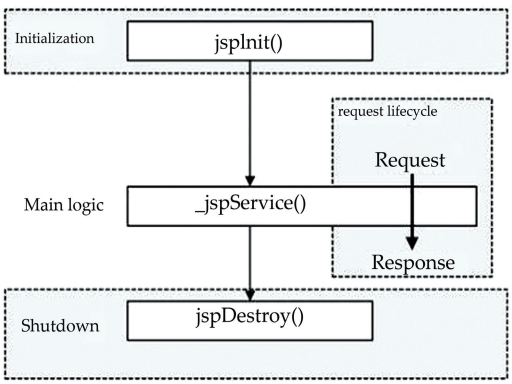
A JSP life cycle is defined as the process from its creation till the destruction. This is similar to a servlet life cycle with an additional step which is required to compile a JSP into servlet.

Paths Followed By JSP

The following are the paths followed by a JSP –

- Compilation
- Initialization
- Execution
- Cleanup

The four major phases of a JSP life cycle are very similar to the Servlet Life Cycle. The four phases have been described below –



JSP Compilation

When a browser asks for a JSP, the JSP engine first checks to see whether it needs to compile the page. If the page has never been compiled, or if the JSP has been modified since it was last compiled, the JSP engine compiles the page.

The compilation process involves three steps –

- Parsing the JSP.
- Turning the JSP into a servlet.
- Compiling the servlet.

JSP Initialization

When a container loads a JSP it invokes the **jspInit()** method before servicing any requests. If you need to perform JSP-specific initialization, override the **jspInit()** method –

```
public void jspInit(){
   // Initialization code...
}
```

Typically, initialization is performed only once and as with the servlet init method, you generally initialize database connections, open files, and create lookup tables in the jspInit method.

JSP Execution

This phase of the JSP life cycle represents all interactions with requests until the JSP is destroyed.

Whenever a browser requests a JSP and the page has been loaded and initialized, the JSP engine invokes the **_jspService()** method in the JSP.

The _jspService() method takes an **HttpServletRequest** and an **HttpServletResponse** as its parameters as follows –

```
void _jspService(HttpServletRequest request,
HttpServletResponse response) {
   // Service handling code...
}
```

The **_jspService()** method of a JSP is invoked on request basis. This is responsible for generating the response for that request and this method is also responsible for generating responses to all seven of the HTTP methods, i.e, **GET, POST, DELETE**, etc.

JSP Cleanup

The destruction phase of the JSP life cycle represents when a JSP is being removed from use by a container.

The **jspDestroy()** method is the JSP equivalent of the destroy method for servlets. Override jspDestroy when you need to perform any cleanup, such as releasing database connections or closing open files.

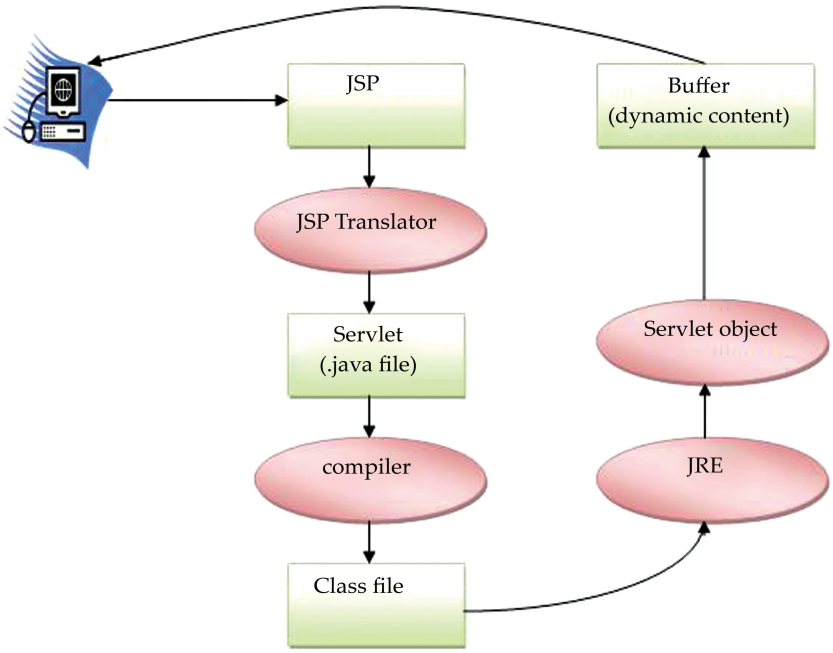
The jspDestroy() method has the following form −

```
public void jspDestroy() {
   // Your cleanup code goes here.
}
```

The JSP pages follows these phases:

- Translation of JSP Page
- Compilation of JSP Page
- Classloading (class file is loaded by the classloader)
- Instantiation (Object of the Generated Servlet is created).
- Initialization (jspInit() method is invoked by the container).
- Reqeust processing (_jspService() method is invoked by the container).
- Destroy (jspDestroy() method is invoked by the container).

Note: jspInit(), _jspService() and jspDestroy() are the life cycle methods of JSP.



As depicted in the above diagram, JSP page is translated into servlet by the help of JSP translator. The JSP translator is a part of webserver that is responsible to translate the JSP page into servlet. Afterthat Servlet page is compiled by the compiler and gets converted into the class file. Moreover, all the processes that happens in servlet is performed on JSP later like initialization, committing response to the browser and destroy.

Creating a simple JSP Page

To create the first jsp page, write some html code as given below, and save it by .jsp extension. We have save this file as index.jsp. Put it in a folder and paste the folder in the web-apps directory in apache tomcat to run the jsp page.

index.jsp

Let's see the simple example of JSP:

1. <html>
2. <body>
3. <% out.print(2*5); %>

4. </body>
5. </html>

It will print **10** on the browser.

How to run a simple JSP Page?

Follow the following steps to execute this JSP page:

- Start the server
- put the jsp file in a folder and deploy on the server
- visit the browser by the url http://localhost:portno/contextRoot/jspfile e.g. http://localhost:8888/myapplication/index.jsp

SETUP THE JSP ENVIRONMENT

A development environment is where you would develop your JSP programs, test them and finally run them.

Setting up Java Development Kit

This step involves downloading an implementation of the Java Software Development Kit (SDK) and setting up the PATH environment variable appropriately.

You can download SDK from Oracle's Java site – Java SE Downloads.

Once you download your Java implementation, follow the given instructions to install and configure the setup. Finally set the **PATH and JAVA_HOME**environment variables to refer to the directory that contains **java** and **javac**, typically **java_install_dir/bin** and **java_install_dir** respectively.

If you are running Windows and install the SDK in **C:\jdk1.5.0_20**, you need to add the following line in your **C:\autoexec.bat** file.

```
set PATH = C:\jdk1.5.0_20\bin;%PATH%
set JAVA_HOME = C:\jdk1.5.0_20
```

Alternatively, on **Windows NT/2000/XP**, you can also right-click on **My Computer**, select **Properties**, then **Advanced**, followed by **Environment Variables**. Then, you would update the PATH value and press the OK button.

On Unix (Solaris, Linux, etc.), if the SDK is installed in **/usr/local/jdk1.5.0_20** and you use the C shell, you will put the following into your **.cshrc** file.

```
setenv PATH /usr/local/jdk1.5.0_20/bin:$PATH

setenv JAVA_HOME /usr/local/jdk1.5.0_20
```

Alternatively, if you use an Integrated Development Environment (IDE) like Borland JBuilder, Eclipse, IntelliJ IDEA, or Sun ONE Studio, compile and run a simple program to confirm that the IDE knows where you installed Java.

Setting up Web Server: Tomcat

A number of Web Servers that support JavaServer Pages and Servlets development are available in the market. Some web servers can be downloaded for free and Tomcat is one of them.

Apache Tomcat is an open source software implementation of the JavaServer Pages and Servlet technologies and can act as a standalone server for testing JSP and Servlets, and can be integrated with the Apache Web Server. Here are the steps to set up Tomcat on your machine −

- Download the latest version of Tomcat from https://tomcat.apache.org/.
- Once you downloaded the installation, unpack the binary distribution into a convenient location. For example, in **C:\apache-tomcat-5.5.29 on windows, or /usr/local/apache-tomcat-5.5.29** on Linux/Unix and create **CATALINA_HOME** environment variable pointing to these locations.

Tomcat can be started by executing the following commands on the Windows machine −

```
%CATALINA_HOME%\bin\startup.bat
```

or

C:\apache-tomcat-5.5.29\bin\startup.bat

Tomcat can be started by executing the following commands on the Unix (Solaris, Linux, etc.) machine −

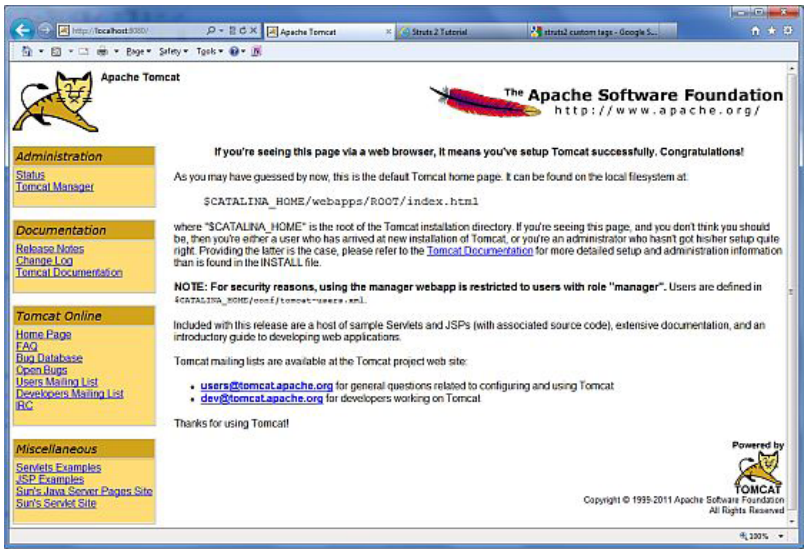
$CATALINA_HOME/bin/startup.sh

or

/usr/local/apache-tomcat-5.5.29/bin/startup.sh

After a successful startup, the default web-applications included with Tomcat will be available by visiting **http://localhost:8080/**.

Upon execution, you will receive the following output −



Tomcat can be stopped by executing the following commands on the Windows machine −

%CATALINA_HOME%\bin\shutdown

or

C:\apache-tomcat-5.5.29\bin\shutdown

Tomcat can be stopped by executing the following commands on Unix (Solaris, Linux, etc.) machine −

$CATALINA_HOME/bin/shutdown.sh

or

/usr/local/apache-tomcat-5.5.29/bin/shutdown.sh

Setting up CLASSPATH

Since servlets are not part of the Java Platform, Standard Edition, you must identify the servlet classes to the compiler.

If you are running Windows, you need to put the following lines in your **C:\autoexec.bat** file.

set CATALINA = C:\apache-tomcat-5.5.29

set CLASSPATH = %CATALINA%\common\lib\jsp-api.jar;%CLASSPATH%

Alternatively, on **Windows NT/2000/XP**, you can also right-click on **My Computer**, select **Properties**, then **Advanced**, then **Environment Variables**. Then, you would update the CLASSPATH value and press the OK button.

On Unix (Solaris, Linux, etc.), if you are using the C shell, you would put the following lines into your **.cshrc** file.

setenv CATALINA = /usr/local/apache-tomcat-5.5.29

setenv CLASSPATH $CATALINA/common/lib/jsp-api.jar:$CLASSPATH

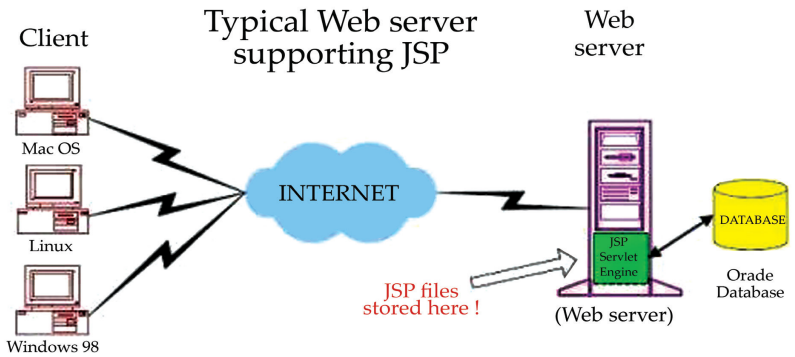
NOTE − Assuming that your development directory is **C:\JSPDev (Windows)** or **/usr/JSPDev (Unix)**, then you would need to add these directories as well in CLASSPATH.

JSP ARCHITECTURE

The web server needs a JSP engine, i.e, a container to process JSP pages. The JSP container is responsible for intercepting requests for JSP pages. This tutorial makes use of Apache which has built-in JSP container to support JSP pages development.

A JSP container works with the Web server to provide the runtime environment and other services a JSP needs. It knows how to understand the special elements that are part of JSPs.

Following diagram shows the position of JSP container and JSP files in a Web application.



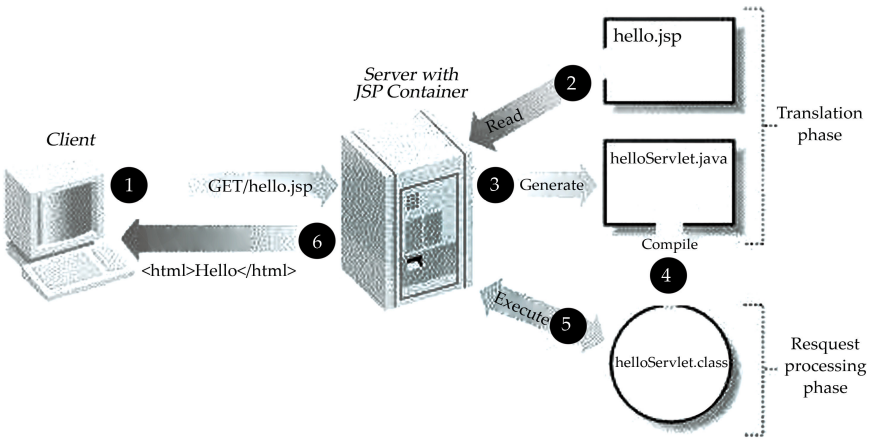
JSP Processing

The following steps explain how the web server creates the Webpage using JSP –

- As with a normal page, your browser sends an HTTP request to the web server.
- The web server recognizes that the HTTP request is for a JSP page and forwards it to a JSP engine. This is done by using the URL or JSP page which ends with **.jsp** instead of **.html**.
- The JSP engine loads the JSP page from disk and converts it into a servlet content. This conversion is very simple in which all template text is converted to println( ) statements and all JSP elements are converted to Java code. This code implements the corresponding dynamic behavior of the page.

- The JSP engine compiles the servlet into an executable class and forwards the original request to a servlet engine.
- A part of the web server called the servlet engine loads the Servlet class and executes it. During execution, the servlet produces an output in HTML format. The output is furthur passed on to the web server by the servlet engine inside an HTTP response.
- The web server forwards the HTTP response to your browser in terms of static HTML content.
- Finally, the web browser handles the dynamically-generated HTML page inside the HTTP response exactly as if it were a static page.

All the above mentioned steps can be seen in the following diagram –



Typically, the JSP engine checks to see whether a servlet for a JSP file already exists and whether the modification date on the JSP is older than the servlet. If the JSP is older than its generated servlet, the JSP container assumes that the JSP hasn't changed and that the generated servlet still matches the JSP's contents. This makes the process more efficient than with the other scripting languages (such as PHP) and therefore faster.

So in a way, a JSP page is really just another way to write a servlet without having to be a Java programming wiz. Except for the translation phase, a JSP page is handled exactly like a regular servlet.

SYNTAX IN JSP

We will understand the basic use of simple syntax (i.e, elements) involved with JSP development.

Elements of JSP

The elements of JSP have been described below –

The Scriptlet

A scriptlet can contain any number of JAVA language statements, variable or method declarations, or expressions that are valid in the page scripting language.

Following is the syntax of Scriptlet –

<% code fragment %>

You can write the XML equivalent of the above syntax as follows –

<jsp:scriptlet>

code fragment

</jsp:scriptlet>

Any text, HTML tags, or JSP elements you write must be outside the scriptlet. Following is the simple and first example for JSP –

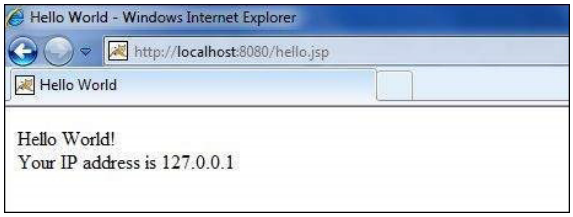
```
<html>
   <head><title>Hello World</title></head>

   <body>
      Hello World!<br/>
      <%
            out.println("Your IP address is " + request.
getRemoteAddr());
      %>
```

```
</body>
</html>
```

NOTE – Assuming that Apache Tomcat is installed in C:\apache-tomcat-7.0.2 and your environment is setup as per environment setup tutorial.

Let us keep the above code in JSP file **hello.jsp** and put this file in **C:\apache-tomcat7.0.2\webapps\ROOT** directory. Browse through the same using URL **http://localhost:8080/hello.jsp**. The above code will generate the following result –



JSP Declarations

A declaration declares one or more variables or methods that you can use in Java code later in the JSP file. You must declare the variable or method before you use it in the JSP file.

Following is the syntax for JSP Declarations –

```
<%! declaration; [ declaration; ]+ ... %>
```

You can write the XML equivalent of the above syntax as follows –

```
<jsp:declaration>
   code fragment
</jsp:declaration>
```

Following is an example for JSP Declarations –

```
<%! int i = 0; %>
<%! int a, b, c; %>
<%! Circle a = new Circle(2.0); %>
```

JSP Expression

A JSP expression element contains a scripting language expression that is evaluated, converted to a String, and inserted where the expression appears in the JSP file.

Because the value of an expression is converted to a String, you can use an expression within a line of text, whether or not it is tagged with HTML, in a JSP file.

The expression element can contain any expression that is valid according to the Java Language Specification but you cannot use a semicolon to end an expression.

Following is the syntax of JSP Expression −

<%= expression %>

You can write the XML equivalent of the above syntax as follows −

<jsp:expression>

expression

</jsp:expression>

Following example shows a JSP Expression −

```
<html>
   <head><title>A Comment Test</title></head>

   <body>
         <p>Today's date: <%= (new java.util.Date()).
toLocaleString()%></p>
   </body>
</html>
```

The above code will generate the following result −

Today's date: 11-Sep-2010 21:24:25

JSP Comments

JSP comment marks text or statements that the JSP container should ignore. A JSP comment is useful when you want to hide or "comment out", a part of your JSP page.

Following is the syntax of the JSP comments −

<%-- This is JSP comment --%>

Following example shows the JSP Comments −

```
<html>
   <head><title>A Comment Test</title></head>

   <body>
      <h2>A Test of Comments</h2>
       <%-- This comment will not be visible in the page source --%>
   </body>
</html>
```

The above code will generate the following result −

A Test of Comments

There are a small number of special constructs you can use in various cases to insert comments or characters that would otherwise be treated specially. Here's a summary −

S. No.	Syntax & Purpose
1	**<%-- comment --%>** A JSP comment. Ignored by the JSP engine.
2	**<!-- comment -->** An HTML comment. Ignored by the browser.
3	**<\%** Represents static <% literal.
4	**%\>** Represents static %> literal.

5	\' A single quote in an attribute that uses single quotes.
6	\" A double quote in an attribute that uses double quotes.

JSP Directives

A JSP directive affects the overall structure of the servlet class. It usually has the following form −

<%@ directive attribute="value" %>

There are three types of directive tag −

S. No.	Directive & Description
1	**<%@ page ... %>** Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
2	**<%@ include ... %>** Includes a file during the translation phase.
3	**<%@ taglib ... %>** Declares a tag library, containing custom actions, used in the page

We would explain the JSP directive in a separate chapter JSP - Directives

JSP Actions

JSP actions use **constructs** in XML syntax to control the behavior of the servlet engine. You can dynamically insert a file, reuse JavaBeans components, forward the user to another page, or generate HTML for the Java plugin.

There is only one syntax for the Action element, as it conforms to the XML standard −

<jsp:action_name attribute="value" />

Action elements are basically predefined functions. Following table lists out the available JSP Actions −

S. No.	Syntax & Purpose
1	**jsp:include** Includes a file at the time the page is requested.
2	**jsp:useBean** Finds or instantiates a JavaBean.
3	**jsp:setProperty** Sets the property of a JavaBean.
4	**jsp:getProperty** Inserts the property of a JavaBean into the output.
5	**jsp:forward** Forwards the requester to a new page.
6	**jsp:plugin** Generates browser-specific code that makes an OBJECT or EMBED tag for the Java plugin.
7	**jsp:element** Defines XML elements dynamically.
8	**jsp:attribute** Defines dynamically-defined XML element's attribute.
9	**jsp:body** Defines dynamically-defined XML element's body.
10	**jsp:text** Used to write template text in JSP pages and documents.

We would explain JSP actions in a separate chapter JSP - Actions

JSP Implicit Objects

JSP supports nine automatically defined variables, which are also called implicit objects. These variables are −

S. No.	Object & Description
1	**request** This is the **HttpServletRequest** object associated with the request.
2	**response** This is the **HttpServletResponse** object associated with the response to the client.
3	**out** This is the **PrintWriter** object used to send output to the client.
4	**session** This is the **HttpSession** object associated with the request.
5	**application** This is the **ServletContext** object associated with the application context.
6	**config** This is the **ServletConfig** object associated with the page.
7	**pageContext** This encapsulates use of server-specific features like higher performance **JspWriters**.
8	**page** This is simply a synonym for **this**, and is used to call the methods defined by the translated servlet class.
9	**Exception** The **Exception** object allows the exception data to be accessed by designated JSP.

We would explain JSP Implicit Objects in a separate chapter JSP - Implicit Objects.

Control-Flow Statements

You can use all the APIs and building blocks of Java in your JSP programming including decision-making statements, loops, etc.

Decision-Making Statements

The **if...else** block starts out like an ordinary Scriptlet, but the Scriptlet is closed at each line with HTML text included between the Scriptlet tags.

```
<%! int day = 3; %>

<html>
```

```
<head><title>IF...ELSE Example</title></head>

<body>
   <% if (day == 1 | day == 7) { %>
      <p> Today is weekend</p>
   <% } else { %>
      <p> Today is not weekend</p>
   <% } %>
</body>
</html>
```

The above code will generate the following result –

Today is not weekend

Now look at the following **switch...case** block which has been written a bit differentlty using **out.println()** and inside Scriptletas –

```
<%! int day = 3; %>
<html>
   <head><title>SWITCH...CASE Example</title></head>

   <body>
      <%
         switch(day) {
            case 0:
               out.println("It\'s Sunday.");
               break;
            case 1:
               out.println("It\'s Monday.");
```

```
            break;
         case 2:
            out.println("It\'s Tuesday.");
            break;
         case 3:
            out.println("It\'s Wednesday.");
            break;
         case 4:
            out.println("It\'s Thursday.");
            break;
         case 5:
            out.println("It\'s Friday.");
            break;
         default:
            out.println("It's Saturday.");
      }
   %>
  </body>
</html>
```

The above code will generate the following result –

It's Wednesday.

Loop Statements

You can also use three basic types of looping blocks in Java: **for, while, and do…while** blocks in your JSP programming.

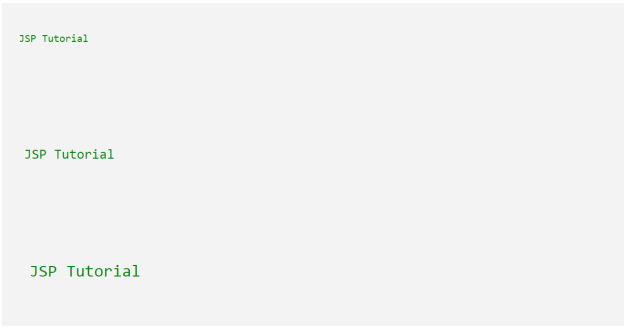
Let us look at the following **for** loop example –

```
<%! int fontSize; %>
```

```
<html>
   <head><title>FOR LOOP Example</title></head>

   <body>
      <%for ( fontSize = 1; fontSize <= 3; fontSize++){ %>
         <font color = "green" size = "<%= fontSize %>">
            JSP Tutorial
      </font><br />
      <%}%>
   </body>
</html>
```

The above code will generate the following result –



Above example can be written using the **while** loop as follows –

```
<%! int fontSize; %>
<html>
   <head><title>WHILE LOOP Example</title></head>

   <body>
      <%while ( fontSize <= 3){ %>
         <font color = "green" size = "<%= fontSize %>">
```

```
            JSP Tutorial
         </font><br />
         <%fontSize++;%>
      <%}%>
   </body>
</html>
```

The above code will generate the following result –

```
JSP Tutorial


 JSP Tutorial


  JSP Tutorial
```

JSP Operators

JSP supports all the logical and arithmetic operators supported by Java. Following table lists out all the operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom.

Within an expression, higher precedence operators will be evaluated first.

Category	Operator	Associativity
Postfix	() [] . (dot operator)	Left to right
Unary	++ - - ! ~	Right to left
Multiplicative	* / %	Left to right
Additive	+ -	Left to right
Shift	>> >>> <<	Left to right
Relational	> >= < <=	Left to right
Equality	== !=	Left to right
Bitwise AND	&	Left to right

Bitwise XOR	^	Left to right
Bitwise OR	|	Left to right
Logical AND	&&	Left to right
Logical OR	||	Left to right
Conditional	?:	Right to left
Assignment	= += -= *= /= %= >>= <<= &= ^= |=	Right to left
Comma	,	Left to right

JSP Literals

The JSP expression language defines the following literals –

- **Boolean** – true and false
- **Integer** – as in Java
- **Floating point** – as in Java
- **String** – with single and double quotes; " is escaped as \", ' is escaped as \', and \ is escaped as \\.
- **Null** – null

JSP DIRECTIVES

These directives provide directions and instructions to the container, telling it how to handle certain aspects of the JSP processing.

A JSP directive affects the overall structure of the servlet class. It usually has the following form –

<%@ directive attribute = "value" %>

Directives can have a number of attributes which you can list down as key-value pairs and separated by commas.

The blanks between the @ symbol and the directive name, and between the last attribute and the closing %>, are optional.

There are three types of directive tag −

S. No.	Directive & Description
1	**<%@ page ... %>** Defines page-dependent attributes, such as scripting language, error page, and buffering requirements.
2	**<%@ include ... %>** Includes a file during the translation phase.
3	**<%@ taglib ... %>** Declares a tag library, containing custom actions, used in the page

JSP The page Directive

The page directive is used to provide instructions to the container. These instructions pertain to the current JSP page. You may code page directives anywhere in your JSP page. By convention, page directives are coded at the top of the JSP page.

Following is the basic syntax of the page directive −

<%@ page attribute = “value” %>

You can write the XML equivalent of the above syntax as follows −

<jsp:directive.page attribute = “value” />

Attributes

Following table lists out the attributes associated with the page directive −

S. No.	Attribute & Purpose
1	**buffer** Specifies a buffering model for the output stream.
2	**autoFlush** Controls the behavior of the servlet output buffer.

3	**contentType** Defines the character encoding scheme.
4	**errorPage** Defines the URL of another JSP that reports on Java unchecked runtime exceptions.
5	**isErrorPage** Indicates if this JSP page is a URL specified by another JSP page's errorPage attribute.
6	**extends** Specifies a superclass that the generated servlet must extend.
7	**import** Specifies a list of packages or classes for use in the JSP as the Java import statement does for Java classes.
8	**info** Defines a string that can be accessed with the servlet's **getServletInfo()** method.
9	**isThreadSafe** Defines the threading model for the generated servlet.
10	**language** Defines the programming language used in the JSP page.
11	**session** Specifies whether or not the JSP page participates in HTTP sessions
12	**isELIgnored** Specifies whether or not the EL expression within the JSP page will be ignored.
13	**isScriptingEnabled** Determines if the scripting elements are allowed for use.

The include Directive

The include directive is used to include a file during the translation phase. This directive tells the container to merge the content of other external files with the current JSP during the translation phase. You may code the ***include***directives anywhere in your JSP page.

The general usage form of this directive is as follows −

<%@ include file = “relative url” >

The filename in the include directive is actually a relative URL. If you just specify a filename with no associated path, the JSP compiler assumes that the file is in the same directory as your JSP.

You can write the XML equivalent of the above syntax as follows −

<jsp:directive.include file = “relative url” />

The taglib Directive

The JavaServer Pages API allow you to define custom JSP tags that look like HTML or XML tags and a tag library is a set of user-defined tags that implement custom behavior.

The taglib directive declares that your JSP page uses a set of custom tags, identifies the location of the library, and provides means for identifying the custom tags in your JSP page.

The taglib directive follows the syntax given below −

<%@ taglib uri=”uri” prefix = “prefixOfTag” >

Here, the uri attribute value resolves to a location the container understands and the prefix attribute informs a container what bits of markup are custom actions.

You can write the XML equivalent of the above syntax as follows −

<jsp:directive.taglib uri = “uri” prefix = “prefixOfTag” />

KEY TERMS

Component Object Model (COM)
Java servlet
Integrated Development Environment (IDE)
Virtual machine (VM)
Java Virtual Machine
Special construct

DISCUSSION QUESTIONS

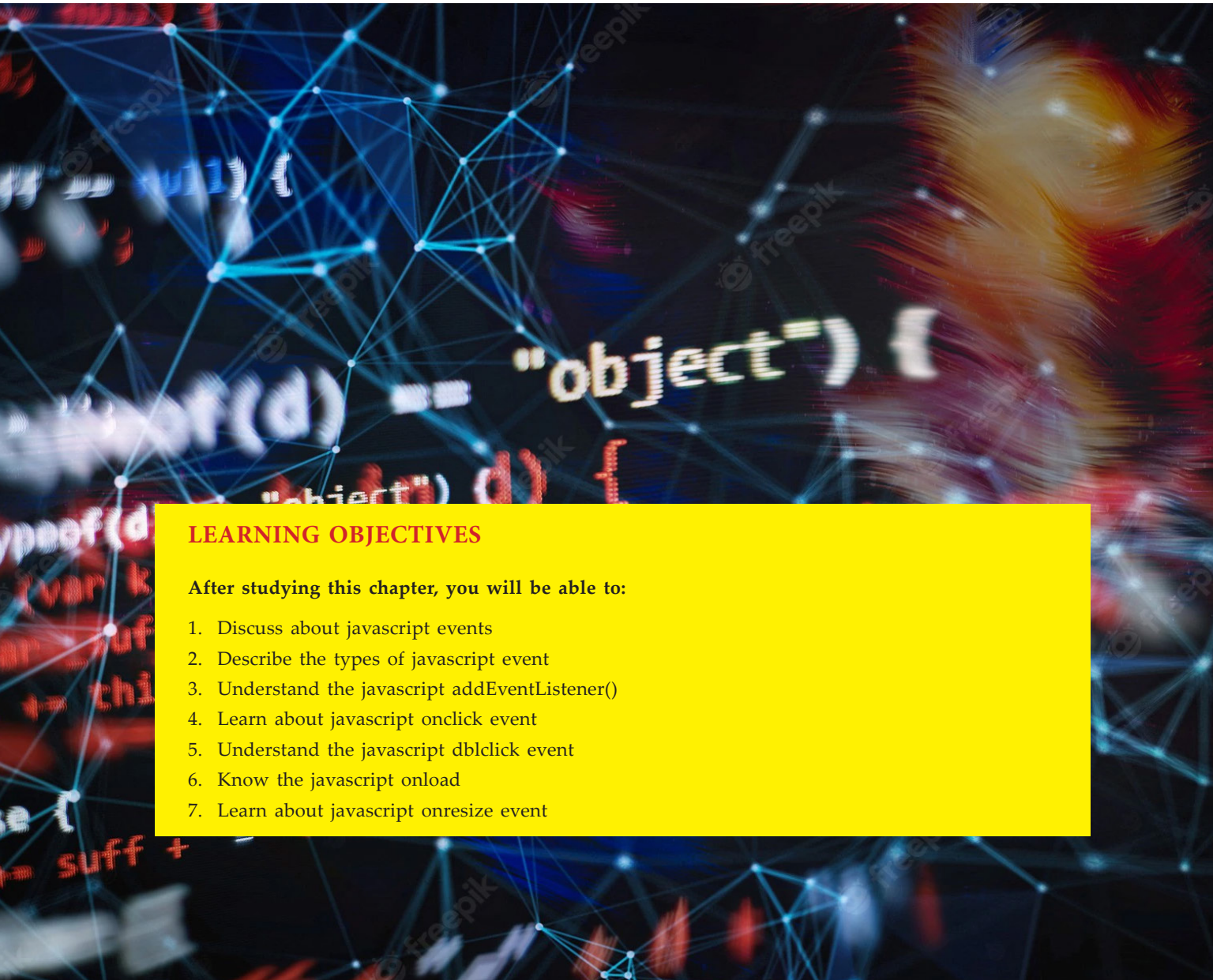
1. What do you understand by the javaserver pages (JSP)?
2. Discuss about the JSP Processing.
3. How to create a simple JSP page?
4. Define the JSP directives.
5. Explain the syntax in JSP.
6. Discuss about JSP the page directive
7. Explain the include directive
8. Focus on the taglib directive

REFERENCES

1. Bergsten, Hans (2003). JavaServer Pages (3rd ed.). O'Reilly Media. ISBN 978-0-596-00563-4.
2. Brown, Simon; Dalton, Sam; Jepp, Daniel; Johnson, Dave; Li, Sing; Raible, Matt. Pro JSP 2. Apress. ISBN 1-59059-513-0.
3. Hanna, Phil (2003). JSP 2.0 - The Complete Reference. McGraw-Hill Osborne Media. ISBN 978-0-07-222437-5.
4. Marty Hall, et al., "Core Servlets and JavaServer Pages", vol.1 (2nd eds, 2003) and vol. 2 (2nd eds, 2006), Prentice Hall.

CHAPTER 7

JAVASCRIPT EVENTS



LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Discuss about javascript events
2. Describe the types of javascript event
3. Understand the javascript addEventListener()
4. Learn about javascript onclick event
5. Understand the javascript dblclick event
6. Know the javascript onload
7. Learn about javascript onresize event

INTRODUCTION

The Javascript interacts with the documents HTML code using ***events***, which are triggered when a particular moment of interest happens in the document or the browser window.

Javascript Events makes the webpages interactive and responsive. These events are asynchronous (i.e it can occur anytime). Most events are triggered by user action but there are some exceptions like the event load.

When an event occurs there are some default action which the browser takes (like clicking on a link open up the location specified in attribute *href*).An event may be a click, mouseover, keystroke etc.

Javascript responds to events by calling a function which performs some task as defined in the function.

As of today, Javascript has three event models for programming events: the inline model, the Scripting Model and the DOM2 Model

Javascript can respond to the following type of events: Mouse Actions, Keyboard Actions, Form Actions, Page Loads, Time Intervals and Errors.

JAVASCRIPT EVENTS

The change in the state of an object is known as an Event. In html, there are various events which represents that some activity is performed by the user or by the browser. When javascript code is included in HTML, js react over these events and allow the execution.

This process of reacting over the events is called Event Handling. Thus, js handles the HTML events via Event Handlers. For example, when a user clicks over the browser, add js code, which will execute the task to be performed on the event.

Some of the HTML events and their event handlers are:

Mouse Events

Event Performed	Event Handler	Description
click	onclick	When mouse click on an element
mouseover	onmouseover	When the cursor of the mouse comes over the element
mouseout	onmouseout	When the cursor of the mouse leaves an element
mousedown	onmousedown	When the mouse button is pressed over the element
mouseup	onmouseup	When the mouse button is released over the element
mousemove	onmousemove	When the mouse movement takes place.

Click Event

```
1. <html>
2. <head> Javascript Events </head>
3. <body>
4. <script language="Javascript" type="text/Javascript">
5.     <!--
6.     function clickevent()
7.     {
8.         document.write("This is JavaTpoint");
9.     }
10.    //-->
11. </script>
12. <form>
13. <input type="button" onclick="clickevent()" value="Who's
    this?"/>
14. </form>
15. </body>
16. </html>
```

MouseOver Event

```
1. <html>
2. <head>
3. <h1> Javascript Events </h1>
4. </head>
5. <body>
6. <script language="Javascript" type="text/Javascript">
7.     <!--
8.     function mouseoverevent()
9.     {
10.        alert("This is JavaTpoint");
11.    }
12.    //-->
13. </script>
14. <p onmouseover="mouseoverevent()"> Keep cursor over me</p>
15. </body>
16. </html>
```

- **Keyboard Events**

Event Performed	Event Handler	Description
Keydown & Keyup	onkeydown & onkeyup	When the user press and then release the key

Keydown Event

```
1. <html>
2. <head> Javascript Events</head>
3. <body>
4. <h2> Enter something here</h2>
5. <input type="text" id="input1" onkeydown="keydownevent()"/>
6. <script>
7. <!--
8.     function keydownevent()
```

```
9.      {
10.         document.getElementById("input1");
11.         alert("Pressed a key");
12.     }
13. //-->
14. </script>
15. </body>
16. </html>
```

- **Form Events**

Event Performed	Event Handler	Description
focus	onfocus	When the user focuses on an element
submit	onsubmit	When the user submits the form
blur	onblur	When the focus is away from a form element
change	onchange	When the user modifies or changes the value of a form element

Focus Event

```
1. <html>
2. <head> Javascript Events</head>
3. <body>
4. <h2> Enter something here</h2>
5. <input type="text" id="input1" onfocus="focusevent()"/>
6. <script>
7. <!--
8.      function focusevent()
9.      {
10.             document.getElementById("input1").style.
    background=" aqua";
11.     }
12. //-->
13. </script>
```

14. </body>
15. </html>

- **Window/Document Events**

Event Performed	Event Handler	Description
load	onload	When the browser finishes the loading of the page
unload	onunload	When the visitor leaves the current webpage, the browser unloads it
resize	onresize	When the visitor resizes the window of the browser

Load Event

1. <html>
2. <head>Javascript Events</head>
3. </br>
4. <body onload="window.alert('Page successfully loaded');">
5. <script>
6. <!--
7. document.write("The page is loaded successfully");
8. //-->
9. </script>
10. </body>
11. </html>

JAVASCRIPT EVENT TYPES

These are the top 8 types of JavaScript Event discussed below:

User Interface Events

These occur as the result of any interaction with the browser window rather than the HTML page. In these events, we attach the event listener to the window object, not the document object.

The various UI events are as follows:

- **load**: The load event fires when the webpage finishes loading. It can also fire on nodes of elements like *images, scripts, or objects.*
- **unload**: This event fires before the users leave the page, i.e., the webpage is unloading. Page unloading usually happens because a new page has been requested.
- **error**: This event fires when the browser encounters a JavaScript error or an asset that doesn't exist.
- **resize**: It fires when we resize the browser window. But browsers repeatedly fire this event, so avoid using this event to trigger complicated code; it might make the page less responsive.
- **scroll**: This event fires when the user scrolls up/down on the browser window. It can relate to the entire page or a specific element on the page.

Focus and Blur Events

These events fire when the HTML elements you can interact with gain/ lose focus. They are most commonly used in forms and especially helpful when you want to do the following tasks:

- To show tips or feedback to users as they interact with an element within a form. The tips are usually shown in the elements other than the one the user is interacting with.
- To trigger form validation as a user moves from one control to the next without waiting to submit the form.

The different focus and blur events are as follows:

- **focus**: This event fires, for a specific DOM node, when an element gains focus.
- **blur**: This fires, for a specific DOM node, when an element loses focus.
- **focusin**: This event is the same as the focus event. But Firefox doesn't yet support the focusin event.
- **focusout**: This is the same event as the blur event. This is a new event type in JavaScript, thus not supported in Firefox right now.

The focus and blur events use the capture approach, while the focusin and focusout events use both capture and bubble approach of the event flow.

Mouse Events

These events fire when the mouse moves or the user clicks a button. All the elements of the page support these events and use the bubbling approach. These actions work differently on **touchscreen** devices. Preventing the default behavior of mouse events can cause unexpected results.

The various mouse events of JavaScript are as follows:

- click: This event fires when the user clicks on the primary mouse button (usually the left button). This event also fires if the user presses the Enter key on the keyboard when an element has focus.

  Touch-screen: A tap on the screen acts like a single left-click.
- dblclick: This event fires when the user clicks the primary mouse button, in quick succession, twice.

  Touch-screen: A double-tap on the screen acts like a double left-click.

  Accessibility: You can add the above two events to any element, but it's better to apply it only on the items that are usually clicked, or it will not be accessible through keyboard navigation. All the mouse events discussed below cannot be triggered by the keyboard.
- mousedown: It fires when the user clicks down on any mouse button.

  Touch-screen: You can use the **touchstart** event.
- Mouseup: It fires when the user releases a mouse button.

  Touch-screen: You can use the **touchend** event.

  We have separate **mousedown** and **mouseup** events to add drag-and-drop functionality or controls in game development. Don't forget a **click** event is the combination of **mousedown** and **mouseup** events.

- mouseover: It fires when the user moves the cursor, which was outside an element before, inside the element. We can say that it fires when we move the cursor over the element.
- mouseout: It fires when the user moves the cursor, which was inside an element before, outside the element. We can say that it fires when the cursor moves off the element.

  The mouseover and mouseout events usually change the appearance of graphics on our webpage. A preferred alternative to this is to use the **CSS: hover** pseudo-class.
- mousemove: It fires when the user moves the cursor around the element. This event is frequently triggered.

Keyboard Events

These events fire on any kind of device when a user interacts with a keyboard.

- **input**: This event fires when the value of an **<input>** or a **<textarea>** changes (doesn't fire for deleting in IE9). You can use keydown as a fallback in older browsers.
- **keydown**: It fires when the user presses any key in the keyboard. If the user holds down the key, this event fires repeatedly.
- **keypress**: It fires when the user presses a key that results in printing a character on the screen. This event fires repeatedly if the user holds down the key. This event will not fire for the enter, tab, or arrow keys; the **keydown** event would.
- **keyup**: The keyup event fires when the user releases a key on the keyboard.

The keydown and keypress events fire before a character appears on the screen, the keyup fires after it shows.

To know the key pressed when you use the keydown and keypress events, the event object has a **keyCode** property. This property, instead of returning the letter for that key, returns the ASCII code of the lowercase for that key.

Form Events

These events are common while using forms on a **webpage**. In particular, we see the submit event mostly in form of validation (checking form values). As described in our tutorial; *Features of JavaScript*, if the users miss any required information or enter incorrect input, validation before sending the data to the server is faster.

The list below explains the different form of events available to the user:

- **submit**: This event fires on the node representing the **<form>** element when a user submits a form.
- **change**: It fires when the status of various form elements change. This is a better option than using the click event because clicking is not the only way users interact with the form.
- **input**: The input event is very common with the <input> and the <textarea> elements.

We often use the focus and blur events with forms, but they are also available in conjunction with other elements like links.

Mutation Events and Observers

Whenever the structure of the DOM tree changes, it triggers a mutation event. The change in the tree may be due to the addition or removal of a DOM node through your script. But these have an alternative that will replace them: mutation observers. The following are the numerous mutation events in JavaScript.

- **DOMNodeInserted**: It fires when the script inserts a new node in the DOM tree using *appendChild(), replaceChild(), insertBefore(), etc.*
- **DOMNodeRemoved**: This event fires when the script removes an existing node from the tree using *removeChild(), replaceChild(), etc.*
- **DOMSubtreeModified**: It fires when the structure of the DOM tree changes i.e. the above two events occur.
- **DOMNodeInsertedIntoDocument**: This event fires when the script inserts a node in the DOM tree as the descendant of another node already in the document.

- **DOMNodeRemovedFromDocument**: This event fires when the script removes a node from the DOM tree as the descendant of another node already in the document.

The problem with the mutation events is that lots of changes to your page can make your page feel slow or unresponsive. These can also trigger other event listeners, modifying DOM and leading to more mutation events firing. This is the reason for introducing mutation observers to the script. Mutation observers wait until the script finishes its current task before reacting, then reports the changes in a batch (not one at a time). This reduces the number of events that fire when you change the DOM tree through your script. You can also specify which changes in the DOM you want them to react to.

HTML5 Events

These are the page-level events included in the versions of the HTML5 specialization. New events support more recent devices like phones and tablets. They respond to events such as gestures and movements. You will understand them better after you master the above concepts, thus they are not discussed for now. Work with the events below for now and when you are a better developer, you can search for other events available.

The three HTML5 events we will learn are as follows:

- **DOMContentLoaded**: This event triggers when the DOM tree forms i.e. the script is loading. Scripts start to run before all the resources like *images, CSS, and JavaScript loads*. You can attach this event either to the **window** or the **document** objects.
- **hashchange**: It fires when the URL hash changes without refreshing the entire window. Hashes (#) link specific parts (known as **anchors**) within a page. It works on the **window** object; the event object contains both the **oldURL** and the **newURL** properties holding the URLs before and after the hashchange.
- **beforeunload**: This event fires on the window object just before the page unloads. This event should only be helpful for the user, not encouraging them to stay on the page. You can add a dialog box to your event, showing a message alerting the users like their changes are not saved.

CSS Events

These events trigger when the script encounters a CSS element. As CSS is a crucial part of web development, the developers decided to add these events to js to make working with CSS easier. Some of the most common CSS events are as follows:

- **transitionend**: This event fires when a CSS transition ends in a program. It is useful to notify the script of the end of transition so that it can take further action.
- **animationstart**: These events fire when CSS animation starts in the program.
- **animationiteration**: This event occurs when any CSS animation repeats itself. With this event, we can determine the number of times an animation iterates in the script.
- **animationend**: It fires when the CSS animation comes to an end in the program. This is useful when we want to act just after the **animation** process finishes.

JAVASCRIPT ADDEVENTLISTENER()

The **addEventListener()** method is used to attach an **event handler** to a particular element. It does not override the existing event handlers. Events are said to be an essential part of the JavaScript. A web page responds according to the event that occurred. Events can be user-generated or generated by API's. An event listener is a JavaScript's procedure that waits for the occurrence of an event.

The addEventListener() method is an inbuilt function of JavaScript. We can add multiple event handlers to a particular element without overwriting the existing event handlers.

Syntax

element.addEventListener(event, function, useCapture);

Although it has three parameters, the parameters ***event*** and ***function*** are widely used. The third parameter is optional to define. The values of this function are defined as follows.

Parameter Values

Event: It is a required parameter. It can be defined as a string that specifies the event's name.

function: It is also a required parameter. It is a JavaScript function which responds to the event occur.

useCapture: It is an optional parameter. It is a Boolean type value that specifies whether the event is executed in the bubbling or capturing phase. Its possible values are **true** and **false**. When it is set to true, the event handler executes in the capturing phase. When it is set to false, the handler executes in the bubbling phase. Its default value is **false**.

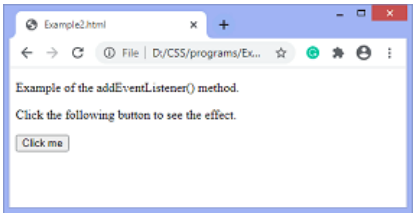
Let's see some of the illustrations of using the addEventListener() method.

Example

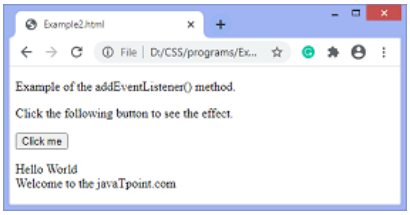
It is a simple example of using the addEventListener() method. We have to click the given HTML button to see the effect.

```
1. <!DOCTYPE html>
2. <html>
3. <body>
4. <p> Example of the addEventListener() method. </p>
5. <p> Click the following button to see the effect. </p>
6. <button id = "btn"> Click me </button>
7. <p id = "para"></p>
8. <script>
9. document.getElementById("btn").
   addEventListener("click", fun);
10. function fun() {
11. document.getElementById("para").innerHTML = "Hello
    World" + "<br>" + "Welcome to the  javaTpoint.com";
12. }
13. </script>
14. </body>
15. </html>
```

Output



After clicking the given HTML button, the output will be -



Now, in the next example we will see how to add many events to the same element without overwriting the existing events.

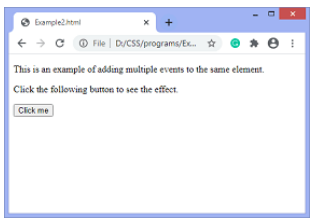
Example

In this example, we are adding multiple events to the same element.

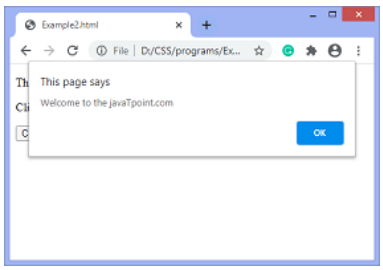
```
1. <!DOCTYPE html>
2. <html>
3. <body>
4. <p> This is an example of adding multiple events to the same element. </p>
5. <p> Click the following button to see the effect. </p>
6. <button id = "btn"> Click me </button>
7. <p id = "para"></p>
8. <p id = "para1"></p>
9. <script>
10. function fun() {
11.     alert("Welcome to the javaTpoint.com");
12. }
```

```
13.
14. function fun1() {
15.    document.getElementById("para").innerHTML = "This
    is second function";
16.
17. }
18. function fun2() {
19.       document.getElementById("para1").innerHTML =
    "This is third function";
20. }
21. var mybtn = document.getElementById("btn");
22. mybtn.addEventListener("click", fun);
23. mybtn.addEventListener("click", fun1);
24. mybtn.addEventListener("click", fun2);
25. </script>
26. </body>
27. </html>
```

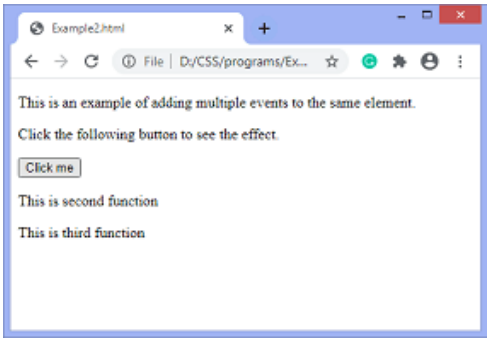
Output



Now, when we click the button, an alert will be displayed. After clicking the given HTML button, the output will be -



When we exit the alert, the output is -



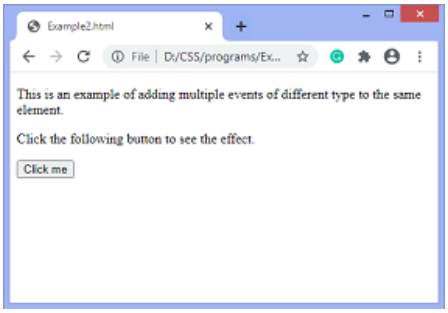
Example

In this example, we are adding multiple events of a different type to the same element.

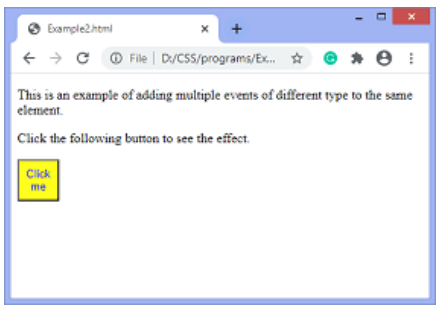
```
1.  <!DOCTYPE html>
2.  <html>
3.  <body>
4.  <p> This is an example of adding multiple events of different type to the same element. </p>
5.  <p> Click the following button to see the effect. </p>
6.  <button id = "btn"> Click me </button>
7.  <p id = "para"></p>
8.  <script>
9.  function fun() {
10.     btn.style.width = "50px";
11.     btn.style.height = "50px";
12.     btn.style.background = "yellow";
13.     btn.style.color = "blue";
14. }
15.
16. function fun1() {
17.   document.getElementById("para").innerHTML = "This is second function";
18.
19. }
20. function fun2() {
```

```
21.     btn.style.width = "";
22.     btn.style.height = "";
23.     btn.style.background = "";
24.     btn.style.color = "";
25. }
26. var mybtn = document.getElementById("btn");
27. mybtn.addEventListener("mouseover", fun);
28. mybtn.addEventListener("click", fun1);
29. mybtn.addEventListener("mouseout", fun2);
30. </script>
31. </body>
32. </html>
```

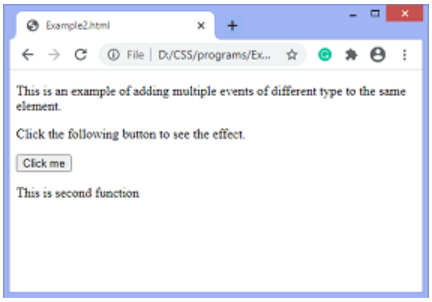
Output



When we move the cursor over the button, the output will be -



After clicking the button and leave the cursor, the output will be -



Event Bubbling or Event Capturing

Now, we understand the use of the third parameter of JavaScript's addEventListener(), i.e., ***useCapture.***

In HTML DOM, Bubbling and Capturing are the two ways of event propagation. We can understand these ways by taking an example.

Suppose we have a div element and a paragraph element inside it, and we are applying the **"click"** event to both of them using the **addEventListener()** method. Now the question is on clicking the paragraph element, which element's click event is handled first.

So, in Bubbling, the event of paragraph element is handled first, and then the div element's event is handled. It means that in bubbling, the inner element's event is handled first, and then the outermost element's event will be handled.

In Capturing the event of div element is handled first, and then the paragraph element's event is handled. It means that in capturing the outer element's event is handled first, and then the innermost element's event will be handled.

addEventListener(event, function, useCapture);

We can specify the propagation using the ***useCapture*** parameter. When it is set to false (which is its default value), then the event uses bubbling propagation, and when it is set to true, there is the capturing propagation.

We can understand the *bubbling* and *capturing* using an illustration.

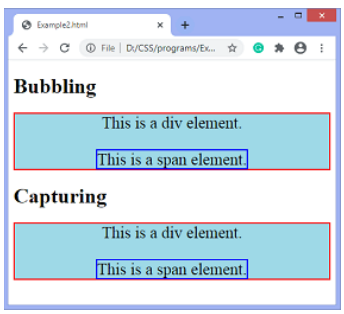
Example

In this example, there are two div elements. We can see the bubbling effect on the first div element and the capturing effect on the second div element. When we double click the span element of the first div element, then the span element's event is handled first than the div element. It is called *bubbling*. But when we double click the span element of the second div element, then the div element's event is handled first than the span element. It is called *capturing*.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. div{
6. background-color: lightblue;
7. border: 2px solid red;
8. font-size: 25px;
9. text-align: center;
10. }
11. span{
12. border: 2px solid blue;
13. }
14. </style>
15. </head>
16. <body>
17. <h1> Bubbling </h1>
18. <div id = "d1">
19. This is a div element.
20. <br><br>
21. <span id = "s1"> This is a span element. </span>
22. </div>
23. <h1> Capturing </h1>
24. <div id = "d2"> This is a div element.
25. <br><br>
```

```
26. <span id = "s2"> This is a span element. </span>
27. </div>
28.
29. <script>
30. document.getElementById("d1").
addEventListener("dblclick", function() {alert('You have
double clicked on div element')}, false);
31. document.getElementById("s1").
addEventListener("dblclick", function() {alert('You have
double clicked on span element')}, false);
32. document.getElementById("d2").
addEventListener("dblclick", function() {alert('You have
double clicked on div element')}, true);
33. document.getElementById("s2").
addEventListener("dblclick", function() {alert('You have
double clicked on span element')}, true);
34. </script>
35. </body>
36. </html>
```

Output



We have to double click the specific elements to see the effect.

JAVASCRIPT ONCLICK EVENT

The onclick event generally occurs when the user clicks on an element. It allows the programmer to execute a JavaScript's function when an element gets clicked. This event can be used for validating a form, warning messages and many more.

Using JavaScript, this event can be dynamically added to any element. It supports all HTML elements except <html>, <head>, <title>, <style>, <script>, <base>, <iframe>, <bdo>, <br>, <meta>, and <param>. It means we cannot apply the onclick event on the given tags. In HTML, we can use the onclick attribute and assign a JavaScript function to it. We can also use the JavaScript's addEventListener() method and pass a click event to it for greater flexibility.

Syntax

Now, we see the syntax of using the **onclick** event in HTML and in javascript (without **addEventListener()** method or by using the **addEventListener()** method).

In HTML

<element onclick = "fun()"**>**

In JavaScript

object.onclick = function() { myScript };

In JavaScript by using the addEventListener() method

object.addEventListener("click", myScript);

Let's see how to use **onclick** event by using some illustrations. Now, we will see the examples of using the **onclick** event in HTML, and in JavaScript.

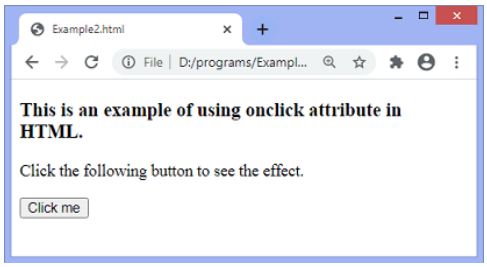
Example1 - Using onclick attribute in HTML

In this example, we are using the HTML **onclick** attribute and assigning a JavaScript's function to it. When the user clicks the given button, the corresponding function will get executed, and an alert dialog box will be displayed on the screen.

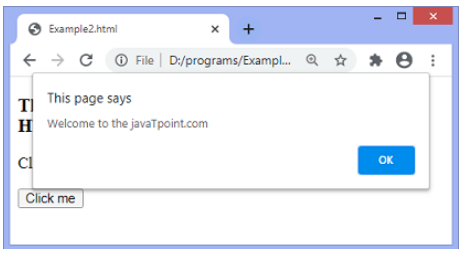
```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <script>
5. function fun() {
```

```
6. alert("Welcome to the javaTpoint.com");
7. }
8. </script>
9. </head>
10. <body>
11. <h3> This is an example of using onclick attribute in HTML.
    </h3>
12. <p> Click the following button to see the effect. </p>
13. <button onclick = "fun()">Click me</button>
14. </body>
15. </html>
```

Output



After clicking the given button, the output will be -



Example2 - Using JavaScript

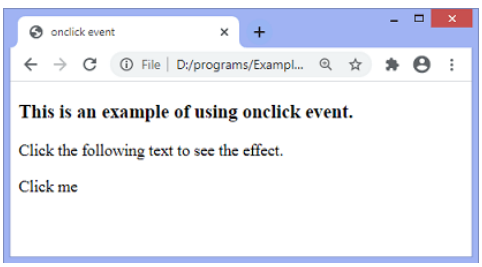
In this example, we are using JavaScript's **onclick** event. Here we are using the **onclick** event with the paragraph element.

When the user clicks on the paragraph element, the corresponding function will get executed, and the text of the

paragraph gets changed. On clicking the **<p>** element, the background color, size, border, and color of the text will also get change.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <title> onclick event </title>
5. </head>
6. <body>
7. <h3> This is an example of using onclick event. </h3>
8. <p> Click the following text to see the effect. </p>
9. <p id = "para">Click me</p>
10. <script>
11. document.getElementById("para").onclick = function() {
12. fun()
13. };
14. function fun() {
15. document.getElementById("para").innerHTML =
    "Welcome to the javaTpoint.com";
16. document.getElementById("para").style.color = "blue";
17. document.getElementById("para").style.backgroundColor
    = "yellow";
18. document.getElementById("para").style.fontSize = "25px";
19. document.getElementById("para").style.border = "4px
    solid red";
20. }
21. </script>
22.
23. </body>
24. </html>
```

Output



After clicking the text ***Click me,*** the output will be -



Example3 - Using addEventListener() method

In this example, we are using JavaScript's **addEventListener()** method to attach a **click** event to the paragraph element. When the user clicks the paragraph element, the text of the paragraph gets changed.

On clicking the paragraph, the background color and font-size of elements will also change.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. </head>
5. <body>
6. <h3> This is an example of using click event. </h3>
7. <p> Click the following text to see the effect. </p>
8. <p id = "para">Click me</p>
9. <script>
10. document.getElementById("para").onclick = function() {
11. fun()
12. };
13. function fun() {
14. document.getElementById("para").innerHTML =
    "Welcome to the javaTpoint.com";
15. document.getElementsByTagName("body")[0].style.color =
    "blue";
16. document.getElementsByTagName("body")[0].style.
    backgroundColor = "lightgreen";
```

```
17. document.getElementsByTagName("body")[0].style.
    fontSize = "25px";
18. document.getElementById("para").style.border = "4px
    solid red";
19. }
20. </script>
21.
22. </body>
23. </html>
```

Output



On clicking the text ***Click me***, the output will be -



JAVASCRIPT DBLCLICK EVENT

The dblclick event generates an event on double click the element. The event fires when an element is clicked twice in a very short span of time. We can also use the JavaScript's addEventListener() method to fire the double click event.

In HTML, we can use the ondblclick attribute to create a double click event.

Syntax

Now, we see the syntax of creating double click event in HTML and in javascript (without using addEventListener() method or by using the addEventListener() method).

In HTML

<element ondblclick = "fun()"**>**

In JavaScript

object.ondblclick = function() { myScript };

In JavaScript by using the addEventListener() method

object.addEventListener("dblclick", myScript);

Let's see some of the illustrations to understand the double click event.

Example - Using ondblclick attribute in HTML

In this example, we are creating the double click event using the HTML **ondblclick** attribute.

1. <!DOCTYPE html>
2. **<html>**
3. **<head>**
4. **</head>**
5.
6. **<body>**
7. **<h1** id = "heading" ondblclick = "fun()"**>** Hello world :):) **</h1>**
8. **<h2>** Double Click the text "Hello world" to see the effect. **</h2>**
9. ** ** This is an example of using the **** ondblclick **** attribute. ** **
10. **<script>**

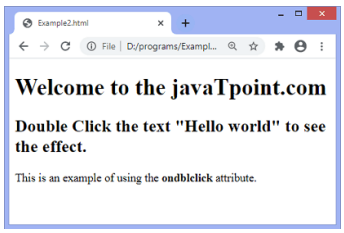
```
11. function fun() {
12. document.getElementById("heading").innerHTML = "
    Welcome to the javaTpoint.com ";
13. }
14. </script>
15. </body>
16. </html>
```

Output

After the execution of the above code, the output will be -



After double-clicking the text **"Hello world"**, the output will be -



Now, we will see how to create double click event using JavaScript.

Example - Using JavaScript

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. </head>
5.
```

```
6. <body>
7. <h1 id = "heading"> Hello world :):) </h1>
8. <h2> Double Click the text "Hello world" to see the effect.
   </h2>
9. <p> This is an example of creating the double click event
   using JavaScript. </p>
10. <script>
11. document.getElementById("heading").ondblclick =
    function() { fun() };
12. function fun() {
13. document.getElementById("heading").innerHTML = "
    Welcome to the javaTpoint.com ";
14. }
15. </script>
16. </body>
17.
18. </html>
```

Output



After double-clicking the text **"Hello world"**, the output will be -



Example - Using JavaScript's addEventListener() method

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. </head>
5.
6. <body>
7. <h1 id = "heading"> Hello world :):) </h1>
8. <h2> Double Click the text "Hello world" to see the effect.
   </h2>
9. <p> This is an example of creating the double click event
   using the <b> addEventListener() method </b>. </p>
10. <script>
11. document.getElementById("heading").
   addEventListener("dblclick", fun);
12. function fun() {
13. document.getElementById("heading").innerHTML = "
   Welcome to the javaTpoint.com ";
14. }
15. </script>
16. </body>
17.
18. </html>
```

Output



After double-clicking the text **"Hello world"**, the output will be -



JAVASCRIPT ONLOAD

In JavaScript, this event can apply to launch a particular function when the page is fully displayed. It can also be used to verify the type and version of the visitor's browser. We can check what cookies a page uses by using the onload attribute.

In HTML, the onload attribute fires when an object has been loaded. The purpose of this attribute is to execute a script when the associated element loads.

In HTML, the onload attribute is generally used with the **<body>** element to execute a script once the content (including CSS files, images, scripts, etc.) of the webpage is completely loaded. It is not necessary to use it only with <body> tag, as it can be used with other HTML elements.

The difference between the **document.onload** and **window.onload** is: **document.onload** triggers before the loading of images and other external content. It is fired before the **window.onload**. While the **window.onload** triggers when the entire page loads, including CSS files, script files, images, etc.

Syntax

```
window.onload = fun()
```

Let's understand this event by using some examples.

Example1

In this example, there is a div element with a height of 200px and a width of 200px. Here, we are using the **window.onload()**

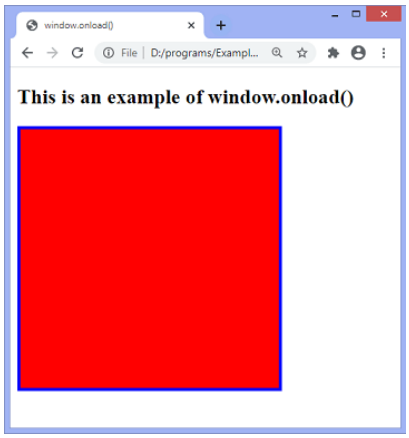
to change the background color, width, and height of the **div** element after loading the web page.

The background color is set to **'red'**, and width and height are set to **300px** each.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <meta charset = " utf-8">
5. <title> window.onload() </title>
6. <style type = "text/css">
7. #bg{
8. width: 200px;
9. height: 200px;
10. border: 4px solid blue;
11. }
12. </style>
13. <script type = "text/javascript">
14. window.onload = function(){
15. document.getElementById("bg").style.backgroundColor =
    "red";
16. document.getElementById("bg").style.width = "300px";
17. document.getElementById("bg").style.height = "300px";
18. }
19. </script>
20. </head>
21. <body>
22. <h2> This is an example of window.onload() </h2>
23. <div id = "bg"></div>
24. </body>
25. </html>
```

Output

After the execution of the code and loading of the page, the output will be -



Example2

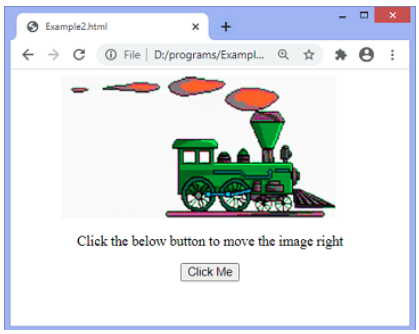
In this example, we are implementing a simple animation by using the properties of the DOM object and functions of javascript. We use the JavaScript function getElementById() for getting the DOM object and then assign that object into a global variable.

```
1. <html>
2.   <head>
3.     <script type = "text/javascript">
4.
5.         var img = null;
6.         function init(){
7.           img = document.getElementById('myimg');
8.           img.style.position = 'relative';
9.           img.style.left = '50px';
10.        }
11.        function moveRight(){
12.          img.style.left = parseInt(
13.          img.style.left) + 100 + 'px';
14.        }
```

```
15.          window.onload = init;
16.
17.     </script>
18.   </head>
19.
20.   <body>
21.     <form>
22.       <img id = “myimg” src = “train1.png” />
23.       <center>
24.         <p>Click the below button to move the image right</
    p>
25.         <input type = “button” value = “Click Me” onclick =
    “moveRight();” />
26.     </center>
27.     </form>
28.   </body>
29.
30. </html>
```

Output

After the successful execution of the above code, the output will be -



Now, there is an example in which we will use the HTML **onload** attribute and the JavaScript functions.

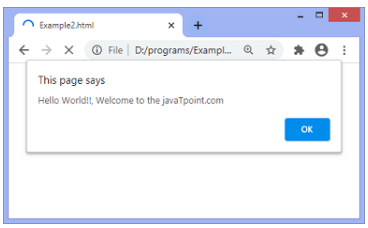
Example3

It is a simple example of using the HTML **onload** attribute with the function defined in JavaScript. In this example, the **alert()** function gets called whenever the document refresh.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. <style>
5. </style>
6. <script>
7.    function fun() {
8.       alert("Hello World!!, Welcome to the javaTpoint.com");
9.    }
10. </script>
11. </head>
12. <body onload = "fun()">
13. <h1> Example of the HTML onload attribute </h1>
14. <p> Try to refresh the document to see the effect. </p>
15. </body>
16. </html>
```

Output

After the execution of the above code, the output will be -



JAVASCRIPT ONRESIZE EVENT

The onresize event in JavaScript generally occurs when the window has been resized. To get the size of the window, we

can use the JavaScript's *window.outerWidth* and *window.outerHeight* events. We can also use the JavaScript's properties such as *innerWidth, innerHeight, clientWidth, ClientHeight, offsetWidth, offsetHeight* to get the size of an element.

In HTML, we can use the onresize attribute and assign a JavaScript function to it. We can also use the JavaScript's addEventListener() method and pass a resize event to it for greater flexibility.

Syntax

Now, we see the syntax of using the **onresize** event in HTML and in javascript (without **addEventListener()** method or by using the **addEventListener()** method).

In HTML

<element onresize = "fun()"**>**

In JavaScript

object.onresize = function() { myScript };

In JavaScript by using the addEventListener() method

object.addEventListener("resize", myScript);

Let's see some of the illustrations to understand the **onresize** event.

Example

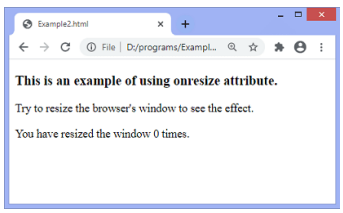
In this example, we are using the HTML **onresize** attribute. Here, we are using the ***window.outerWidth*** and ***window.outerHeight*** events of JavaScript to get the height and width of the window. When the user resizes the window, the updated width and height of the window will be displayed on the screen. It will also display how many times the user tried to resize the window. When we change the height of the window, the updated height will change accordingly. Similarly, when we change the width of the window, the updated width will change accordingly.

1. <!DOCTYPE html>
2. **<html>**

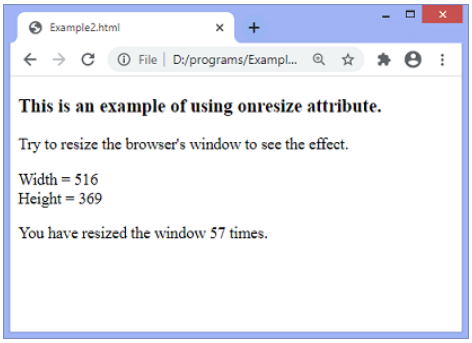
```
3. <head>
4. <script>
5. var i = 0;
6.
7. function fun() {
8. var res = “Width = “ + window.outerWidth + “<br>” + “Height = “ + window.outerHeight;
9. document.getElementById(“para”).innerHTML = res;
10.
11. var res1 = i += 1;
12. document.getElementById(“s1”).innerHTML = res1;
13. }
14. </script>
15. </head>
16. <body onresize = “fun()”>
17. <h3> This is an example of using onresize attribute. </h3>
18. <p> Try to resize the browser’s window to see the effect. </p>
19.
20. <p id = “para”> </p>
21. <p> You have resized the window <span id = “s1”> 0 </span> times.</p>
22. </body>
23. </html>
```

Output

After the execution of the above code, the output will be -



When we try to resize the window, the output will be: -



Example - Using JavaScript

In this example, we are using JavaScript's **onresize** event.

1. <!DOCTYPE html>
2. **<html>**
3. **<head>**
4. **</head>**
5. **<body>**
6. **<h3>** This is an example of using JavaScript's onresize event. **</h3>**
7. ** ** Try to resize the browser's window to see the effect. ** **
8.
9. **<p** id = "para"> ** **
10. ** ** You have resized the window **<span** id = "s1"> 0 **** times.** **
11. **<script>**
12. document.getElementsByTagName("BODY")[0].onresize = function() {fun()};
13. var i = 0;
14.
15. function fun() {
16. var res = "Width = " + window.outerWidth + "**
**" + "Height = " + window.outerHeight;
17. document.getElementById("para").innerHTML = res;

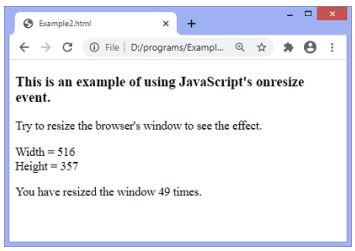
```
18.
19. var res1 = i += 1;
20. document.getElementById("s1").innerHTML = res1;
21. }
22. </script>
23. </body>
24. </html>
```

Output

After the execution of the above code, the output will be -



When we try to resize the window, the output will be -



Example - Using addEventListener() method

In this example, we are using JavaScript's **addEventListener()** method.

```
1. <!DOCTYPE html>
2. <html>
3. <head>
4. </head>
5. <body>
6. <h3> This is an example of using JavaScript's addEventListener() method. </h3>
```

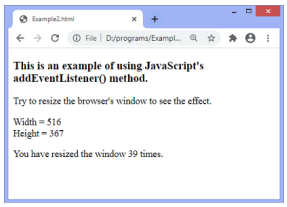
```
7. <p> Try to resize the browser's window to see the effect. </
   p>
8.
9. <p id = "para"> </p>
10. <p> You have resized the window <span id = "s1"> 0 </span>
    times.</p>
11. <script>
12. window.addEventListener("resize", fun);
13. var i = 0;
14.
15. function fun() {
16. var res = "Width = " + window.outerWidth + "<br>" +
    "Height = " + window.outerHeight;
17. document.getElementById("para").innerHTML = res;
18.
19. var res1 = i += 1;
20. document.getElementById("s1").innerHTML = res1;
21. }
22. </script>
23. </body>
24. </html>
```

Output

After the execution of the above code, the output will be -



When we try to resize the window, the output will be -



KEY TERMS

Touchscreen	Webpage	Animation
Event handler	Event bubbling	JavaScript Event
Browser window	Mutation event	Trigger

DISCUSSION QUESTIONS

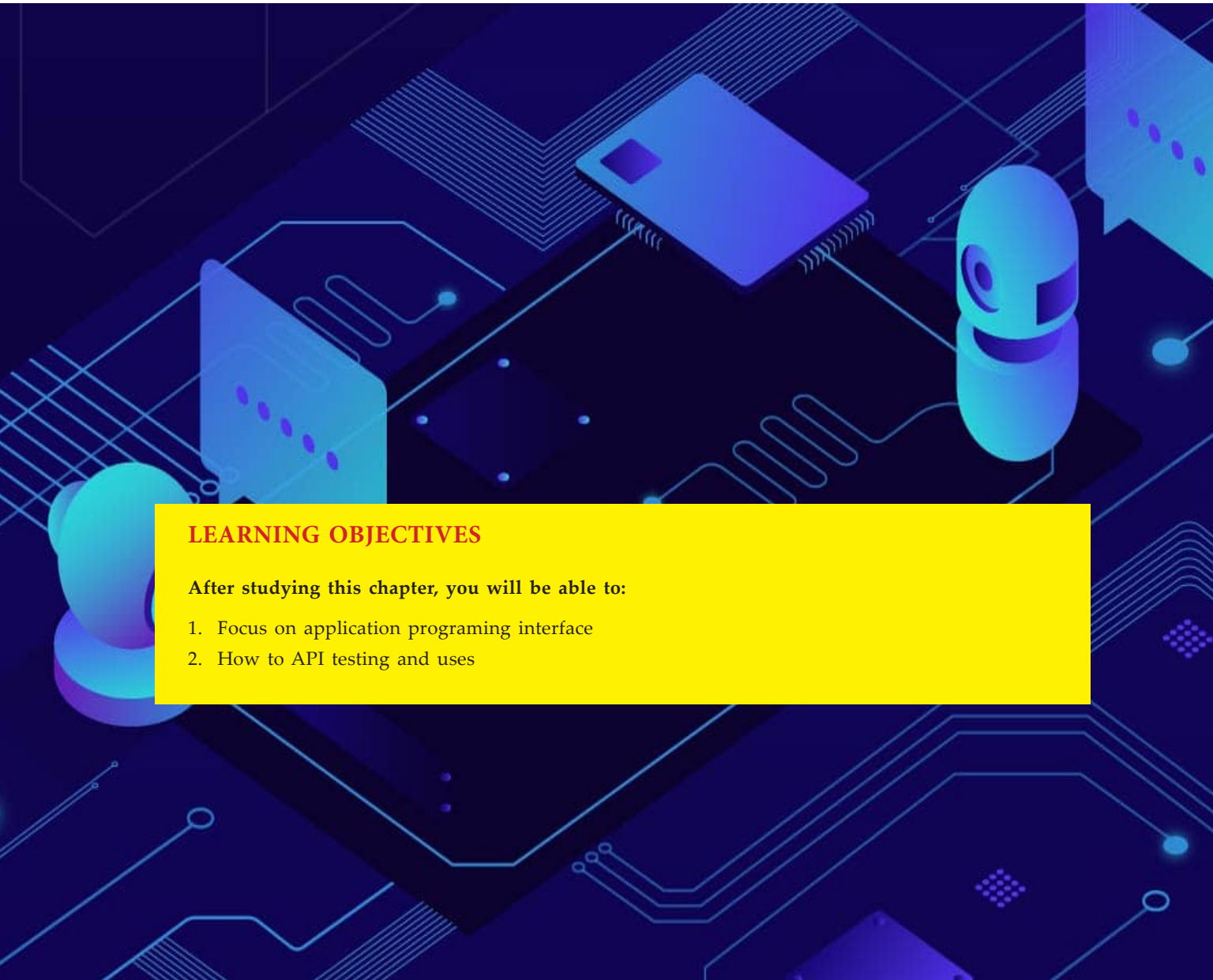
1. Describe the types of JavaScript event.
2. What does addEventListener do in JavaScript?
3. What is onclick event in JavaScript?
4. What is dblclick in JavaScript?
5. Distinguish between event bubbling and event capturing.

REFERENCES

1. Enzer, Larry (31 August 2018). "The Evolution of the Web Browsers". Monmouth Web Developers. Retrieved 31 August 2018.
2. Fedortsova, Irina (June 2012). "Concurrency in JavaFX". JavaFX Documentation Home. Oracle. Retrieved 4 January 2018.
3. Gillies, James; Cailliau, R. (2000). How the Web was Born: The Story of the World Wide Web. Oxford University Press. pp. 6. ISBN 0192862073.
4. Mössenböck, Hanspeter (2002-03-25). "Advanced C#: Variable Number of Parameters" (PDF). Institut für Systemsoftware, Johannes Kepler Universität Linz, Fachbereich Informatik. p. 26. Retrieved 2011-08-05.
5. Samek, Miro (11 March 2009). "State Machines for Event-Driven Systems". Retrieved 19 March 2013.
6. Stewart, William. "Web Browser History". Archived from the original on 20 January 2011.
7. Vivek Gupta, Ethan Jackson, Shaz Qadeer and Sriram Rajamani (November 2012). "P: Safe Asynchronous Event-Driven Programming". Microsoft Research. Retrieved 20 February 2017.

CHAPTER 8

WEB API ESSENTIALS



LEARNING OBJECTIVES

After studying this chapter, you will be able to:

1. Focus on application programing interface
2. How to API testing and uses

INTRODUCTION

A Web API is an application programming interface for either a web server or a web browser. It is a web development concept, usually limited to a web application's client-side (including any web frameworks being used), and thus usually does not include web server or browser implementation details such as SAPIs or APIs unless publicly accessible by a remote web application. An application program interface is not a user interface, but a software-to-software interface. In fact, it allows programs to communicate with each other without any user knowledge or intervention. For example, when you buy concert tickets online with a credit card, the website uses an API to send your card data to a remote application that checks its validity. Once the payment has been confirmed, the application responds to the website giving it the okay to issue the tickets.

Application program interfaces designed to provide the fastest execution usually contain sets of procedures, functions, variables, and data structures. A complete API is often defined as one that provides an abstraction layer that prevents the developer from having to know how its function relates to the lower abstraction levels. This allows the programmer to enhance or redesign the functions within the API without breaking the code that needs it.

An application program interface is quite similar to Software-as-a-Service (SaaS). This is mainly because programmers do not have to start from scratch every time they write an application. Rather than scripting one core program designed to handle everything, the same program can outsource certain responsibilities to other applications capable of doing the job much better. This makes the API very useful for integrating with a website and various web applications. The application program interface has become an integral part of Web 2.0 and the social networking phenomenon. Several popular communities use it to share content under an open architecture. Users can share comments, photos, videos, and all sorts of information with outside applications, utilizing an open API to deliver new functionality to their community. Two classic examples of APIs

that enable this sharing are the Facebook Application Platform and Open Social platform. When developed precisely and used correctly, these can benefit both the company providing the service and its users. Perhaps the biggest downside to APIs is that they are dependent on the providing service.

APPLICATION PROGRAMING INTERFACE

Application program interface (API) is a set of routines, protocols, and tools for building software applications. An API specifies how software components should interact. Additionally, APIs are used when programming graphical user interface (GUI) components. An application programming interface or API is a set of data structures, protocols, routines and tools for accessing a web-based software application. It provides all the building blocks for developing programs with ease. An application programming interface may be language dependent or language independent. A language dependent API is available only in a particular programming language. Language independent APIs are not bound to a particular language, system or process. They can be called from several programming languages.

APIs run silently in the background. They are completely invisible to software users and web surfers. They're primary role is to provide a channel for applications to work with each other to make sure the user gets the required functionality and information.

For example, when a user buys a product online and enters his or her credit card information, the product website uses an API to send the credit card information to another application. The application verifies whether the information is correct or not. It then authorizes the product website to issue the purchased product after the information is verified and payment is confirmed. A website has many such applications that work together using APIs.

Application programming interfaces ensure that software developers do not have to start building applications from scratch. There is no need to build one application that performs every function. The application can transfer certain responsibilities to a remote application. Thus, APIs are similar to SaaS (Software as a Service).

Language Used

An API can be:

- Language-dependent, meaning it is only available by using the syntax and elements of a particular language, which makes the API more convenient to use.
- Language-independent, written so that it can be called from several programming languages. This is a desirable feature for a service-oriented API that is not bound to a specific process or system and may be provided as remote procedure calls or web services. *For example*, a website that allows users to review local restaurants is able to layer their reviews over maps taken from Google Maps, because Google Maps has an API that facilitates this functionality. Google Maps' API controls what information a third-party site can use and how they can use it.

The term API may be used to refer to a complete interface, a single function, or even a set of APIs provided by an organization. Thus, the scope of meaning is usually determined by the context of usage.

Explanation

An API may describe the ways in which a particular task is performed. In procedural languages like C language the action is usually mediated by a function call. Hence the API usually includes a description of all the functions/routines it provides. For instance: the math.h include file for the C language contains the definition of the function prototypes of the mathematical functions available in the C language library for mathematical processing (usually called libm). This file describes how to use the functions included in the given library: the function prototype is a signature that describes the number and types of the parameters to be passed to the functions and the type of the return value. The behavior of the functions is usually described in more details in a human readable format in printed books or in electronic formats like the man pages: e.g. on Unix systems the command man 3 sqrt will present the signature of the function sqrt in the form:

SYNOPSIS

#include <math.h>

double sqrt(double X);

float sqrtf(float X);

DESCRIPTION

DESCRIPTION

sqrt computes the positive square root of the argument. ...

RETURNS

On success, the square root is returned. If X is real and

positive...

That means that the function returns the square root of a positive floating point number (single or double precision) as another floating point number. Hence the API in this case can be interpreted as the collection of the include files used by the C language and its human readable description provided by the man pages.

Documentation

Many program development environments provide the documentation associated with an API in some digital format,

e.g. perl comes with the tool perldoc:

$ perldoc -f sqrt

sqrt EXPR

sqrt #Return the square root of EXPR. If EXPR is omitted,

returns

#square root of $_. Only works on non-negative

operands, unless

#you've loaded the standard Math::Complex module.

python comes with the tool pydoc:

```
$ pydoc math.sqrt

Help on built-in function sqrt in math:

    math.sqrt = sqrt(...)

        sqrt(x)
        Return the square root of x.
```

ruby comes with the tool ri:

```
 $ ri Math::sqrt

------------------------------------------------------------ Math::sqrt

 Math.sqrt(numeric) => float

------------------------------------------------------------------------

 Returns the non-negative square root of _numeric_.
```

Java comes with the documentation organized in HTML pages (JavaDoc format), while Microsoft distributes the API documentation for its languages (Visual C++, C#, Visual Basic, F#, etc...) embedded in Visual Studio's help system.

API in Object-oriented Languages

In object-oriented languages, an API usually includes a description of a set of class definitions, with a set of behaviors associated with those classes. This abstract concept is associated with the real functionality exposed, or made available, by the classes that are implemented in terms of class methods (or more generally by all its public components hence all public methods, but also possibly including any internal entity made public, like fields, constants, nested objects, enums...). The API in this case can be conceived as the totality of all the methods publicly exposed by the classes (usually called the class interface). This means that the API prescribes the methods by which one interacts with/handles the objects derived from the class definitions. More generally, one can see the API as the collection of all the kinds of objects one can derive from the class definitions, and their associated possible behaviors. Again: the use is mediated by the public methods, but in this interpretation, the methods are seen as a technical detail of how the behavior is implemented. For instance: a class representing a Stack can simply expose publicly

two methods push() (to add a new item to the stack), and pop() (to extract the last item, ideally placed on top of the stack).

In this case the API can be interpreted as the two methods pop() and push(), or, more generally, as the idea that one can use an item of type Stack that implements the behavior of a stack: a pile exposing its top to add/remove elements. The second interpretation appears more appropriate in the spirit of object orientation. This concept can be carried to the point where a class interface in an API has no methods at all, but only behaviors associated with it. For instance, the Java language and Lisp (programming language) API include the interface Serializable, which is a marker interface that requires that each class that implements it should behave in a serialized fashion. This does not require to have any public method, but rather requires that any class that implements it to have a representation that can be saved (serialized) at any time (this is typically true for any class containing simple data and no link to external resources, like an open connection to a file, a remote system, or an external device). Similarly the behavior of an object in a concurrent (multi-threaded) environment is not necessarily determined by specific methods, belonging to the interface implemented, but still belongs to the API for that Class of objects, and should be described in the documentation.

In this sense, in object-oriented languages, the API defines a set of object behaviors, possibly mediated by a set of class methods. In such languages, the API is still distributed as a library. For example, the Java language libraries include a set of APIs that are provided in the form of the JDK used by the developers to build new Java programs. The JDK includes the documentation of the API in **JavaDoc** notation. The quality of the documentation associated with an API is often a factor determining its success in terms of ease of use.

API Libraries and Frameworks

An API is usually related to a software library: the API describes and prescribes the expected behavior while the library is an actual implementation of this set of rules. A single API can have multiple implementations (or none, being abstract) in the form of different libraries that share the same programming interface. An

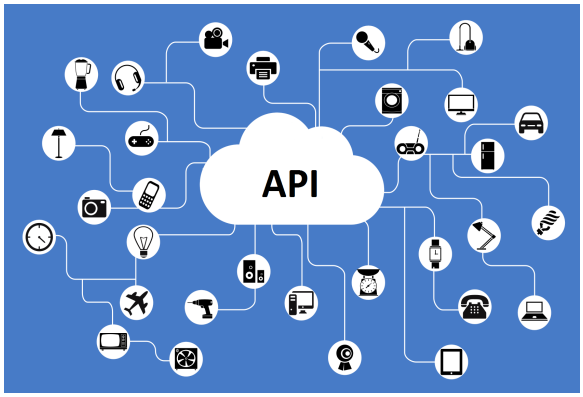
API can also be related to a software framework: a framework can be based on several libraries implementing several APIs, but unlike the normal use of an API, the access to the behavior built into the framework is mediated by extending its content with new classes plugged into the framework itself. Moreover the overall program flow of control can be out of the control of the caller, and in the hands of the framework via inversion of control or a similar mechanisms.

API and Protocols

An API can also be an implementation of a protocol. In general the difference between an API and a protocol is that the protocol defines a standard way to exchange requests and responses based on a common transport and agreeing on a data/message exchange format, while an API (not implementing a protocol) is usually implemented as a library to be used directly: hence there can be no transport involved (no information physically transferred from/to some remote machine), but rather only simple information exchange via function calls (local to the machine where the elaboration takes place) and data is exchanged in formats expressed in a specific language.

When an API implements a protocol it can be based on proxy methods for remote invocations that underneath rely on the communication protocol.

The role of the API can be exactly to hide the detail of the transport protocol. E.g.: RMI is an API that implements the JRMP protocol or the IIOP as RMI-IIOP. Protocols are usually shared between different technologies (system based on given computer programming languages in a given operating system) and usually allow the different technologies to exchange information, acting as an abstraction/mediation level between the two worlds. While APIs can be specific to a given technology: hence the APIs of a given language cannot be used in other languages, unless the function calls are wrapped with specific adaptation libraries.



Object API and Protocols

An object API can prescribe a specific object exchange format, an object exchange protocol can define a way to transfer the same kind of information in a message sent to a remote system. When a message is exchanged via a protocol between two different platforms using objects on both sides, the object in a programming language can be transformed (marshalled and unmarshalled) in an object in a remote and different language: so, e.g., a program written in Java invokes a service via SOAP or IIOP written in C# both programs use APIs for remote invocation (each locally to the machine where they are working) to (remotely) exchange information that they both convert from/to an object in local memory. Instead when a similar object is exchanged via an API local to a single machine the object is effectively exchanged (or a reference to it) in memory: e.g. via the memory allocated by a single process, or among multiple processes using shared memory or other sharing techologies like tuple spaces.

API Sharing and Reuse via Virtual Machine

Some languages like those running in a virtual machine (e.g. .NET CLI compliant languages in the Common Language Runtime and JVM compliant languages in the Java Virtual Machine) can share APIs. In this case the virtual machine enables the language interoperation thanks to the common denominator of the virtual machine that abstracts from the specific language using an intermediate bytecode and its language binding. Hence this approach maximizes the code reuse potential for all the existing libraries and related APIs.

Web APIs

When used in the context of web development, an API is typically defined as a set of Hypertext Transfer Protocol (HTTP) request messages, along with a definition of the structure of response messages, which is usually in an Extensible Markup Language (XML) or JavaScript Object Notation (JSON) format.

While "Web API" is virtually a synonym for web service, the recent trend (so-called Web 2.0) has been moving away from Simple Object Access Protocol (SOAP) based services towards more direct **Representational State Transfer (REST)** style communications. Web APIs allow the combination of multiple services into new applications known as mashups.

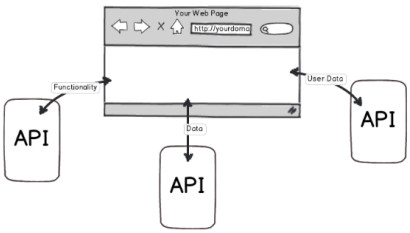
Web Use to Share Content

The practice of publishing APIs has allowed web communities to create an open architecture for sharing content and data between communities and applications. In this way, content that is created in one place can be dynamically posted and updated in multiple locations on the web.

1. Photos can be shared from sites like Flickr and Photobucket to social network sites like Facebook and MySpace.
2. Content can be embedded, e.g. embedding a presentation from SlideShare on a LinkedIn profile.
3. Content can be dynamically posted. Sharing live comments made on Twitter with a Facebook account, for example, is enabled by their APIs.
4. Video content can be embedded on sites which are served by another host.
5. User information can be shared from web communities to outside applications, delivering new functionality to the web community that shares its user data via an open API. One of the best examples of this is the Facebook Application platform. Another is the Open Social platform.

Implementations

The POSIX standard defines an API that allows a wide range of common computing functions to be written in a way such that they may operate on many different systems (Mac OS X, and various Berkeley Software Distributions (BSDs) implement this interface); however, making use of this requires re-compiling for each platform. A compatible API, on the other hand, allows compiled object code to function without any changes to the system implementing that API. This is beneficial to both software providers (where they may distribute existing software on new systems without producing and distributing upgrades) and users (where they may install older software on their new systems without purchasing upgrades), although this generally requires that various software libraries implement the necessary APIs as well.

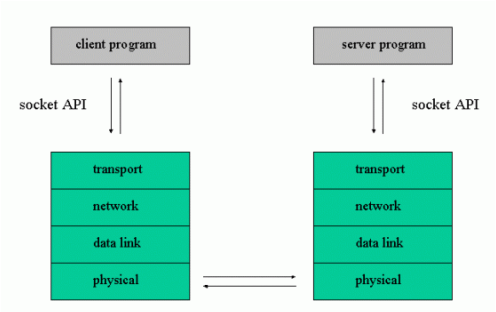


Apple Inc. has shown less concern, breaking compatibility or implementing an API in a slower "emulation mode"; this allows greater freedom in development, at the cost of making older software obsolete. Among Unix-like operating systems, there are many related but incompatible operating systems running on a common hardware platform (particularly Intel 80386-compatible systems). There have been several attempts to standardize the API such that software vendors may distribute one binary application for all these systems; however, to date, none of these have met with much success. The Linux Standard Base is attempting to do this for the Linux platform, while many of the BSD Unixes, such as FreeBSD, **NetBSD**, and OpenBSD, implement various levels of API compatibility for both backward compatibility (allowing programs written for older versions to run on newer distributions of the system) and cross-platform compatibility (allowing execution of foreign code without recompiling).

Release Policies

The two options for releasing API are:

1. Protecting information on APIs from the general public. For example, Sony used to make its official PlayStation 2 API available only to licensed PlayStation developers. This enabled Sony to control who wrote PlayStation 2 games. This gives companies quality control privileges and can provide them with potential licensing revenue streams.
2. Making APIs freely available. For example, Microsoft makes the Microsoft Windows API public, and Apple releases its APIs Carbon and Cocoa, so that software can be written for their platforms. A mix of the two behaviors can be used as well.



APIs and Copyrights

In 2010 Oracle sued Google for having distributed a new implementation of Java embedded in the Android operating system. The litigation concerns also the fact that Google did not acquire any permission to reproduce the Java API. Instead a similar permission was given to the OpenJDK project. However, Hon. William Alsup ruled on May 31, 2012, that APIs cannot be copyrighted.

API examples

- ASPI for SCSI device interfacing
- Carbon and Cocoa for the Macintosh
- DirectX for Microsoft Windows

- EHLLAPI
- Java APIs
- ODBC for Microsoft Windows
- OpenAL cross-platform sound API
- OpenCL cross-platform API for general-purpose computing for CPUs & GPUs
- OpenGL cross-platform graphics API
- OpenMP API that supports multi-platform shared memory multiprocessing programming in C, C++ and Fortran on many architectures, including Unix and Microsoft Windows platforms.
- Simple DirectMedia Layer (SDL)
- Talend integrates its data management with BPM from Bonita Open Solution

Language Bindings and Interface Generators

APIs that are intended to be used by more than one high-level programming language often provide, or are augmented with, facilities to automatically map the API to features (syntactic or semantic) that are more natural in those languages. This is known as language binding, and is itself an API. The aim is to encapsulate most of the required functionality of the API, leaving a "thin" layer appropriate to each language.

Some interface generator tools which bind languages to APIs at compile time.

- SWIG open-source interfaces bindings generator from many languages to many languages (Typically Compiled->Scripted)
- F2PY: Fortran to Python interface generator.

Create Web API project

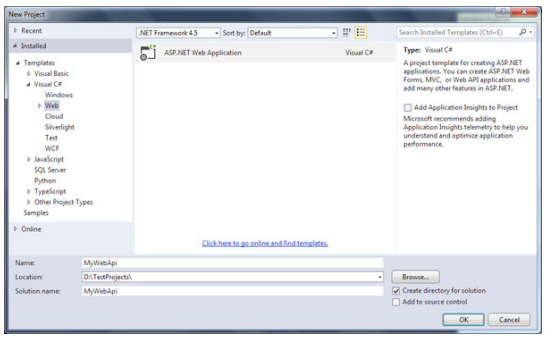
You can create a Web API project in two ways.

- Web API with MVC Project
- Stand-alone Web API Project

Web API with MVC Project

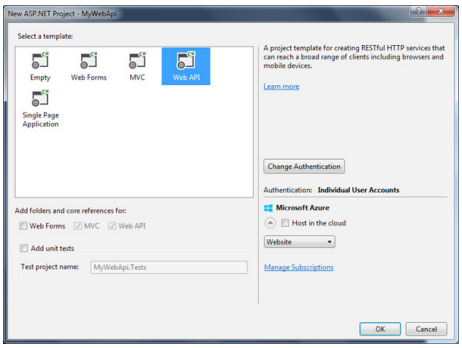
Visual Studio 2013 for Web includes Web API template which creates a new Web API project with ASP.NET MVC application and includes all the necessary references to get started.

For this, open Visual Studio 2013 for Web and click on File menu and click on New Project. This will open New Project popup as below.



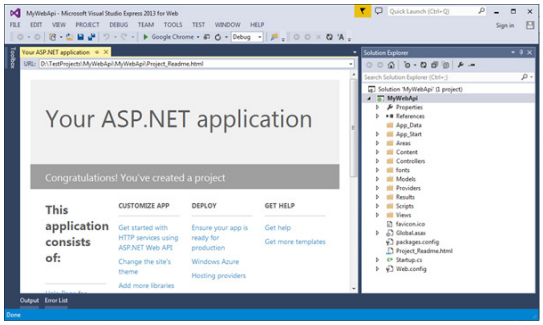
In the New Project popup, expand Visual C# and select Web node in the left pane. Select ASP.NET Web Application template in the middle pane and enter the name of a project and click OK.

This will open New ASP.NET Project popup as shown given.

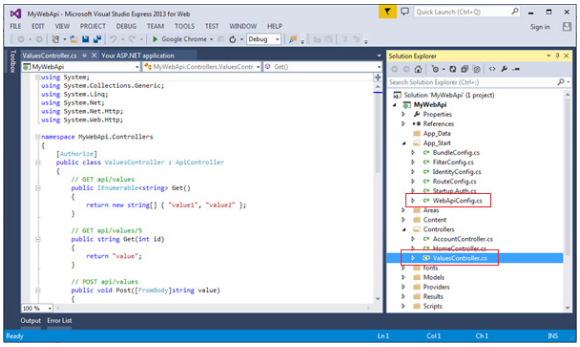


Select Web API in the above popup. Notice that this has selected and disabled MVC and Web API checkboxes. It means that it will add necessary folders and references for both MVC and Web API.

Click Ok to create a new Web API + MVC project as shown given.



This project is same as default MVC project with two specific files for Web API, WebApiConfig.cs in App_Start folder and ValuesController.cs in Controllers folder as shown

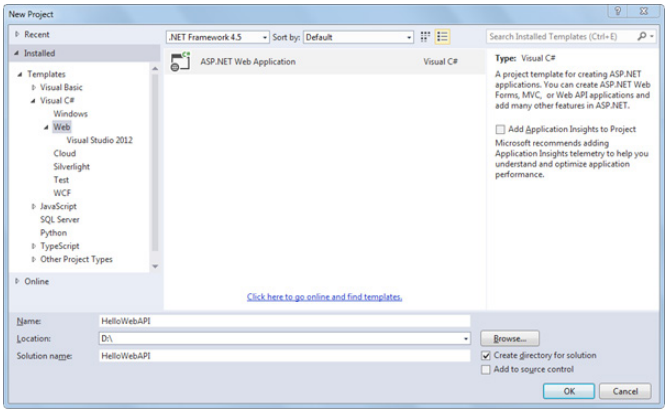


The WebApiConfig.cs is configuration file for Web API. You can configure routes and other things for web API, same like RouteConfig.cs is used to configure MVC routes. It also creates Web API controller ValuesController.cs by default.

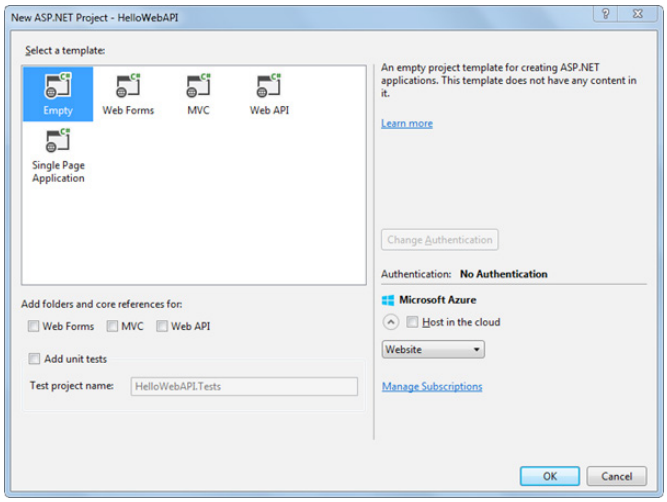
Stand-alone Web API Project

We will create a new stand-alone Web API project without MVC project.

For this, open Visual Studio 2013 for Web -> go to File menu and select New Project. This will open New Project popup as

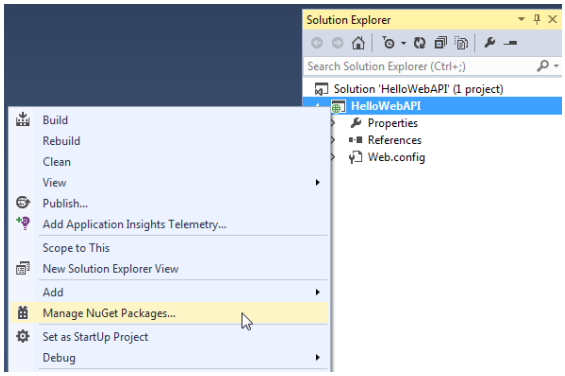


Select Web template in the left pane and ASP.NET Web Application in the middle pane. Enter a name of the project, location and Solution name as shown above. Clicking on OK will open a popup as shown

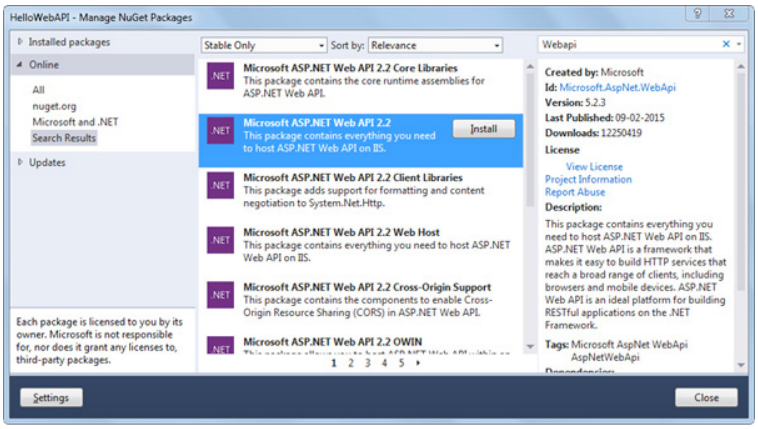


In the above popup, select Empty as a template and click ok. This will create an empty "HelloWebAPI" project.

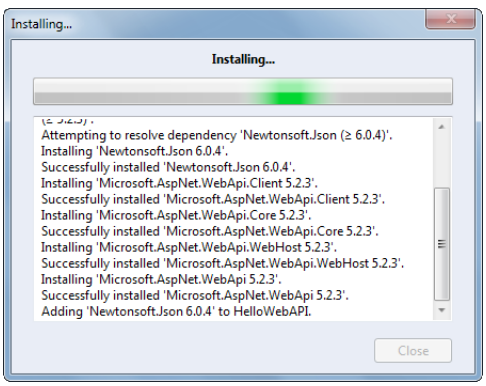
Now, we need to add latest Web API references using NuGet Package Manager. Right Click on the project and click Manage NuGet Packages. as shown



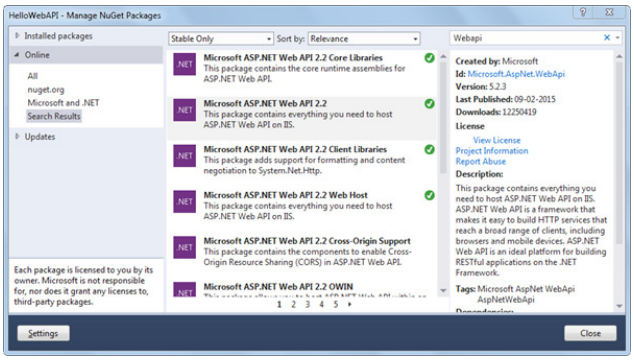
This will open Manage NuGet Packages popup. Select Online in the left pane and search for webapi (make sure that internet connection is on). This will display all the Web API related packages. Select Microsoft ASP.NET Web API2.2 package and click on Install as shown



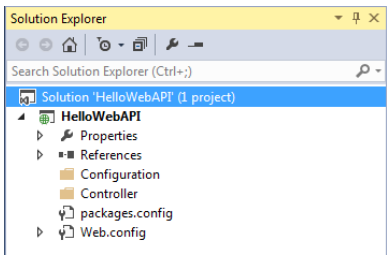
Accept the license agreement and continue.



The following Web API packages are displayed upon successful installation.



Now, create Controllers and Configuration folder in the HelloWebAPI project as shown below. We will add Web API controller in the Controllers folder and configuration class in the Configuration folder.



Now, add a new class in the configuration folder and name it "HelloWebAPIConfig" with following content.

Example: Web API Configuration

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

using System.Web.Http;

namespace HelloWebAPI.Configuration

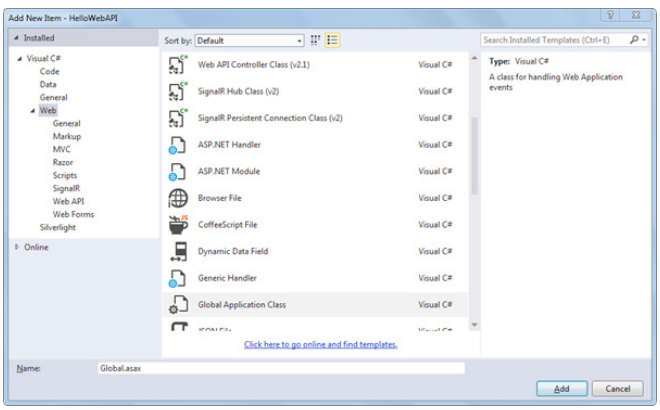
{

    public static class HelloWebAPIConfig
```

```
    {
        public static void Register(HttpConfiguration config)
        {
            // Web API routes
            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: "DefaultApi",
                routeTemplate: "api/{controller}/{id}",
               defaults: new { id = RouteParameter.Optional }
            );
        }
    }
}
```

Now, add Global.asax by right clicking on the project -> select Add -> click New Item. to open Add New Item popup as below. Select Global Application Class and click OK.



This will add Global.asax file into the project. We need to configure our Web API routes when application starts. So call HelloWebAPIConfig.Register() method in the Application_Start event in the Global.asax as shown

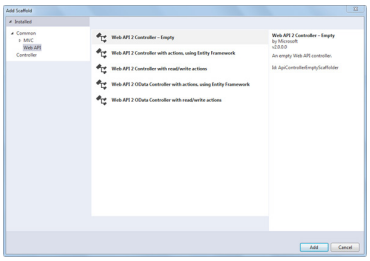
Global.asax

```
public class Global : System.Web.HttpApplication
{
    protected void Application_Start(object sender, EventArgs
e)
    {
        GlobalConfiguration.Configure(HelloWebAPIConfig.
Register);
    }
}
```

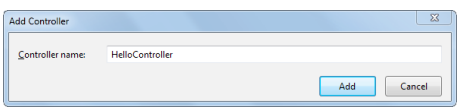
Web API is configured only using code based configuration using GlobalConfiguration class. The Configure() method requires a callback method where you have configured your Web API.

Thus, when application starts it will call Application_Start event which in turn calls HelloWebAPIConfig.Register() method to configure your Web API.

Now, let's add Web API controller by right clicking on the Controllers folder -> select Controller.. this will open popup as below.



Select Web API in the left pane and Web API 2 Controller - Empty in the middle pane and click Add. This will open another popup to enter a name of your controller as below. Enter controller name and click Add.



This will create following empty HelloController class in Controllers folder.

Example: Web API Controller

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;

namespace HelloWebAPI.Controller
{
    public class HelloController : ApiController
    {

    }
}
```

Web API Controller

Web API Controller is similar to ASP.NET MVC controller. It handles incoming HTTP requests and send response back to the caller.

Web API controller is a class which can be created under the Controllers folder or any other folder under your project's root folder. The name of a controller class must end with "Controller" and it must be derived from System.Web.Http.ApiController class. All the public methods of the controller are called action methods.

Example: Simple Web API Controller

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;

namespace MyWebAPI.Controllers
{
    public class ValuesController : ApiController
    {
        // GET: api/student
        public IEnumerable<string> Get()
        {
            return new string[] { “value1”, “value2” };
        }

        // GET: api/student/5
        public string Get(int id)
        {
            return “value”;
        }

        // POST: api/student
        public void Post([FromBody]string value)
        {
```

```
        }

        // PUT: api/student/5
        public void Put(int id, [FromBody]string value)
        {
        }

        // DELETE: api/student/5
        public void Delete(int id)
        {
        }
    }
}
```

Based on the incoming request URL and HTTP verb (GET/POST/PUT/PATCH/DELETE), Web API decides which Web API controller and action method to execute e.g. Get() method will handle HTTP GET request, Post() method will handle HTTP POST request, Put() mehtod will handle HTTP PUT request and Delete() method will handle HTTP DELETE request for the above Web API.

The following figure illustrates the significance of Web API controller and action methods.



If you want to write methods that do not start with an HTTP verb then you can apply the appropriate http verb attribute on the method such as HttpGet, HttpPost, HttpPut etc. same as MVC controller.

Example: Simple Web API Controller

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;

namespace MyWebAPI.Controllers
{
    public class ValuesController : ApiController
    {
        [HttpGet]
        public IEnumerable<string> Values()
        {
            return new string[] { “value1”, “value2” };
        }

        [HttpGet]
        public string Value(int id)
        {
            return “value”;
        }
```

```
        [HttpPost]
        public void SaveNewValue([FromBody]string value)
        {
        }

        [HttpPut]
    public void UpdateValue(int id, [FromBody]string value)
        {
        }

        [HttpDelete]
        public void RemoveValue(int id)
        {
        }
    }
}
```

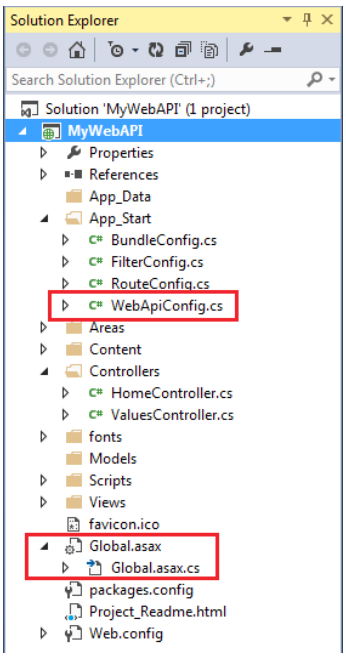
Web API Controller Characteristics

- It must be derived from System.Web.Http.ApiController class.
- It can be created under any folder in the project's root folder. However, it is recommended to create controller classes in the Controllers folder as per the convention.
- Action method name can be the same as HTTP verb name or it can start with HTTP verb with any suffix (case in-sensitive) or you can apply Http verb attributes to method.
- Return type of an action method can be any primitive or complex type.

Configure Web API

Web API supports code based configuration. It cannot be configured in web.config file. We can configure Web API to customize the behavior of Web API hosting infrastructure and components such as routes, formatters, filters, DependencyResolver, MessageHandlers, ParamterBindingRules, properties, services etc.

We created a simple Web API project in the Create Web API Project section. Web API project includes default WebApiConfig class in the App_Start folder and also includes Global.asax as shown below.



Global.asax

```
public class WebAPIApplication : System.Web.HttpApplication
{
    protected void Application_Start()
    {
     GlobalConfiguration.Configure(WebApiConfig.Register);
```

```
        //other configuration
    }
}
```

WebApiConfig

```
public static class WebApiConfig
    {
        public static void Register(HttpConfiguration config)
        {

            config.MapHttpAttributeRoutes();

            config.Routes.MapHttpRoute(
                name: “DefaultApi”,
                routeTemplate: “api/{controller}/{id}”,
                defaults: new { id = RouteParameter.Optional }
            );

            // configure additional webapi settings here..
        }
    }
```

Web API configuration process starts when the application starts. It calls GlobalConfiguration.Configure(WebApiConfig. Register) in the Application_Start method. The Configure() method requires the callback method where Web API has been configured in code. By default this is the static WebApiConfig. Register() method.

API USES

Web APIs are the defined interfaces through which interactions happen between an enterprise and applications that use its assets. An API is a set of protocols, subroutines and tools to develop software programs. Like a graphical interface, the objective of an API is to make the developer's work easier. To achieve this, an API provides remote calls, object classes, data structures or variables, which are all the blocks required for a developer to use certain technologies in his application, reducing the cognitive load to which he is exposed. The maintenance and development of applications that use APIs, becomes more expensive and time consuming, without proper documentation. This, should contain all the relevant information about the services available in the API, as well as the way they work, so the developer knows what the expected response when interacting with it is. Despite the three components introduced being different from each other, there is a strong connection between an API, library and framework. An API describes a behavior while a library contains all the implementation of the rules used in this behavior. Regarding the API-framework relation, an API is the public side of a framework, which means, from all the methods implemented in a framework, only the ones relevant to the user will be presented in an API.

Library

A library is composed by a set of rules used by an application, this means the library contains all the implementation for the expected behavior. Example given, when a programmer wants to develop an application, he can use a library to make the system calls instead of implementing them. The code available in the library is independent from the application itself and is structured in a way that it can be used and reused by multiple applications, without these being connected between each other. These libraries, allow the code to be shared by multiple developers, which makes it easy when developing new software projects.

Framework

Last but not least, a framework is composed by a group of libraries that allows to execute a bigger operation. These frameworks can

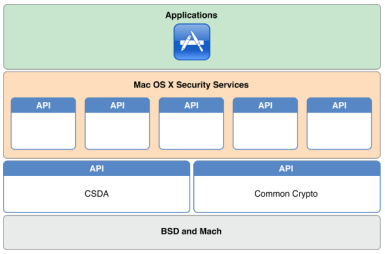
be seen as development platforms, which should be adapted to the developer's requirements. In the end, a framework can be seen as the skeleton of the application, for example, considering a team that is developing a website for a company, by using a framework, they can focus their efforts on specific code for the website itself instead of all the handling and management of the requests.

Operating Systems

An API can specify the interface between an application and the operating system. POSIX, for example, specifies a set of common APIs that aim to enable an application written for a POSIX conformant operating system to be compiled for another POSIX conformant operating system. Linux and Berkeley Software Distribution are examples of operating systems that implement the POSIX APIs. Microsoft has shown a strong commitment to a backward-compatible API, particularly within their Windows API (Win32) library, so older applications may run on newer versions of Windows using an executable-specific setting called "Compatibility Mode". An API differs from an application binary interface (ABI) in that an API is source code based while an ABI is binary based. For instance, POSIX provides APIs, while the Linux Standard Base provides an ABI.

- Software developers have demonstrated that many different programs can take advantage of a number of similar modules.

For example, word processors, spreadsheets, presentation software, and database software, although they accomplish quite different tasks, all need to be able to permit the user to save work in progress into a data file, with the user being able to select the disk drive, folder name, file name, and in some cases file format.



Every version of the Macintosh OS and every version of Windows has included, as part of the operating system, a module that provides that functionality.

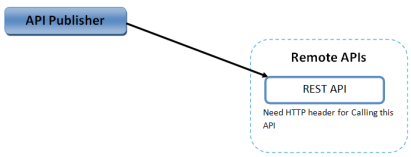
- Software developers writing application programs do not have to write their own module that performs such functions. Instead, they write their software to use the operating system›s module.
- Some operating systems provide modules that support graphical user interfaces, other operating systems provide modules that support command line user interfaces, and some provide both.

Remote APIs

Remote APIs allow clients connecting from a separate tier to communicate with the Alfresco content application server. Remote APIs are based on web services and RESTful and CMIS protocols, and are language agnostic, allowing you to develop against these APIs using a range of languages including Java, PHP, Ruby, and .NET.

There are several Remote APIs available, including:

- **Web services** - object-oriented API using SOAP and supporting a range of content services including authentication, query, node creation and update, access control, and actions
- **RESTful** - HTTP-based resource oriented interfaces used by the Surf framework and Alfresco Share
- **CMIS** - a standard that defines web services and REST-based bindings for working with CMIS compliant repositories



A remote API allowing to control a simulation (or the simulator itself) from an external application or a remote hardware (e.g. real robot, remote computer, etc.). The remote API

is composed by approximately one hundred specific functions and one generic function that can be called from a C/C++ application, a Python script, a Java application, a Matlab/Octave program, an Urbi script, or a Lua script. The remote API functions are interacting with via socket communication in a way that reduces lag and network load to a great extent. All this happens in a hidden fashion to the user. The remote API can let one or several external applications interact with V-REP in a synchronous* or asynchronous* way (asynchronous by default), and even remote control of the simulator is supported (e.g. remotely loading a scene, starting, pausing or stopping a simulation for instance).

The remote API functionality comes in 2 separate entities that are interacting via socket communication:

- The client side (i.e. your application): the remote API on the client side is available for many different programming languages. Currently following languages are supported: C/C++, Python, Java, Matlab, Octave, Urbi and Lua. Bindings for other languages can easily be created by yourself. Instructions on how to enable the remote API on the client side are given here.
- The server side: the remote API on the server side is implemented via a plugin that should be loaded by default: v_repExtRemoteApi.dll, libv_repExtRemoteApi.dylib or libv_repExtRemoteApi.so. The plugin project files are located in installation directory, in the "programming" folder. Instructions on how to enabled the remote API on the server side are given here.

Use in Web APIs

An API approach is an architectural approach that revolves around providing programmable interfaces to a set of services to different applications serving different types of consumers. When used in the context of web development, an API is typically defined as a set of Hypertext Transfer Protocol (HTTP) request messages, along with a definition of the structure of response messages, which is usually in an Extensible Markup Language (XML) or **JavaScript Object Notation (JSON)** format. An example might be a shipping company API that can be added to an eCommerce-focused web site, to facilitate ordering shipping

services and automatically include current shipping rates, without the site developer having to enter the shipper's rate table into a web database.

While "web API" historically has been virtually synonymous for web service, the recent trend (so-called Web 2.0) has been moving away from Simple Object Access Protocol (SOAP) based web services and service-oriented architecture (SOA) towards more direct representational state transfer (REST) style web resources and resource-oriented architecture (ROA).

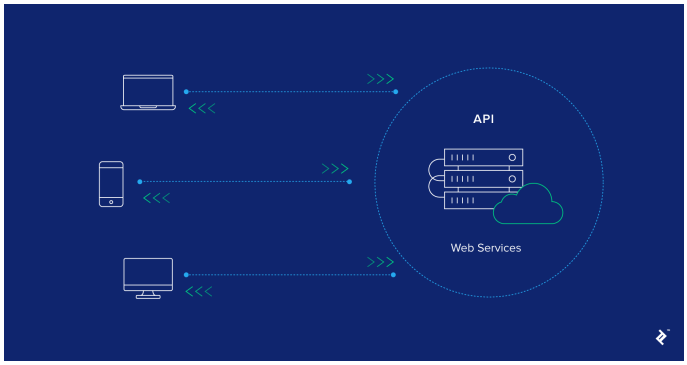
Part of this trend is related to the Semantic Web movement toward Resource Description Framework (RDF), a concept to promote web-based ontology engineering technologies. Web APIs allow the combination of multiple APIs into new applications known as mashups.

In the social media space, web APIs have allowed web communities to facilitate sharing content and data between communities and applications. In this way, content that is created in one place can be dynamically posted and updated in multiple locations on the web.

Web API as the name suggests, is an API over the web which can be accessed using HTTP protocol. It is a concept and not a technology. We can build Web API using different technologies such as Java, .NET etc. For example, Twitter's REST APIs provide programmatic access to read and write data using which we can integrate twitter's capabilities into our own application.

Web API in ASP.NET

The ASP.NET Web API is an extensible framework for building HTTP based services that can be accessed in different applications on different platforms such as web, windows, mobile etc. It works more or less the same way as ASP.NET MVC web application except that it sends data as a response instead of html view. It is like a webservice or WCF service but the exception is that it only supports HTTP protocol.



ASP.NET Web API Characteristics:

- ASP.NET Web API is an ideal platform for building RESTful services.
- ASP.NET Web API is built on top of ASP.NET and supports ASP.NET request/response pipeline
- ASP.NET Web API maps HTTP verbs to method names.
- ASP.NET Web API supports different formats of response data. Built-in support for JSON, XML, BSON format.
- ASP.NET Web API can be hosted in IIS, Self-hosted or other web server that supports .NET 4.0+.
- ASP.NET Web API framework includes new HttpClient to communicate with Web API server. HttpClient can be used in ASP.MVC server side, Windows Form application, Console application or other apps.

API TESTING

API testing is entirely different from GUI testing and mainly concentrates on the business logic layer of the software architecture. This testing won't concentrate on the look and feel of an application. The first part of API testing involves setting up a testing environment, with the required set of parameters around the API.

API testing is a type of software testing that involves testing application programming interfaces (APIs) directly and as part of integration testing to determine if they meet expectations for functionality, reliability, performance, and security. Since APIs lack a GUI, API testing is performed at the message layer. API testing is now considered critical for automating testing because

APIs now serve as the primary interface to application logic and because GUI tests are difficult to maintain with the short release cycles and frequent changes commonly used with Agile software development and DevOps).

The API Testing is performed for the system, which has a collection of API that ought to be tested. During Testing, a test of following things is looked at.

- Exploring boundary conditions and ensuring that the test harness varies parameters of the API calls in ways that verify functionality and expose failures.
- Generating more value added parameter combinations to verify the calls with two or more parameters.
- Verifying the behavior of the API which is considering the external environment conditions such as files, peripheral devices, and so forth.
- Verifying the Sequence of API calls and check if the API's produce useful results from successive calls.

Instead of using standard user inputs (keyboard) and outputs, in API Testing, you use software to send calls to the API, get output, and note down the system's response.

API Testing requires an application to interact with API. In order to test an API, you will need to

- Use Testing Tool to drive the API
- Write your own code to test the API

Set-up of API Test environment

- API testing is different than other testing types as GUI is not available, and yet you are required to setup initial environment that invokes API with required set of parameters and then finally examines the test result.
- Hence, Setting up a testing environment for API testing seems a little complex.
- Database and server should be configured as per the application requirements.
- Once the installation is done, API Function should be called to check whether that API is working.

Common Tests performed on API's

- Return Value based on input condition - The return value from the API's are checked based on the input condition.
- Verify if the API's does not return anything.
- Verify if the API triggers some other event or calls another API. The Events output should be tracked and verified.
- Verify if the API is updating any data structure.

Challenges with API Testing

An API is an interface intended to be interpreted by a computer. Since the interface boundary has to be technology agnostic, the inputs and outputs are normally text-formatted messages. Text messages may be easy for a human to read in order to interpret the inputs and outputs, but in-depth knowledge of the internals of the application is required to sufficiently test it.

This is the difference between black-box testing and **white-box testing**. In black-box testing, the tester is only concerned with the functionality of the overall application, exercising only the inputs and checking the outputs.

In white-box testing, the tester is much more concerned with internal operations of the application, exercising the individual paths the data can travel through the application.

API testing normally includes only the white-box testing approach.

Another difficulty is that testers are used to working from a set of provided business requirements. However, business requirements normally describe use cases at a fairly high level, and do not go down to the level of individual application functions.

It is not unusual that business analysts are not even aware of low level detail and the amount of granularity that everything has to be broken down into, and are not aware that software is able to be tested at this level.

Typically, project managers are not concerned with assigning time specifically to developing a rich and detailed API, let alone testing it. Paradoxically, if everything is done correctly, the API is the only place where business rules are coded and enforced. The end-user GUI only displays results and any error messages provided by the API.

Sadly, because of the above problems, developers are often left with testing their own code, if there is any testing at this level at all. And due to time pressures, unit tests often only glance at the positive tests – the happy path – and more in depth testing is left until later in the project.

Of course, looking at only the positive paths through the application is not really testing, but only checking that the software fulfils business requirements under the most optimal conditions.

The GUI usually exposes only part of the overall functionality and by design testers are not able to go really deep into the application code, especially when it comes to testing all the negative test cases. This leaves the internals of the application possibly exposed to all sorts of breakage and attacks by end-users, accidental or malicious.

Lastly, there are situations where the API is the final product – a public API. A simple Internet search will reveal there are thousands of public APIs for anything from GPS and mapping solutions to locating a radio station that is playing your favorite song right now. However, these APIs expose functionality that the developer hopes or thinks other developers/ users will want. In this case, the tester's role needs also to include a user advocate role, to make sure the offering is complete – everything that any user may need is all there. This is in addition to the multitude of hats, the tester already wears: functional testing, security testing, load testing, etc.

Advantages of API Testing

Putting more effort into API testing leads to a much healthier final product. Ensuring that all data access (read and write) goes only through the API significantly simplifies security and compliance testing and thereby certification, since there is only one interface.

Ensuring that all the required business rules are being enforced at the API tier allows time for much more complete user-experience tests once the UI is released, and not having to concentrate on testing every single business rule and path through the application near the end of the project. Ensuring that the API offers complete functionality allows for easy future expansion of the application as new business needs arise.

Start API testing

Once you've set up your API testing environment, make an API call right away to make sure nothing is broken before you go forward to start your more thorough testing.

You can start combining your application data with your API tests to ensure that the API performs as expected against possible known input configurations. REST Testing example for beginners to help with the visualization.

Next, you need to organize yourself around the API test. Start by asking yourself these questions:

- Who is your target audience? Who is your API consumer?
- What environment/s should the API typically be used?
- What aspects are you testing?
- What problems are we testing for?
- What are your priorities to test?
- What is supposed to happen in normal circumstances?
- What could potentially happen in abnormal circumstances?
- What is defined as a Pass or a Fail? What data is the desired output? What is the chain of events?
- What other APIs could this API interact with?
- Who on your team is in charge of testing what?

After you have created these testing boundaries and requirements, you need to decide what you want to test your API for. These are some of the common kinds of API testing:

- Functionality testing — the API technically works.
- Usability testing — the API is easy to work with.

- Reliability testing — the API can be consistently connected to and lead to consistent results.
- Load testing — the API can handle a large amount of calls.
- Creativity testing — the API can handle being used in different ways.
- Security testing — the API has defined security requirements including authentication, permissions and access controls. See some API security tips for protectiving vital data.
- Proficiency testing — the API increases what developers are able to do.
- API documentation testing — also called discovery testing, the API documentation easily guides the user.

Tests will surely vary but here are common API test examples:

- Checking API return values based on the input condition.
- Verifying if the API doesn't return anything at all or the wrong results.
- Verifying if the API triggers some other event or calls another API.
- Verifying if the API is updating any data structures.

KEY TERMS

Programming language
Representational State Transfer (REST)
Website
White-box testing
Graphical user interface
Object-oriented languages
Javadoc
NetBSD
JavaScript Object Notation (JSON)
Interfaces
Function prototype

DISCUSSION QUESTIONS

1. Explain the application programing interface.
2. Describe the role of API in object-oriented languages.
3. Discuss on API libraries and frameworks.
4. Give an overview on API and protocols
5. What do you understand by the remote APIs?
6. Determine the process of API testing
7. Identify the challenges in API testing.
8. Focus on web API controller.
9. How to configure web API?
10. Learn about remote APIs

REFERENCES

1. An Online Platform for Web APIs and Service Mashups". IEEE Internet Computing. 12 (5). Sep–Oct 2008.
2. API Development with Lumen (Laravel) Microframework". Retrieved 2016-09-19.
3. Benslimane, D.; Dustdar, S.; Sheth, A. (2008). "Services Mashups: The New Generation of Web Applications". IEEE Internet Computing. 10 (5): 13–15. doi:10.1109/MIC.2008.110.
4. Bloch, Joshua (2008). "Effective Java (2nd edition)" (http://java.sun.com/docs/books/effective/). Addison-Wesley. pp. 259–312. ISBN 978-0-321-35668-0. .
5. Clarifai API: Large Scale Visual Recognition". developer.clarifai.com. Retrieved 2015-11-04.
6. Difference between Web API and Web Service?". programmers. stackexchange.com. Retrieved 2015-11-03.
7. Niccolai, James (2008-04-23), "So What Is an Enterprise Mashup, Anyway?", PC World, "Dynamic Community content via APIs". October 26, 2009.
8. Oracle and the End of Programming As We Know It" (http://www.drdobbs.com/jvm/232901227). DrDobbs. 2012-05-01. Retrieved 2012-05-09. 10 "F2PY.org" (http://www.f2py.org/). F2PY.org. . Retrieved 2011-12-18.
9. REST vs SOAP, the difference between soap and rest: spf13.com". spf13 is Steve Francia. 2010-01-15. Retrieved 2015-11-02.
10. Stoughton, Nick (April 2005). "Update on Standards". USENIX. . Retrieved 2009-06-04.

INDEX

A

Active Server Pages (ASP) 280
addEventListener() method 322, 323, 328, 331, 334, 335, 336, 339, 345, 348
Application program interface (API) 355
Application programming interfaces (APIs) 152

B

Berkeley Software Distributions (BSDs) 363
blur events 317, 318, 320
box model 218, 221, 225
browser window 312, 316, 317
Bubbling 328

C

Capturing 328
Cascading style sheets (CSS) 190
Class interface 358
click event 318, 320, 328, 331, 334, 335, 336, 337, 338, 339
Common Gateway Interface (CGI) 241, 282
Communication protocol 360
Component Object Model (COM) 280
Computer programming languages 360
Computer system 234
Content development 8, 9
Content management system (CMS) 22
CSS events 322

D

dblclick event 311, 335
Development environment 244
Document Type Definitions (DTDs) 147
Domain Name Servers (DNS) 12

E

element backgrounds 212
embedded style sheet 195, 198
Event 312, 313, 314, 315, 316, 328, 330, 351
event flow 318
Event Handlers 312
Event Handling 312
event listener 316, 322
event load 312
Exchange information 360, 361
Extensible HyperText Markup Language (XHTML) 41
Extensible Markup Language (XML) 41, 129, 133, 152, 154, 362, 383
external style sheet 203, 204, 206

F

Facebook Application Platform 355
File Transfer Protocol (FTP) 235
Function call 360

G

Google Maps 356
Google Web Server (GWS) 235
Graphical user interface (GUI) 355
Grid Layout 232

H

horizontal length 214
Horizontal lines 53
HTML5 events 321
HTML code 39, 42, 45, 46
HTML document 42, 44, 45, 46, 47, 48, 54, 56, 120
HTML element 40, 56, 57, 63
HTML Formatting 64
HTML (Hypertext Markup Language) 38, 39
HTML page 316
HTML tags 42, 43, 44, 45, 57, 63
Hypertext Markup Language (HTML) 133
Hypertext Preprocessor (PHP) 14
Hypertext Transfer Protocol (HTTP) 362, 383

I

inline styles 194, 198, 203
Integrated Development Environment (IDE) 245
Interface Definition Language (IDL) 165
internationalization attributes 61
Internet Information Server (IIS) 235, 240

J

Java language libraries 359
Java programming language 234
JavaScript error 317
JavaScript Object Notation (JSON) 362, 383
JavaServer Pages (JSP) 280, 281
Java Virtual Machine™ (JVM). 281
JSP Directives 297

K

keypress events 319

L

Language dependent API 355
Language independent APIs 355
Linux Standard Base 363, 381
Local Area Network (LAN) 3
Local restaurant 356

M

Machine language code 4
markup code 38
Markup language 4
Metropolitan Area Network (MAN) 3
Microsystems Java Software Development Kit (JDK) 252
mouse events 318
mouse moves 318
mutation event 320
mutation observers 320, 321

O

Object-oriented languages 358, 359, 391
onclick event 311, 330, 331, 332, 333, 350
onload attribute 340, 343, 344
onresize event 311, 344, 345, 347

P

Programming interface 359
Prototype 356

R

Remote system 359, 361
Representational State Transfer (REST) 362
Resource Description Framework (RDF) 384
Resource-oriented architecture (ROA) 384
Responsive web design (RWD) 17

S

Service Modeling Language (SML) 149
Service-oriented architecture (SOA) 384
Servlet container 241, 242, 255
Servlet Life Cycle 283
Servlet technology 234, 236

Simple DirectMedia Layer (SDL) 365
Simple Object Access Protocol (SOAP) 362, 384
Software-as-a-Service (SaaS) 354
Software Development Kit (SDK) 244
Software library 359
Software-to-software interface 354
Standard Generalized Markup Language (SGML) 133, 139
Style sheets 193, 203

V

virtual machine (VM) 280
void elements 56

W

Web browser 38, 39, 42
Web pages 39, 42
Web site 203
Web technology 3, 4, 6, 27, 28
Wide Area Network (WAN) 3
World Wide Web Consortium (W3C) 26, 38, 40, 44, 57, 133, 147, 157

X

XML Schema Definition (XSD) 147, 148

3G E-LEARNING

Level: Beginner to Advanced
Subject: Computer and Information Science

Web Engineering

3rd Edition

Web engineering, an emerging new discipline, advocates a process and systematic approach to developing high-quality web-based systems. It promotes the establishment and application of sound scientific, technical and managerial principles, as well as disciplined and systematic approaches to the development, deployment and maintenance of web-based systems. Earlier web developers considered web engineering as a specialized field of software engineering, but it encompasses more than it, since every web application involves content, hypertext structure and presentation to a potentially very diverse audience, web developers need a wide range of domain knowledge provide. Web engineering is a holistic approach and deals with all aspects of web-based system development, starting with conception and development through to implementation, performance evaluation and ongoing maintenance. Building and deploying a web-based system involves several iterative steps. Most web-based systems are constantly evolving to keep information current and to meet user needs. Web engineering represents a proactive approach to building web applications. Web engineering methodologies have been successfully applied to a number of web applications. In this edition of the book, the text has been revised and restructured, and a new chapter has been added. This edition contains eight chapters. With explicit descriptions and scripting languages such as VBScript, JavaScript, and more, this book will be a valuable tool for anyone involved in web engineering.

ISBN 978-1-98468-225-3

BIBLIOTEX
Digital Library

www.3ge-learning.com
email: info@3ge-learning.com