BENDRASIS PROGRAMAVIMO DOKUMENTAS

EUROPOS SĄJUNGA

NACIONALINĖ ŽEMĖS TARNYBA
PRIE ŽEMĖS ŪKIO MINISTERIJOS

THE BASIS OF AN EU CO-FINANCED PROJECT "QUALIFICATION IMPROVEMENT OF THE LITHUANIAN GEOGRAPHICAL INFORMATION MANAGERS" (NO BPD2004-ESF-2.2.0.-02-05/0143)

Civil servants learning programme

DISTANCE LEARNING OF GEOGRAPHIC INFORMATION INFRASTRUCTURE

Training Material

WEB PROGRAMMING FOR GEOGRAPHIC INFORMATION INFRASTRUCTURE

GII-09

Vilnius, 2008

MALASPINA
UNIVERSITY-COLLEGE

HNIT - BALTIC

Training material „WEB programming for geographic information infrastructure“ (GII-09)

Authors

1 Module - Frank Fucile

2 Module - Michael O'Neal Campbell

3, 4, 5 Modules - Michael Govorov and dr. Gennady Gienko

Reviewed and edited by

prof. Romualdas Baušys (Vilnius Gediminas Technical University)

Reviewed by

prof. habil. dr. Genadijus Kulvietis (Vilnius Gediminas Technical University)

From English translated and edited by

Astraneta UAB

Study material for civil servants distance learning programme “GII Distance Learning”, which has been approved by a law “RE Approval of Study Programmes for Civil Servants” (No 27V-72; Apr 17, 2007) by the State Service Department under the Ministry of the Interior has been prepared on the basis of an EU co-financed project “Qualification Improvement of the Lithuanian Geographical Information Managers” (No BPD2004-ESF-2.2.0.-02-05/0143)

Beneficiary of the project National Land Service under the Ministry of Agriculture.

Contractor of the project HNIT-BALTIC UAB, Vilnius Gediminas Technical University

Table of Contents

Introduction

A Web application provides content from a server to a client as a visual interface for a user, often within a Web browser. Much of the business logic, data, and other processes are managed on the server-side using a rich programming environment, such as C#, Visual Basic.NET, Java, etc., but also scripting languages such as PHP, Python, JavaScript, VBScript. Since the user interacts with a Web application via a Web browser, the client-side application is built using Web standards to create a user interface, such as HTML, CSS, and scripting languages (e.g., JavaScript). For sever-client communication and data encoding, other Web standards are used, such as XML or SOAP. In addition, Web applications can be implemented in different environments, such as N-tier architecture, SOA, and using different technologies such as .NET, J2EE, or AJAX.

This course provides an overview of the web technologies and programming environments for building web mapping applications and discusses some of these technologies in detail. The first module of the course discusses the origins of the Internet and how it works in regard to the creation and coding of webpages. Students will be learning the basic building blocks of the Internet in terms of HTML codes, more specifically the use of tables, links, frames, images, and HTML forms. Students will also learn the more advanced code with the use of cascading style sheets (CSS) to control the look, feel, and layout of webpages. This knowledge can be used to design web-mapping applications that require HTML and CSS skills.

The second module discussed scripting languages and associated technologies, such as DHTML, HTML Document Object Models (DOMs), and JavaScript. Many web-mapping applications use JavaScript to perform many client-side functions, such as map-click interaction and retrieving data from the server via callbacks. In addition, XML and XSL languages are analyzed here. These languages are very important for web request-response communication, spatial data and application schemas encoding, and data rendering. This module provides an introduction to these web application technologies, building on topics already considered in Module 1. This module also describes how recent developments in web application technology have facilitated an increased efficiency of internet technology.

The third module introduces web mapping and related technology and specifications. The first topic to be discussed is multi-tiered architecture that serves as the environment for web-mapping services. AJAX web technology, often used for web mapping applications on the front client tier, is also overviewed. Next, the three-tiered architecture is analyzed for web mapping applications and systems within Web Mapping Server (WMS) and thick Web Feature Server (WFS), Web mapping clients are also introduced. Further, the OGC standard implementation specifications that defined functionalities of Web Mapping Server, Web Feature Server, and Web Coverage Server are discussed in detail. These specifications have to be used by developers to build interoperable distributed web mapping systems. In this module, student knowledge, built in previous modules regarding HTTP, XML, XSL, DTD, XSD, etc, is applied to learning how to build web-mapping architectures.

The fourth module devotes the languages for geospatial application that can be implemented in XML. These include GML (Geographic Markup Language), ArcXML (ESRI Arc Extensible Markup Language), KLM (Google Keyhole Markup Language), SVG (Scalable Vector Graphics), GeoRSS (Geographic Really Simple Syndication), and many others. Moreover, XML is sometimes used as the specification language for such application languages. This module discusses a few XML application grammas that are used for spatial data inter-exchange encoding; web mapping service communication and map definition; consuming and mashuping web services within Service Oriented Architecture; and, vector graphic representation on the client. These XML application schemas could be used for data modeling and inter-exchange within different public and commercial applications such as cadastral and land use, traffic and transportation, telecommunication, environment, and many others. In addition, topics related to Service Oriented Architecture, web map services, and related technologies are discussed.

In previous modules, web mapping architecture, its components, and web languages for user interface, communication, data, and schema descriptions were discussed. In the last module, previously discussed topics are presented by using implemented industrial examples - ESRI ArcIMS and ArcGIS Server. ESRI was a pioneer in the development of web mapping software. ESRI products were chosen for analytical purposes because they are supplemented by documentation that is easy to use. Using ArcIMS, city and local governments, businesses, and other organizations worldwide publish, discover, and share geospatial information.

1. Internet Technology, HTML, and CSS

In this module you will be discovering the origins of the Internet and how it works in regard to the creation and coding of webpages. You will be learning the basic building blocks of the Internet in terms of HTML code, more specifically the use of tables, links, frames, images, and HTML forms. You will also be learning more advanced code with the use of cascading style sheets to control the look, feel, and layout of your webpages.

The following topics will be covered in Module 1:

Module Outline

Topic 1: How the Internet works?
Topic 2: HTML Tags
Topic 3: Links
Topic 4: Images
Topic 5: Tables
Topic 6: Frames
Topic 7: Divisions
Topic 8: CSS (Cascading Style Sheets)
Topic 9: Forms

1.1. How the Internet works?

1.1.1 What is the Internet?

The Internet is a technology for interconnecting networks of computers worldwide. The internet even extends to the Antarctica and the International Space Station. The Internet was developed out of ARPAnet in the early 1970's by the US government and it is based on the TCP/IP (Transmission Control Protocol/Internet Protocol). The TCP/IP protocol allows you to send and receive any type of digital information

1.1.2 Layers of the Internet

Seven Levels of data transport in the Open Systems Interconnection (OSI) model (source is http://en.wikipedia.org/wiki/OSI_model):

Physical (10Base-T, ISDN, RS-232)
The Physical layer defines all the electrical and physical specifications for devices.
Data Link (Ethernet, Wi-Fi, PPP, Token Ring…)\
The Data Link layer provides the functional and procedural means to transfer data between network entities and to detect and possibly correct errors that may occur in the Physical layer.
Network (IPV4, IPV6, ICMP, IGMP, ARP…)
The Network layer provides the functional and procedural means of transferring variable length data sequences from a source to a destination via one or more networks while maintaining the quality of service requested by the Transport layer.
Transport (TCP, UDP…)
The Transport layer provides transparent transfer of data between end users, providing reliable data transfer services to the upper layers.
Session (RPC)
The Session layer controls the dialogues/connections (sessions) between computers.
Presentation (XDR)
The Presentation layer establishes a context between application layer entities, in which the higher-layer entities can use different syntax and semantics, as long as the Presentation Service understands both and the mapping between them.
Application (NFS)
The application layer interfaces directly to and performs common application services for the application processes; it also issues requests to the presentation layer.

Information to be sent is broken up and reformatted into a number of TCP/IP packets. Different types of information and/or different programs use different packet types

FTP (File Transfer Protocol)

FTP or File Transfer Protocol is used to transfer data from one computer to another over the Internet, or through a network.

Specifically, FTP is a commonly used protocol for exchanging files over any network that supports the TCP/IP protocol (such as the Internet or an intranet). There are two computers involved in an FTP transfer: a server and a client. The FTP server, running FTP server software, listens on the network for connection requests from other computers. The client computer, running FTP client software, initiates a connection to the server. Once connected, the client can do a number of file manipulation operations such as uploading files to the server, download files from the server, rename or delete files on the server and so on. Any software company or individual programmer is able to create FTP server or client software because the protocol is an open standard. Virtually every computer platform supports the FTP protocol. This allows any computer connected to a TCP/IP based network to manipulate files on another computer on that network regardless of which operating systems are involved (if the computers permit FTP access). There are many existing FTP client and server programs. FTP servers can be set up anywhere between game servers, voice servers, internet hosts, and other physical servers.

HTTP (Hypertext Transfer Protocol)

Hypertext Transfer Protocol (HTTP) is a communications protocol used to transfer or convey information on intranets and the World Wide Web. Its original purpose was to provide a way to publish and retrieve hypertext pages. Development of HTTP was coordinated by the W3C (World Wide Web Consortium) and the IETF (Internet Engineering Task Force), culminating in the publication of a series of RFCs, most notably RFC 2616 (June 1999), which defines HTTP/1.1, the version of HTTP in common use.

HTTP is a request/response protocol between a client and a server. The client making an HTTP request - such as a web browser, spider, or other end-user tool - is referred to as the user agent. The responding server - which stores or creates resources such as HTML files and images - is called the origin server. In between the user agent and origin server may be several intermediaries, such as proxies, gateways, and tunnels. HTTP is not constrained to using TCP/IP and its supporting layers, although this is its most popular application on the Internet. Indeed HTTP can be "implemented on top of any other protocol on the Internet, or on other networks. HTTP only presumes a reliable transport; any protocol that provides such guarantees can be used."

Typically, an HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a host (port 80 by default; see List of TCP and UDP port numbers). An HTTP server listening on that port waits for the client to send a request message.

Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message of its own, the body of which is perhaps the requested file, an error message, or some other information.

Resources to be accessed by HTTP are identified using Uniform Resource Identifiers (URIs) (or, more specifically, Uniform Resource Locators (URLs)) using the http: or https URI schemes.

Telnet (Terminal Emulation Protocol)

A terminal emulator, terminal application, term, or tty for short, is a program that emulates a "dumb" video terminal within some other display architecture. Though typically synonymous with a command line shell or text terminal, the term terminal covers all remote terminals, including graphical interfaces. A terminal emulator inside a graphical user interface is often called a terminal window.

A terminal window allows the user access to a text terminal and all its applications such as command line interfaces (CLI) and text user interface applications. These may be running either on the same machine or on a different one via telnet, ssh, or dial-up. On Unix-like operating systems it is common to have one or more terminal windows connected to the local machine.

Terminals usually support a set of escape sequences for controlling color, cursor position, etc. Examples include the family of terminal control sequence standards known as ECMA-48, ANSI X3.64 or ISO/IEC 6429. The xterm, a popular terminal emulator designed for X11.

Early adopters of computer technology, such as banks, insurance companies, and governments, still make frequent use of terminal emulators. They typically have decades old applications running on mainframe computers. The old “dumb” video terminals used to access the mainframe are long since obsolete; however, applications on the mainframe are still in use. Quite often, terminal emulators are the only way a user can access applications running on these older machines.

SMTP (Simple Mail Transfer Protocol)

SMTP is a relatively simple, text-based protocol, in which one or more recipients of a message are specified (and in most cases verified to exist) along with the message text and possibly other encoded objects. The message is then transferred to a remote server using a procedure of queries and responses between the client and server. Either an end-user's email client, a.k.a. MUA (Mail User Agent), or a relaying server's MTA (Mail Transport Agents) can act as an SMTP client.

An email client knows the outgoing mail SMTP server from its configuration. A relaying server typically determines which SMTP server to connect to by looking up the MX (Mail eXchange) DNS record for each recipient's domain name (the part of the email address to the right of the at (@) sign). Conformant MTAs (not all) fall back to a simple A record in the case of no MX. Some current mail transfer agents will also use SRV records, a more general form of MX, though these are not widely adopted. (Relaying servers can also be configured to use a smart host.)

The SMTP client initiates a TCP connection to server's port 25 (unless overridden by configuration). It is quite easy to test an SMTP server using the telnet program (see below).

SMTP is a "push" protocol that does not allow one to "pull" messages from a remote server on demand. To do this a mail client must use POP3 or IMAP. Another SMTP server can trigger a delivery in SMTP using ETRN.

Bittorrent (Data Protocol – used for movies)

BitTorrent is a peer-to-peer file sharing (P2P) communications protocol. BitTorrent is a method of distributing large amounts of data widely without the original distributor incurring the entire costs of hardware, hosting and bandwidth resources. Instead, when data is distributed using the BitTorrent protocol, each recipient supplies pieces of the data to newer recipients, reducing the cost and burden on any given individual source, providing redundancy against system problems, and reducing dependence on the original distributor.

The protocol is the brainchild of programmer Bram Cohen, who designed it in April 2001 and released a first implementation on 2 July 2001. It is now maintained by Cohen's company BitTorrent, Inc.

Usage of the protocol accounts for significant traffic on the Internet, but the precise amount has proven difficult to measure.

There are numerous compatible BitTorrent clients, written in a variety of programming languages, and running on a variety of computing platforms.

1.1.3 Building Block – TCP/IP

The TCP/IP model or Internet reference model, sometimes called the DoD model (DoD, Department of Defense) ARPANET reference model, is a layered abstract description for communications and computer network protocol design. It was created in the 1970s by DARPA for use in developing the Internet's protocols, and the structure of the Internet is still closely reflected by the TCP/IP model.

The original TCP/IP reference model consists of 4 layers, but has according to some authors evolved into a 5-layer model, where the lowest layer (the network access layer) is split into a physical layer and a datalink layer. However, no IETF standards-track document has accepted a five-layer model, probably since physical layer and data link layer protocols are not standardized by IETF. IETF documents deprecate strict layering of all sorts. Given the lack of acceptance of the five-layer model by the body with technical responsibility for the protocol suite, it is not unreasonable to regard five-layer presentations as teaching aids, making it possible to talk about non-IETF protocols at the physical layer.

This model was developed before the OSI Reference Model, and the Internet Engineering Task Force (IETF), which is responsible for the model and protocols developed under it, has never felt obligated to be compliant with OSI. While the basic OSI model is widely used in teaching, OSI, as presented as a seven-layer model, does not reflect real-world protocol architecture (RFC 1122) as used in the dominant Internet environment.

An updated IETF architectural document [1] even contains a section entitled: "Layering Considered Harmful". Emphasizing layering as the key driver of architecture is not a feature of the TCP/IP model, but rather of OSI. Much confusion comes from attempts to force OSI-like layering onto an architecture that minimizes their use.

For more detailed explanation of the TCP/IP model please visit the following website: http://en.wikipedia.org/wiki/TCP/IP_model.

1.1.4 Growth of the Internet

It is safe to say that the Internet has grown at more than an exponential rate over the past few years. Recent statistics has shown that 20 % of the world's population has access to the Internet and more importantly the remaining 80% are beginning to be added to the Internet community as more and more countries add the necessary infrastructure to provide Internet access.

More importantly to note is that typically under-developed countries are slowly starting to become Internet accessible as it becomes more and more evident that a strong economy is tied to the amount and quality of Internet access that is needed by commerce, education, entertainment, and government.

To review more detailed statistics about the Internet please visit the following website: http://www.internetworldstats.com/.

1.1.5 Common uses of the Internet

The Internet is a worldwide, publicly accessible series of interconnected computer networks that transmit data by packet switching using the standard Internet Protocol (IP). It is a "network of networks" that consists of millions of smaller domestic, academic, business, and government networks, which together carry various information and services, such as electronic mail, online chat, file transfer, and the interlinked web pages and other resources of the World Wide Web.

Over the past few years the Internet is being used more for entertainment and collaboration as evidenced by the popularity of social networking websites like My Space and Facebook. Recent statistics recently published by IBM show that people spend more time on the Internet than they do watching TV: http://broadcastengineering.com/iptv/ibm-internet-overtaking-tv-0828/.

1.2. HTML Tags

1.2.1 Tag Basics

The basic HTML tags are quite simple. Firstly, we need to understand how the **H**yper **T**ext **M**arkup **L**anguage syntax operates and formats the surrounding text.

An HTML tag is wrapped with angle brackets (less than **<** and the - greater **>** than characters),

A typical tag - <title> - is composed of 3 parts:

1. < = opening angle bracket
2. title = tag name
3. > = closing angle bracket

- A good majority of tags come in pairs and surround the text they will act upon. The first tag turns the action on and the second tag turns it off.

<title>Title of Webpage</title>

The second tag, let's call that the "off switch", starts with a **forward slash** - / - to designate the end of the operation.

For example: The bold tag works like this <b>**makes this text bold**</b>.

- Like any programming language there are exceptions to the rule. For instance, the <br> tag creates a line break and doesn't have an “**off switch**”. The <br> tag simply adds a line break in between your text.

- Along with HTML tags there are attributes to take into consideration. Attributes can apply values to the tag instructions. For example, the <p> tag starts a new paragraph. If you add an alignment attribute, it will change the placement of the new paragraph in the example below we have centered the text

<p align=”center”>Example of a paragraph aligned to the center</p>

Once you start writing code yourself it will start to make more sense. But first, you need to open a text editor. For this module we will be using the text editor on your Windows computer: Notepad. The Notepad program can be found under the Programs/Accessories menu. Follow these instructions from the desktop:

Click the Start button
Choose Accessories
Click on Notepad

You should now see the Notepad window open. You will use Notepad to type in your HTML code in the upcoming topic.

1.2.2 HTML Layout

Now that you have your Notepad editor open following along in the upcoming topic by typing in the HTML code below.

All webpages must start with the **D**ocument **T**ype **D**efinition code at the very top of the webpage. You will include the **DTD** on every webpage that you create as shown below.

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

In the next line of code the browser will be looking for the **<html>** tag.

The browser will begin reading the webpage once it finds the following <html> tag

<html>

At the end of the document, you need to close the HTML code by typing in the closing tag

</html>

Step 1

Your code should now look like this, begin to type this code into a Notepad document

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
        <html>
        </html>
```

There are two parts to a **HTML** document:

1. **head**
2. **body**.

The **head** is used to display specific information having to do with the web webpage and the **body** is where you put the content that you want viewers to see.

Step 2

Let's add the **head** section right after the opening <html> tag as shown below:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
                <head>
</html>
```

Step 3

Now let's give the webpage a **title** as shown below:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
                <head><title>Lithuania Institute of Technology</title>
</html>
```

Step 4

Next, close the **head** section and open the **body** section:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
            <head><title>Lithuania Institute of Technology</title></head>

    <body>

</html>
```

Step 5

Type the first paragraph into your webpage:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
            <head><title>Lithuania Institute of Technology</title></head>

    <body>
Lithuania Institute of Technology is the best school in the country for GIS.
</html>
```

Step 6

To complete the webpage we must close the **body** section with the **</body>** tag:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
            <head><title>Lithuania Institute of Technology</title></head>
    <body>

Lithuania Institute of Technology is the best school in the country for GIS.

</body>
</html>
```

After the completing the six steps your HTML code should look now look like this:

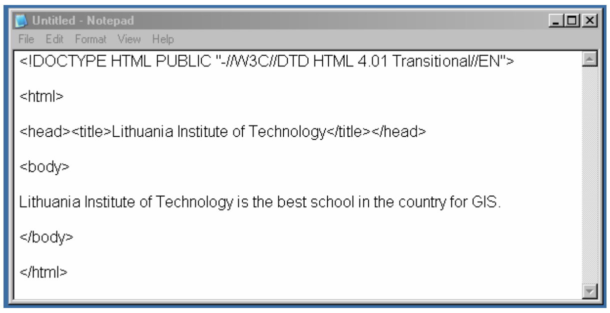
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
            <head><title>Lithuania Institute of Technology</title></head>
    <body>

Lithuania Institute of Technology is the best school in the country for GIS.

</body>
</html>
```

Please Note:

The text has been made **bold** and the returns and spacing have been placed in the code above just to make the code simpler for you to read.

1.2.3 Save and View an HTML Document

You have just created your first simple webpage but to complete the process we must save the file. In Notepad, you should have typed in the following code:



```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head><title>Lithuania Institute of Technology</title></head>

<body>

Lithuania Institute of Technology is the best school in the country for GIS.

</body>

</html>
```

To save the file, follow these directions:

Click on the **File** option on the top tool bar and choose **Save As**.

Save it as a "text only" document and give it this name: **firstpage.html**

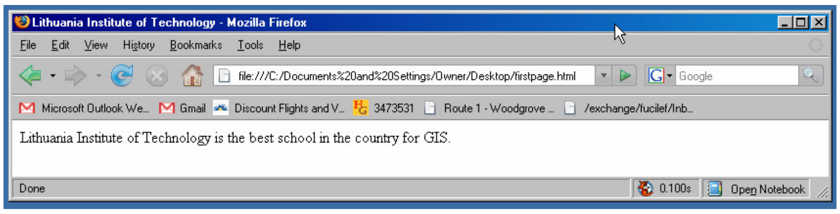
- Save it to your windows **desktop.**

- Next, open a browser (Internet Explorer, Netscape, Mozilla, or any other browser you might have). For this example I will use Firefox as my Internet browser

Click on the **File** option from the top tool bar, and highlight **Open File**.

From the next screen, choose your **firstpage.html** file from the desktop:

You should now see your webpage displayed by your browser like this...


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/Owner/Desktop/firstpage.html
Google
Microsoft Outlook We... Gmail Discount Flights and V... 3473531 Route 1 - Woodgrove ... /exchange/fucilef/Inb...
Done 0.100s Open Notebook

Lithuania Institute of Technology is the best school in the country for GIS.

Now let's look a few things about your first simple webage:

The **title** is way up at the top of the webpage in the **Title Bar** of the browser – Lithuania Institute of Technology

The only thing in the **body** of your website is the one sentence you typed

Lithuania **Institute of Technology is the best school in the country for GIS.**

- The location bar is showing the address to your own computer as local host because the file is located on the desktop of our computer

1.2.4 More Tags

Continuing from the steps above let's add some tags to your webpage to make it more interesting and easier to read.

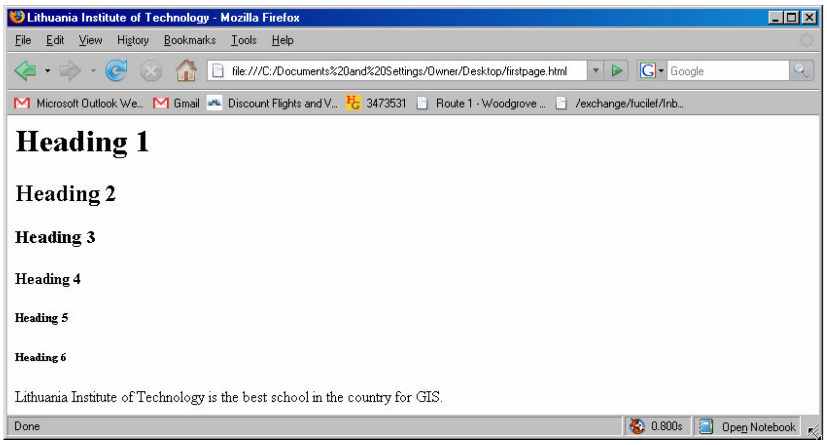
The **Heading** Tag

The **heading** tag tells the browser to make the font larger and bold. Most browsers can read heading tags between 1 and 6. After the <body> tag and before the beginning of the sentence, add the following heading tags.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
                <head><title>Lithuania Institute of Technology</title></head>
    <body>
    <h1>Heading 1</h1>
    <h2>Heading 2</h2>
    <h3>Heading 3</h3>
    <h4>Heading 4</h4>
    <h5>Heading 5</h5>
    <h6>Heading 6</h6>
Lithuania Institute of Technology is the best school in the country for GIS.
</body>
</html>
```

To view the new heading tags save your **firstpage.html** file. Go back to the browser, and click on the Refresh or Reload button at the top of the screen, as shown in the image below.

You should now see the following webpage. Notice the relative sizes of the heading tags – H1 it biggest and H6 is smallest.


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/Owner/Desktop/firstpage.html
Google
Microsoft Outlook We... Gmail Discount Flights and V... 3473531 Route 1 - Woodgrove ... /exchange/lucilef/Inb...
Done
0.800s
Open Notebook

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

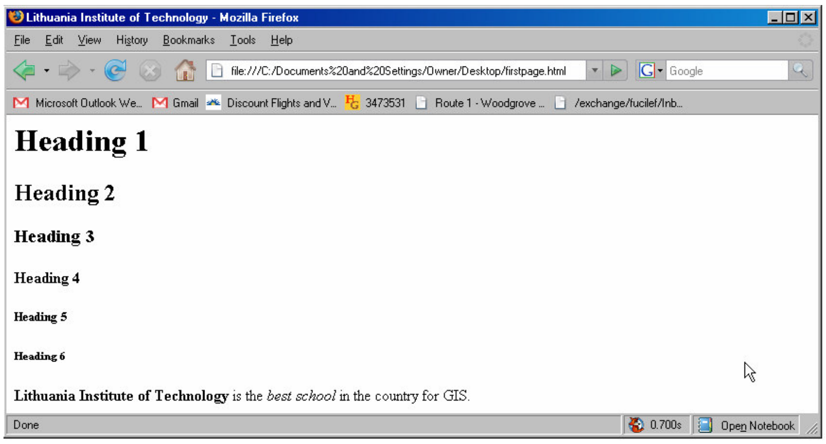
Lithuania Institute of Technology is the best school in the country for GIS.

The **Bold** and **Italic** Tag

You can highlight individual words or phrases by surrounding them with the **<b>** or **<i>** tags. Try putting these tags around some words in the paragraph:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
                <head><title>Lithuania Institute of Technology</title></head>
```

```
          <body>
          <h1>Heading 1</h1>
          <h2>Heading 2</h2>
          <h3>Heading 3</h3>
          <h4>Heading 4</h4>
          <h5>Heading 5</h5>

          <h6>Heading 6</h6>

      <b>Lithuania Institute of Technology</b> is the <i>best school</i> in the country for GIS.

      </body>
      </html>
```


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/Owner/Desktop/firstpage.html
Google
Microsoft Outlook We... Gmail Discount Flights and V... 3473531 Route 1 - Woodgrove ... /exchange/fucilef/Inb...
Done
0.700s
Open Notebook

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

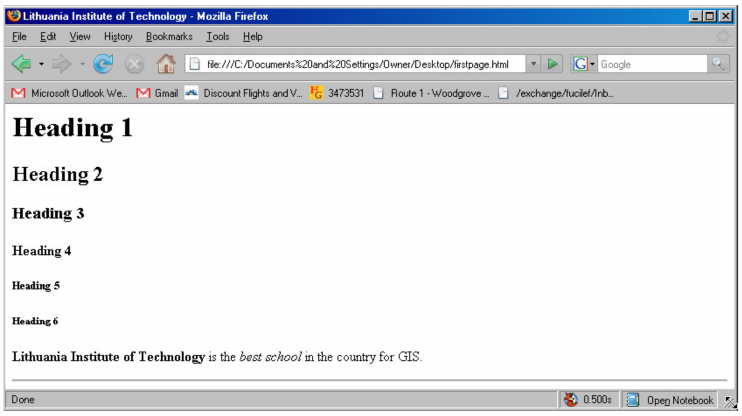
Lithuania Institute of Technology is the *best school* in the country for GIS.

The **Paragraph** and **Horizontal Rule** Tag

Let's add a horizontal bar at the end of the webpage. First, you need to surround your sentence with the paragraph tags **<p>**, this will give you space around the sentence. Next, put a horizontal rule tag at the end: **<hr>**. The horizontal rule tag doesn't require an open and close tag.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

                    <head><title>Lithuania Institute of Technology</title></head>
      <body>
      <h1>Heading 1</h1>
      <h2>Heading 2</h2>
      <h3>Heading 3</h3>
      <h4>Heading 4</h4>
      <h5>Heading 5</h5>

      <h6>Heading 6</h6>

<p><b>Lithuania Institute of Technology</b> is the <i>best school</i> in the country for
GIS.</p>

<hr>

</body>
</html>
```

To view the new tags save your **firstpage.html** file. Go back to the browser, and click on the Refresh or Reload button at the top of the screen, as shown in the image below.


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/Owner/Desktop/firstpage.html
Google
Microsoft Outlook We... Gmail Discount Flights and V... 3473531 Route 1 - Woodgrove ... /exchange/lucilef/Inb...
Done
0.500s
Open Notebook

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

Lithuania Institute of Technology is the *best school* in the country for GIS.

1.2.5 Advanced Tags

In the previous examples you have learned a number of the basic tags and how to use them. There are great many more HTML tags that you can use to create your webpages and websites. It is not necessary to memorize or learn all the tags that are available to use. You can simply refer to an “index” of tags provided by the W3C – http://www.w3.org/

The World Wide Web Consortium (W3C) develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential. W3C is a forum for information, commerce, communication, and collective understanding. Below is a link from the W3 organization of all of the HTML tags available to use in your webpages.

http://www.w3.org/TR/html401/index/elements.html

1.2.6 Tag Attributes

To extend the ability of tags, there are **attributes** that can be assigned to the tags. Attributes are added to the tags inside the bracket and set to a specific value; the attributes are enclosed in quotes.

Align Attributes

Align attributes puts text in a specific place on the webpage. For example;

- <p align=**"center"**> will align the paragraph to the center of the webpage.
- <p align=**"right"**> will align it to the right side of the screen.

Notice that the attribute is formatted like this:

1. First, the name of the attribute: <p **align**="center">
2. Second, an equal sign: <p align**=**"center">
3. And finally, the value you want, enclosed in quotes: <p align="**center**">

Please note, there are **NO** spaces between any of these: **align="center"**.

However, there is a space between the tag and the attribute: <p align="center">

Width and Size Attributes

There are width and size attributes that can be applied to certain tags

For example, you can change the size of the horizontal rule by adding width and size attributes. Notice that all the attribute values are enclosed in quotes:

<hr align="center" width="400" size="5">

In this case, the line will be 400 pixels across the screen (a 15" screen is about 700 pixels wide) and 5 pixels thick.

Various attributes can be added to the tags in any order.

<hr align="center" width="500" size="20"> or

<hr width="500" align="center" size="20">

Let's give it a try. Open your text editor and try different combinations in your **firstpage.html**

Try different sizes and alignments on the horizontal rule:

<p><b>Lithuania Institute of Technology</b> is the <i>*best school*</i> in the country for GIS.</p>

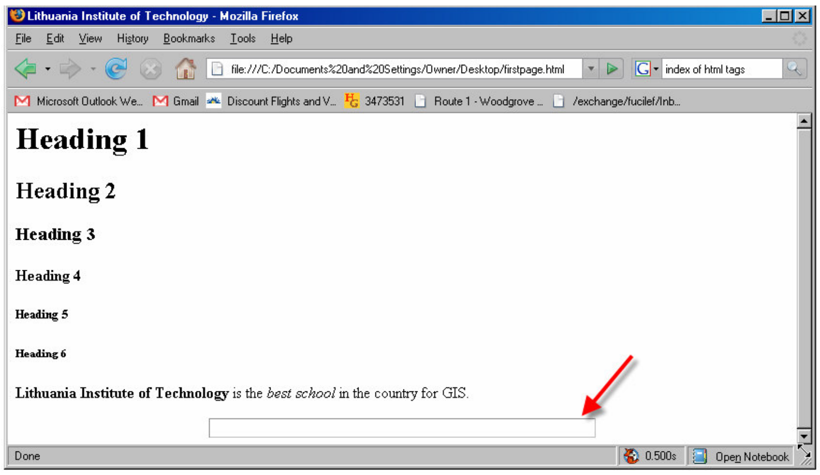
<hr align="center" width="400" size="20">

</body>
</html>

Remember, before you can see the effect of your code, you'll need to:

Save your file **firstpage.html**).

Then, **open** a browser (Explorer, Netscape, Mozilla) and **open** your **firstpage.html**


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/Owner/Desktop/firstpage.html
index of html tags
Microsoft Outlook We... Gmail Discount Flights and V... 3473531 Route 1 - Woodgrove ... /exchange/fucilef/Inb...
Done
0.500s
Open Notebook

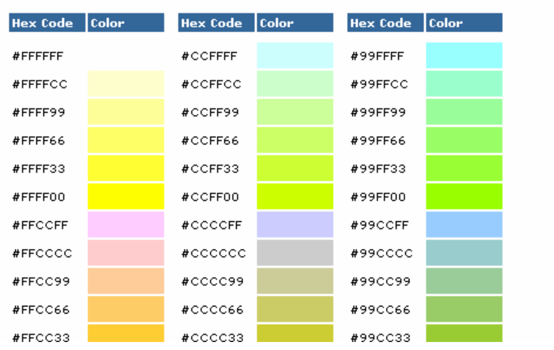
Heading 1
Heading 2
Heading 3
Heading 4
Heading 5
Heading 6

Lithuania Institute of Technology is the *best school* in the country for GIS.

Color Attributes

You can also add color to your webpage using attributes. First, you'll have to find the special codes, called **hexadecimal codes** that will translate into colors. Try this website to get a listing of all the hexadecimal color codes: http://www.webmonkey.com/reference/color_codes/.

Once you have a listing of all the codes you can then simply copy and paste the code you want to you use.



Hex Code	Color	Hex Code	Color	Hex Code	Color
#FFFFFF	⬜	#CCFFFF	🟦	#99FFFF	🟦
#FFFFCC	🟨	#CCFFCC	🟩	#99FFCC	🟩
#FFFF99	🟨	#CCFF99	🟩	#99FF99	🟩
#FFFF66	🟨	#CCFF66	🟩	#99FF66	🟩
#FFFF33	🟨	#CCFF33	🟩	#99FF33	🟩
#FFFF00	🟨	#CCFF00	🟩	#99FF00	🟩
#FFCCFF	🟪	#CCCCFF	🟪	#99CCFF	🟦
#FFCCCC	🟥	#CCCCCC	⬜	#99CCCC	🟦
#FFCC99	🟧	#CCCC99	🟫	#99CC99	🟩
#FFCC66	🟧	#CCCC66	🟨	#99CC66	🟩
#FFCC33	🟧	#CCCC33	🟨	#99CC33	🟩

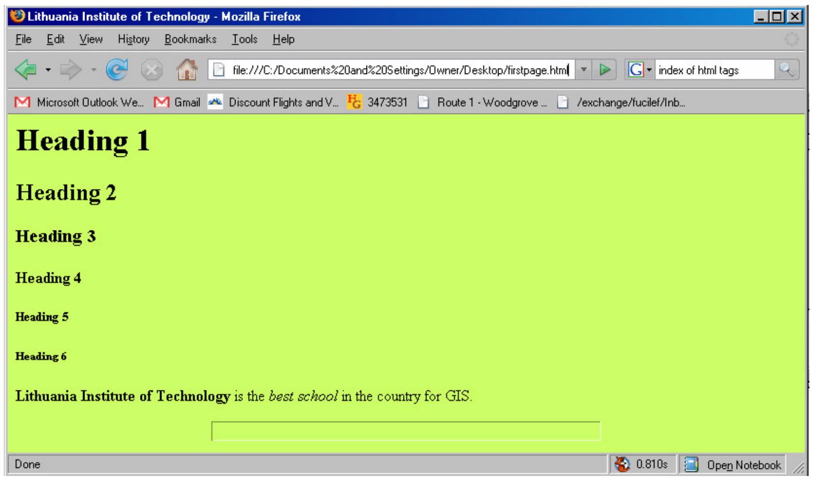
Background Color

For the background of your webpage it is important to use a soft, light color so that the text will contrast and be easily read. For example, #CCFF66 is a soft, light yellow color that shows up dark text easily.

```
<body bgcolor="#CCFF66">
```

Notice that there is no space between the attribute **bgcolor**, the equal sign, or the value. The value, as always, is enclosed in quotation marks. In the case of **color** attribute, the value always begins with the number symbol (**#**).

Try making this change in your **<body>** tag. **Save** the webpage and **Reload** it in the browser to make sure you like the color of the background.


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/Owner/Desktop/firstpage.html
index of html tags
Microsoft Outlook We...
Gmail
Discount Flights and V...
3473531
Route 1 - Woodgrove ...
/exchange/fucilef/Inb...
Done
0.810s
Open Notebook

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

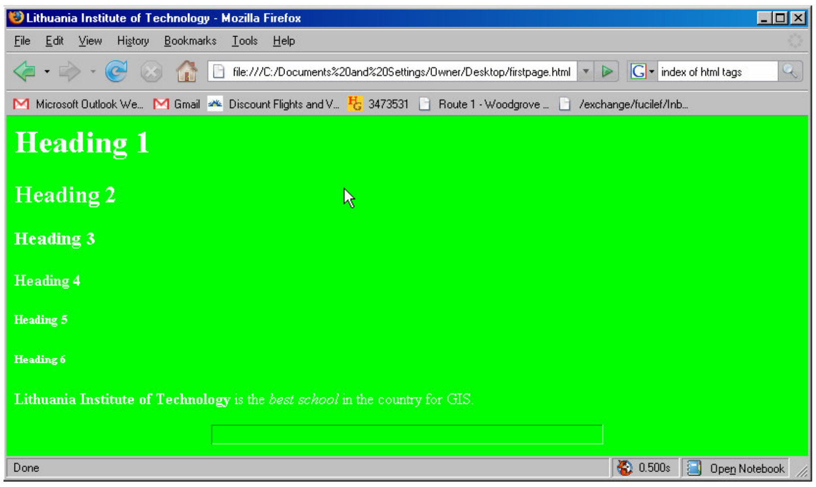
Lithuania Institute of Technology is the *best school* in the country for GIS.

Text Color

You can also change the text colors with these codes:

<body bgcolor=#00ff00 text="#ffffff">

This will change the background again, plus change the color of all the text. **Save** and **Reload** to see the changes.


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/Owner/Desktop/firstpage.html
index of html tags
Microsoft Outlook We... Gmail Discount Flights and V... 3473531 Route 1 - Woodgrove ... /exchange/fucilef/Inb...
Done
0.500s
Open Notebook

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

Lithuania Institute of Technology is the *best school* in the country for GIS.

Link Color

You can also change the color of hypertext links and you can place all this information into the same **<body>** tag as shown here:

<body bgcolor="#FFCCCC" text="#000000" link="#990000" alink="#009900" vlink="#330033">

What do the **<body>** attributes represent?

- **text =** is the base text color on the webpage
- **link =** is the color of the link text on the webpage
- **vlink =** is the color of an already visited link
- **alink =** makes the link text blink another color when activating.

In the above example the code is:

- going to set the text to black
- the links to bright red
- the activated link to dark green
- and the visited link to dark purple.

At the present time you don't have any links on your webpage so you won't see any of these link color changes. We will be looking at links in the Topic 3.

Also, it makes no difference if you type the color values in CAPITAL or small letters as long as you have the **#** symbol and the surrounding quotation marks. For example:

bgcolor="#FFCCCC" **is equal to this** bgcolor="#ffcccc"

But, what if you don't want to change all the text? You just want to change the color on the first heading, maybe to a bright red color.

First, change your **<body>** tag back to a softer color and take out the **text** attribute:

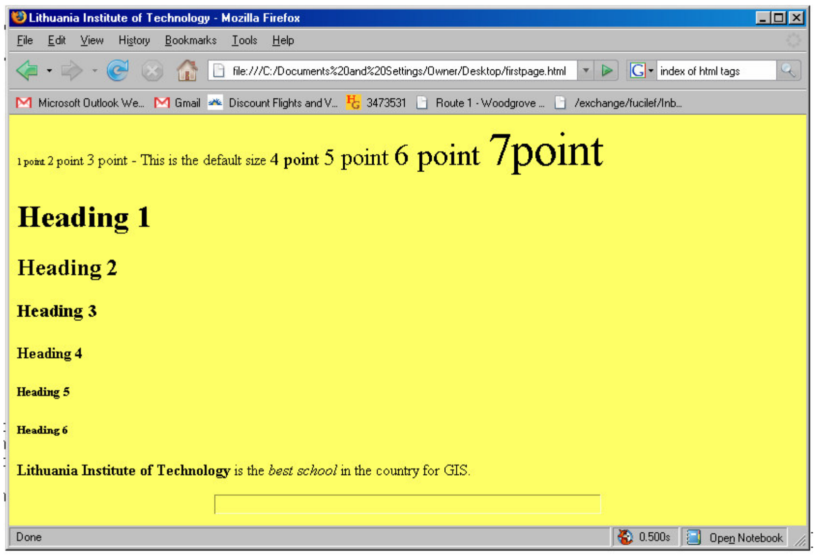
<body bgcolor="#ffff66">

Font Attributes

There are font attributes to change the size of the text. The original way to assign font sizes using Font tag is deprecated in the latest versions of HTML (HTML 4 and XHTML). Style sheets (CSS) will be used to define the layout and display properties of HTML elements we will covering that in more depth in Topic 8.

```
<font size="1">1 point</font>
<font size="1">2 point</font>
<font size="1">3 point</font>
<font size="1">4 point</font>
<font size="1">5 point</font>
<font size="1">6 point</font>
<font size="1">7 point</font>
```

Add the font size code to your firstpage.html just so that you can see how it looks so that you can see the relative size of the font.


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/Owner/Desktop/firstpage.html
index of html tags
Microsoft Outlook We...
Gmail
Discount Flights and V...
3473531
Route 1 - Woodgrove ...
/exchange/fucilet/Inb...
Done
0.500s
Open Notebook

1 point 2 point 3 point - This is the default size 4 point 5 point 6 point 7point

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

Lithuania Institute of Technology is the *best school* in the country for GIS.

1.2.7 Lets Make a List

People read webpages differently than they do a newspaper or magazine article. Web readers will skim over the webpage quickly and they do not read a great deal of information. The eyes will jump to the bigger bold headings, and to the itemized lists of topics. It is also important to note that most people read between 33% and 66% slower off a computer screen than they read from paper, and so to make it easier they will read the text that stand out. This explains why there are so many specific ways to code lists.

- There are bulleted lists like this.

- Each item starts with a little symbol.
- This is an unordered list = **<ul>**

1. There are numbered lists like this.
2. Each item starts with a number or letter.
3. This is an ordered list = **<ol>**

And there are definition lists like this. Each item is indented, but has no symbol or number.

This is a definition list = **<dl>**

Unordered List

Code for an unordered list places the items in sequence and then you can assign various attributes types for the bullet image.

HTML Code	Browser
<ul> **<li>**This is the first item.**</li>** **<li>**This is the second item**</li>** **</ul>**	● This is the first item ● This is the second item

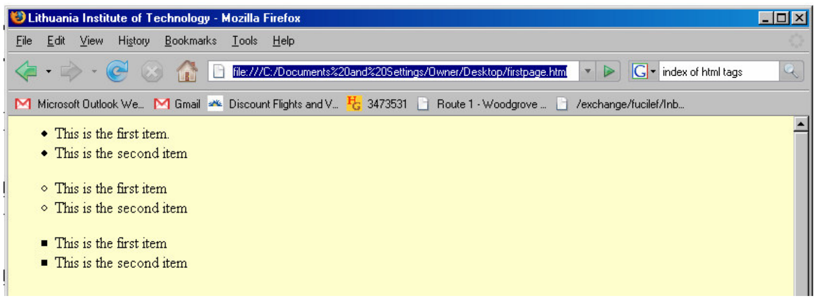
Notice that you'll first open the **<ul>** tag, then the **<li>** tag, and at the end, you'll close the **</li>** tag, and then, the **</ul>** tag. The order is always **first in, last out**. These are called **nested tags**.

You can change the black disk bullet to a square or empty circle by adding a **type** attribute to the **list** tag:

HTML Code	Browser
<ul type="circle"> **<li>** This is the first item **</li>** **<li>** This is the second item**</li>** **</ul>**	○ This is the first item ○ This is the second item

HTML Code	Browser
<ul type="square"> **<li>** This is the first item **</li>** **<li>** This is the second item**</li>** **</ul>**	❐ This is the first item ❐ This is the second item

Type the unordered list code in your firstpage.html file and then view it in the browser. It should look like this:


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/Owner/Desktop/firstpage.html
index of html tags
Microsoft Outlook We... Gmail Discount Flights and V... 3473531 Route 1 - Woodgrove ... /exchange/fucilef/Inb...

- This is the first item.
- This is the second item

○ This is the first item
○ This is the second item

■ This is the first item
■ This is the second item

Ordered List

Code for an ordered list places the list items in specific sequence and uses different bullet types as well.

For example:

HTML Code	Browser
<ol>	1. Place flour in bowl
<**li**>Place flour in bowl</**li**>	2. Add water to bowl of flour
<**li**>Add water to bowl of flour</**li**>	3. Mix according to directions
<**li**>Mix according to directions</**li**>	
</**ol**>	

You can also change the number for letters by adding a **type** attribute to the **list** tag:

HTML Code	Browser
<ol type="A">	A. Place flour in bowl
<**li**>Place flour in bowl</**li**>	B. Add water to bowl of flour
<**li**>Add water to bowl of flour</**li**>	C. Mix according to directions
<**li**>Mix according to directions</**li**>	
</**ol**>	

HTML Code	Browser
<ol type="i">	i. Place flour in bowl
<**li**>Place flour in bowl</**li**>	ii. Add water to bowl of flour
<**li**>Add water to bowl of flour</**li**>	iii. Mix according to directions
<**li**>Mix according to directions</**li**>	
</**ol**>	

You can also start your list at a specific number or letter, in case you have to interrupt your list for comments or instructions. You will use the **start** attribute for this purpose. Notice that you'll have to use the number **3** to signify the 3rd letter of the alphabet.

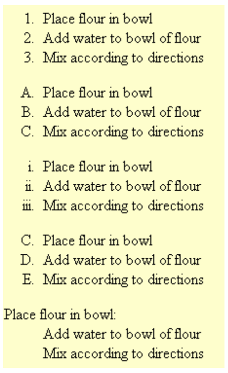
HTML Code	Browser
<ol type="A" start="3">	C. Place flour in bowl
<**li**>Place flour in bowl</**li**>	D. Add water to bowl of flour
<**li**>Add water to bowl of flour</**li**>	E. Mix according to directions
<**li**>Mix according to directions</**li**>	
</**ol**>	

Definition List

Code for a definition list looks like this:

HTML Code	Browser
<**dl**> <**dt**>Place flour in bowl: <**dd**>Add water to bowl of flour</**dd**> <**dd**>Mix according to directions</**dd**> </**dl**>	Place flour in bowl: Add water to bowl of flour Mix according to directions

Place the above list code in your **firstpage.html** and have a look at in your browser. It should look like this:



```
 1. Place flour in bowl
 2. Add water to bowl of flour
 3. Mix according to directions

 A. Place flour in bowl
 B. Add water to bowl of flour
 C. Mix according to directions

  i. Place flour in bowl
 ii. Add water to bowl of flour
iii. Mix according to directions

 C. Place flour in bowl
 D. Add water to bowl of flour
 E. Mix according to directions

Place flour in bowl:
    Add water to bowl of flour
    Mix according to directions
```

Reasons to Use Lists

1. Lists are handier to use than creating simulated lists by coding your own numbers because you can insert a list item at any point and the browser will automatically renumber all items.

Using a list means that all your text for each list item is **indented**. Coding list by hand means that long sentences or paragraphs will wrap under the number instead of remaining indented.

Using the **start** attribute with ordered lists means that you can interrupt your list at any point by canceling it, adding commentary, and then restarting the list at the number or letter you left off with.

You can use other **HTML** tags within a list, such as **paragraph** tags to create separation between list items or **bold** and **italics** tags for emphasis.

1.3. Links

1.3.1 Format of Links

The web works so successfully because of the use of links. Links create the 'hypertext' world that makes the web so powerful and easy to use. The **anchor** tags make all this possible.

To make a link to a website, surround your link text with **anchor tags**. Here's the formula:

- Start and end with the basic anchor tag: **<a>**........ **</a>**
- Within the opening tag, add the **href** attribute - hyperlink reference. The value, enclosed in quotes, will tell the URL that you want to connect to

  <a href="http://www.maps.lt/">....... **</a>**

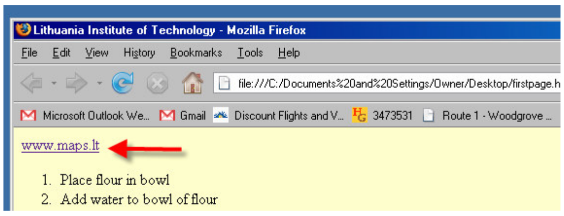
- In the space between the opening and closing tags, type in the words that will appear as the hyperlink. For example,

  <a href="http://www.maps.lt/">www.maps.lt</a>

  Notice that there are no spaces from the attribute **href** to the final closing bracket of the closing **</a>**

Also, the URL is in quotes, just like all your other attributes. Place the link code into your firstpage.html webpage.

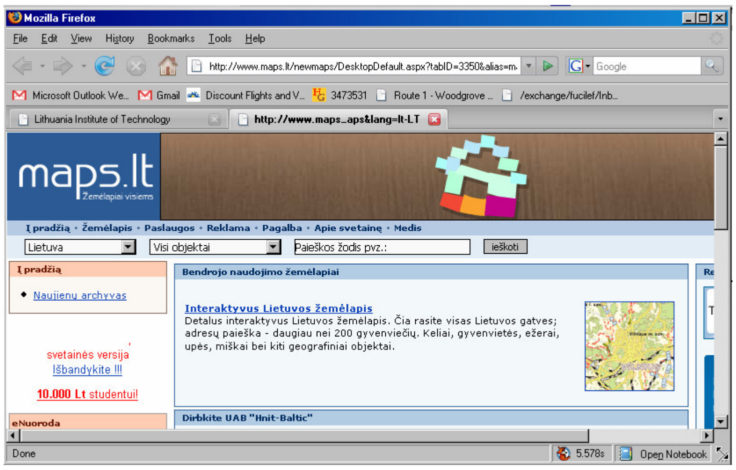
This will create a link on your page that says: **www.maps.lt.** It will usually be printed in blue (unless the link color has been changed), it will be underlined.


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/Owner/Desktop/firstpage.h
Microsoft Outlook We... Gmail Discount Flights and V... 3473531 Route 1 - Woodgrove ...

www.maps.lt

1. Place flour in bowl
2. Add water to bowl of flour

When someone clicks their mouse button on it, the browser will go to **http://www.maps.lt/** homepage.


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://www.maps.lt/newmaps/DesktopDefault.aspx?tabID=3350&alias=m
Microsoft Outlook We... Gmail Discount Flights and V... 3473531 Route 1 - Woodgrove ... /exchange/fucilef/Inb...
Lithuania Institute of Technology
http://www.maps...aps&lang=lt-LT
maps.lt
Žemėlapiai visiems
Į pradžią • Žemėlapis • Paslaugos • Reklama • Pagalba • Apie svetainę • Medis
Lietuva
Visi objektai
Paieškos žodis pvz.:
Ieškoti
Į pradžią
Naujienų archyvas
svetainės versija
Išbandykite !!!
10.000 Lt studentui!
eNuoroda
Bendrojo naudojimo žemėlapiai
Interaktyvus Lietuvos žemėlapis
Dirbkite UAB "Hnit-Baltic"
Done
5.578s
Open Notebook

Detalus interaktyvus Lietuvos žemėlapis. Čia rasite visas Lietuvos gatves; adresų paieška - daugiau nei 200 gyvenviečių. Keliai, gyvenvietės, ežerai, upės, miškai bei kiti geografiniai objektai.

1.3.2 Lists of Links

You can add links to list items just by surrounding the list items with the anchor tags **<a>** as illustrated below. Here are a couple of examples:

Unordered List of Links

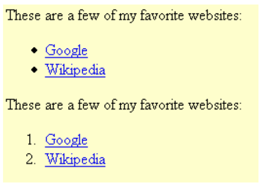
HTML Code	Browser
<p>These are a few of my favorite websites: **<ul>** **<li><a href="http://google.com">**Google**</a></li>** **<li>** **<a href="http://www.wikipedia.org">**Wikipedia**</a></li>** **</ul>** **</p>**	These are a few of my favorite things: Google Wikipedia

Ordered List of Links

HTML Code	Browser

<p>These are a few of my favorite websites:
    <ol>
    <li><a href="http://google.com">Google**</a></li>**
    <li>
    <a href="http://www.wikipedia.org">Wikipedia**</a></li>**
    </ol>
</p>
	These are a few of my favorite things:
	Google
	Wikipedia

Place the above code in your firstpage.html file, save and refresh your browser and you should now see the following list of links


These are a few of my favorite websites:

- Google
- Wikipedia

These are a few of my favorite websites:

1. Google
2. Wikipedia

1.3.3 Targets

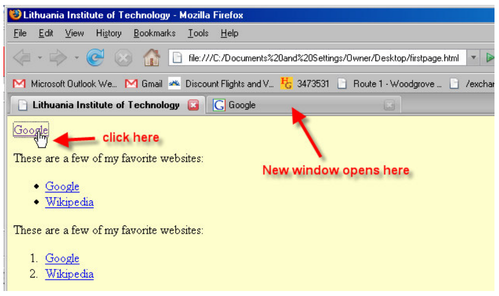
Here's another attribute that you can add to your **<a>** tags **target="_blank"**

This will cause the browser to open the new page in a new window. The original page stays available in the first window. Here's the full code:

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

There is one space between the closing quote of the URL and the word **target**.

Place the above code in your **firstpage.html** file, save and refresh your browser and you should now see the link to Google and when you click on the link "Google" and it will open in another window.


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/Owner/Desktop/firstpage.html
Lithuania Institute of Technology
Google
click here
New window opens here

Google

These are a few of my favorite websites:

- Google
- Wikipedia

These are a few of my favorite websites:

1. Google
2. Wikipedia

1.3.4 Anchors

Links can also connect to specific places in a website or webpage. On very long pages, this helps your reader move around the page more easily.

Anchors connect to places within the page,
Links connect to places outside the page,
but, they both use the <a> tag.

There are 2 steps to making an anchor:

- one to make the link to the specific place, and
- one to mark that place in the document.

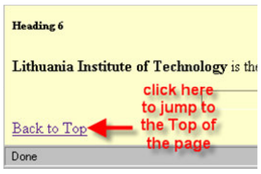
One of the most common anchor links is created so that you can quickly jump to the "top" of the page. Firstly, you'll mark it with a number symbol as shown in the code below. Place this code before the closing body tag:

```
<a href="#top">Back to Top</a>
</body>
```

Then you will use the <a name="top"> tag to mark the spot. Close the </a> tag right after you open it, since there are no words you want linked. This is just to mark the spot that you want the link to go to. Place this code right after the <body> tag:

```
<body>
<a name="top"></a>
```

Place the above code in your firstpage.html file, save and refresh your browser and you should now see this. Scroll down to the bottom of the page to see the "Back to Top" link:


Heading 6

Lithuania Institute of Technology is th

Back to Top

Done

1.3.5 Relative vs. Absolute

When you link to pages outside your own website, you will always use the absolute link, giving the full URL or web address of the outside page. But, when you link to another page on your own website, you have a choice.

For example, the link to the contact us page is simply given as **contactus.html**. The code is written like this:

```
<a href="firstpage.html">First Page</a>
```

This is not a complete URL or web address and wouldn't work from anywhere except from within your computer because at the present time the webpage is located on the desktop of your computer. The browser reads it as "find a document called **firstpage.html** that is in the same directory as the place you are in now. And, it's called a **relative link**.

To link to the absolute link you would write in the entire website address as shown below:

```
<a href="http://www.yourwebsite.com/firstpage.html>First Page</a>
```

This would be called an **absolute link** because it gives the absolute, full, and entire URL or web address. The browser reads it as "find a web address called **yourwebsite.com**, and on that website, find a document called **firstpage.html**

This concept will make more sense once you complete the assignment for Module 1

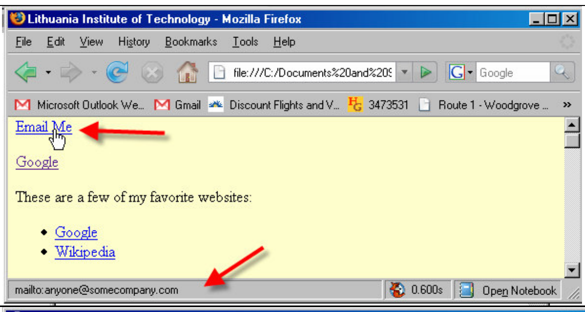
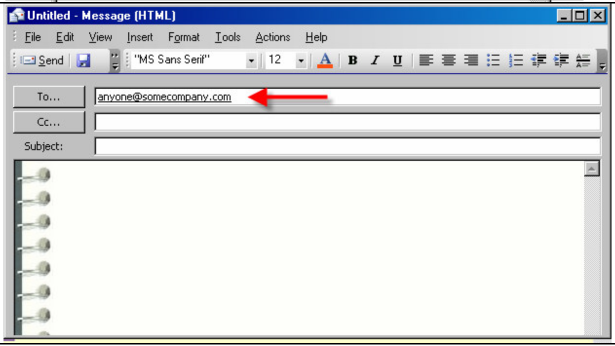
1.3.6 Email

Besides connecting to other websites, links can also connect to an email form, with someone's email address already in the TO: line. The code looks the same, except that:

- the **http:**// part is changed to **mailto:** (no forward slashes) and
- the URL is replaced with an email address.

Here's how it looks:

```
<a href="mailto:anyone@somecompany.com">Email Me</a>
```

If that line is put into the document, the words **Email Me** would be highlighted and linked to an email form with that address in the line. Please use a valid email address. The email address used above is invalid and is used only as an example.	 Email Me Google These are a few of my favorite websites: • Google • Wikipedia mailto:anyone@somecompany.com
When you click on the Email Me link the email program of the viewer will open and place the email address into the To… line of the program as shown below	 To… anyone@somecompany.com Cc… Subject:

1.4. Images

One picture is worth a thousand words, they say, but you have to be careful about pictures on a webpage. Pictures take a lot longer than words to load, and studies have determined that when people have to wait more than 10 seconds, they have a tendency to click to another website.

With that understanding, add pictures to your page. First, you have to have the picture. There are several images galleries on the web that offer free images and graphics. Here are a couple of websites to get you started.

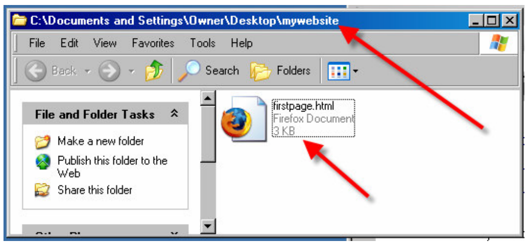
- http://images.google.com/
- http://www.freedigitalphotos.net/

You can always find more image websites by doing a quick search in Google.

1.4.1 Downloading Images

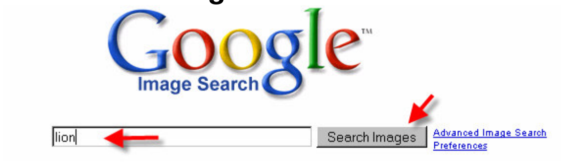
Before you can use an image on your webpage, you need to download it to your computer. Before you can download images, you need to create a folder to keep all your **HTML** documents and image files together.

- Create a new folder on the desktop (pull down the File option from the top horizontal menu bar and select New Folder). Name the folder mywebsite.
- Put your **firstpage.html** document into that folder.

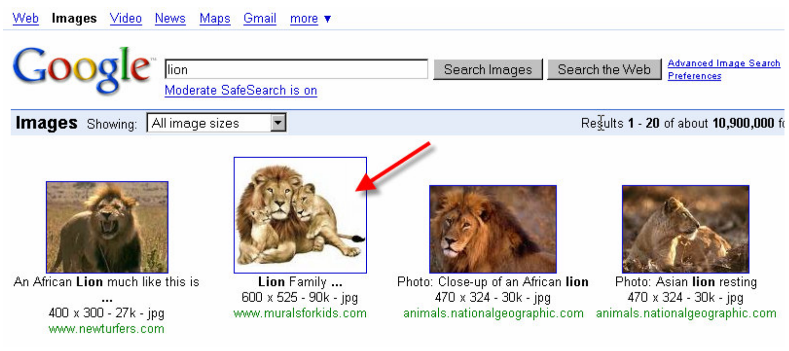

C:\Documents and Settings\Owner\Desktop\mywebsite
File Edit View Favorites Tools Help
Back
Search
Folders
File and Folder Tasks
Make a new folder
Publish this folder to the Web
Share this folder
firstpage.html
Firefox Document
3 KB

Let's use Google Images and look for an image to download. In your browser go to the following website address: http://images.google.com/.

For this example I will search for an image of a lion. Type lion in the search textbox and then click the **Search Images** button:

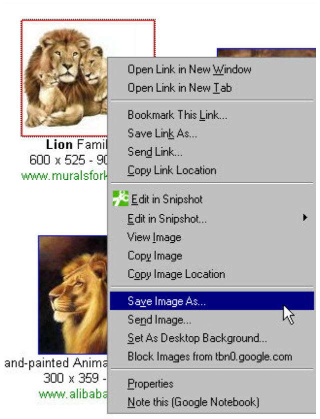


You will get a long list of images that have been found in the database. I will download the second image shown below.


Web Images Video News Maps Gmail more ▾
Google
lion
Search Images
Search the Web
Advanced Image Search
Preferences
Moderate SafeSearch is on
Images Showing: All image sizes
Results 1 - 20 of about 10,900,000
An African Lion much like this is ...
400 x 300 - 27k - jpg
www.newturfers.com
Lion Family ...
600 x 525 - 90k - jpg
www.muralsforkids.com
Photo: Close-up of an African lion
470 x 324 - 30k - jpg
animals.nationalgeographic.com
Photo: Asian lion resting
470 x 324 - 30k - jpg
animals.nationalgeographic.com

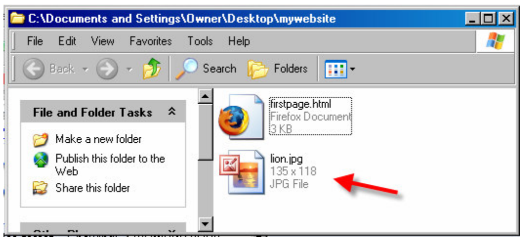
Follow these steps to download the image.

- Right click on the image.
- Choose the **Save Picture As** or **Save Image As** option.



In the Dialog Box that pops up, choose your new **mywebsite** folder to save in.

Name the image you're saving to something that you can remember. Some of the images you download will have a name like 178563AHt.gif - you should rename that to **lion.jpg**, or whatever the image is



1.4.2 Filenames and Formats

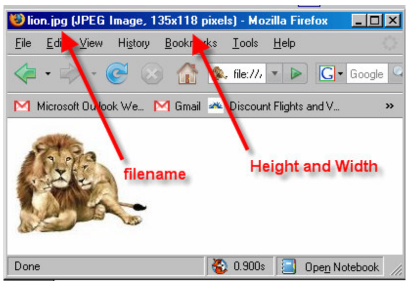
Your image files should all end with either .jpg or .gif. These are the two formats that browsers will read. If your files aren't in either of these forms, you'll need to run them through a graphics program like Photoshop to change the format.

1.4.3 Image Size Attributes

One of the most important image attributes is the size of your image expressed in height and width. You can easily find out this information by opening up the image in your browser.

From the browser, click on the **File** option from the top tool bar and highlight **Open File**. Then, choose your image file. In this example I will open the image of the lion from the above example.

The image will appear all by itself in the browser, and in the title bar, the name of the image, plus the dimensions will be shown.

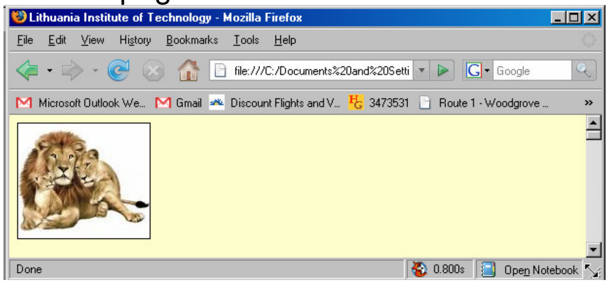


1.4.4 Image Alignment Attributes

Now that we have downloaded an image let's place the image into the firstpage.html file by using the following code. Place the code after the opening body tag <body>. Also, place a couple of <p> tags after the image to leave some space

```
<img alt="lion" border="1" src="lion.jpg"
width="135" height="118">

<p>

<p>
```

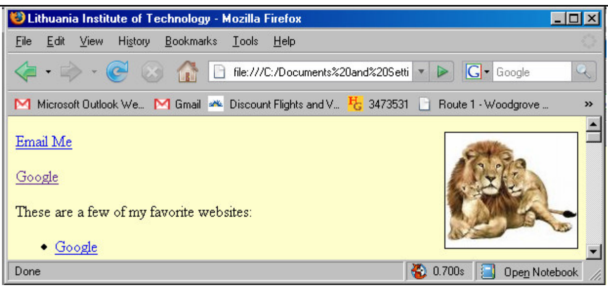
And the page will look like this:



You can add a right alignment attribute to your tag and get the lion placed over on the right side:

```
<img alt="lion" border="1" src="lion.jpg"
width="135" height="118" align="right">
```

And the page will look like this. Please note now that the image is aligned to the right, text that was underneath the image is placed beside it to the left.


Lithuania Institute of Technology - Mozilla Firefox
Email Me
Google
These are a few of my favorite websites:
Google

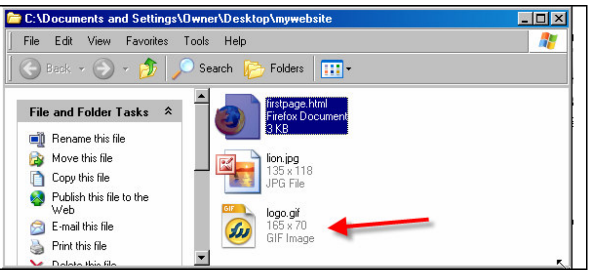
As you can see from the above example the align attribute is not very reliable, and it is a very difficult way of controlling the layout of your page. When you start making tables, you'll be shown a much more efficient method on how to align images on a webpage.

1.4.5 Linking Images

Images can also be used as links. All you have to do is surround the image tags with the anchor <a> tags.

For this demonstration let us look at the GIS-Centras logo and we will use that image as link to the website.

First let us go to the following website - http://www.gis-centras.lt/ - in the top left hand corner is the logo. Download the logo image to your **mywebsite** folder using the instructions from 1.4.1. Once completed you should have the logo image here:


C:\Documents and Settings\Owner\Desktop\mywebsite
File Edit View Favorites Tools Help
Back Search Folders
File and Folder Tasks
Rename this file
Move this file
Copy this file
Publish this file to the Web
E-mail this file
Print this file
firstpage.html Firefox Document 3 KB
lion.jpg 135 x 118 JPG File
logo.gif 165 x 70 GIF Image

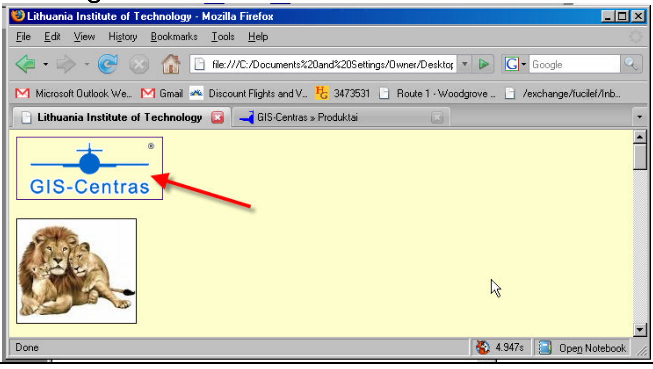
The GIS-Centras logo code is like this:

```
<img border="1" src="logo.gif" alt="GIS-
Centras Logo" width="165" height="70">
```

To make it a link, add the anchor <a> tags:

```
<a href="http://www.gis-centras.lt/"
target="_blank">

<img border="1" src="logo.gif" alt="GIS-
Centras Logo" width="165" height="70">

</a>
```

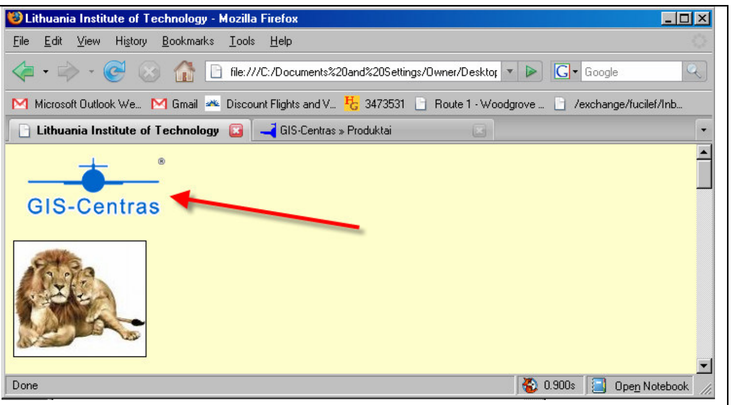
Now, the logo is a link back to the GIS-Centras homepage. Most linked images will have a border around them, put there automatically by the browser when it reads the <a> tags. If you have an image like the lion image below, you might not want to have that border. It loses that floating look.


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/Owner/Desktop
Lithuania Institute of Technology
GIS-Centras » Produktai
GIS-Centras
Done

If you would like to have the image appear to be floating then you must make the border attribute set to "0"

```
<a href="http://www.gis-centras.lt/"
target="_blank">
<img border="0" src="logo.gif" alt="GIS-
Centras Logo" width="165" height="70">

</a>
```

The lion image still has it's border set to "1". We will look at border images in more depth in the next sub-topic.


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/Owner/Desktop
Lithuania Institute of Technology
GIS-Centras » Produktai
GIS-Centras
Done

1.4.6 Image Borders

If you don't include a border attribute, most browsers will automatically put a border around images that are links. The border attribute is like all the other attributes you've seen - spaces before and after, but no spaces within the attribute, the equal sign, and the value. The value, as always, is enclosed in quotes.

Sometimes, you might want a border, just to give your image a more finished look. You can use the border attribute and a pixel-width thickness value to widen your image. The pictures below illustrate the thickness and thinness of image borders.

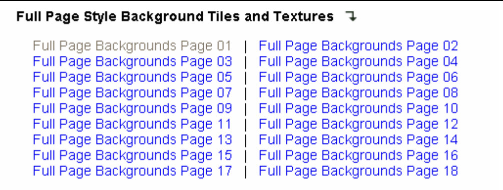
Border="0"	border="3	border="5	border="7

1.4.7 Image Backgrounds

Images can also be used as backgrounds. You can change the color of your background bgcolor attribute as shown here - **<body bgcolor="#ffffff">**

The code **<body bgcolor="#ffffff">** would create a blank solid color for your background. But, you can also take a small image and tell the browser to paste it across the screen until the entire screen is covered, like wallpaper.

You can download free background images from the following website - **http://www.boogiejack.com/free_graphics.html** - about half way down this webpage are a list of "full page" style background images that you can download.



Full Page Style Background Tiles and Textures

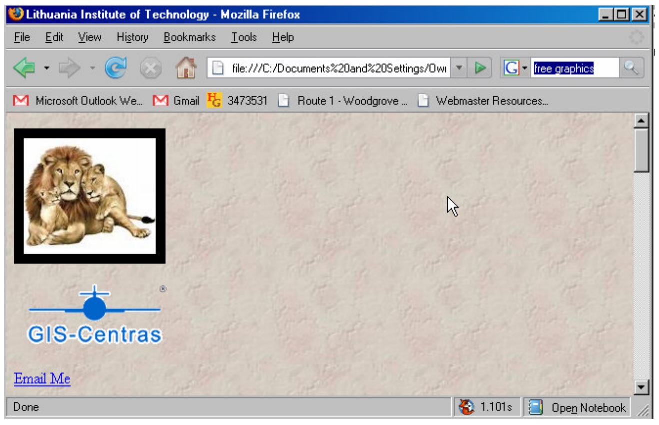
Full Page Backgrounds Page 01 | Full Page Backgrounds Page 02
Full Page Backgrounds Page 03 | Full Page Backgrounds Page 04
Full Page Backgrounds Page 05 | Full Page Backgrounds Page 06
Full Page Backgrounds Page 07 | Full Page Backgrounds Page 08
Full Page Backgrounds Page 09 | Full Page Backgrounds Page 10
Full Page Backgrounds Page 11 | Full Page Backgrounds Page 12
Full Page Backgrounds Page 13 | Full Page Backgrounds Page 14
Full Page Backgrounds Page 15 | Full Page Backgrounds Page 16
Full Page Backgrounds Page 17 | Full Page Backgrounds Page 18

Click on "**Full Page Backgrounds Page 01**" and download the second image from the top left hand corner - **tile_002.jpg** – into your **mywebsite** folder.

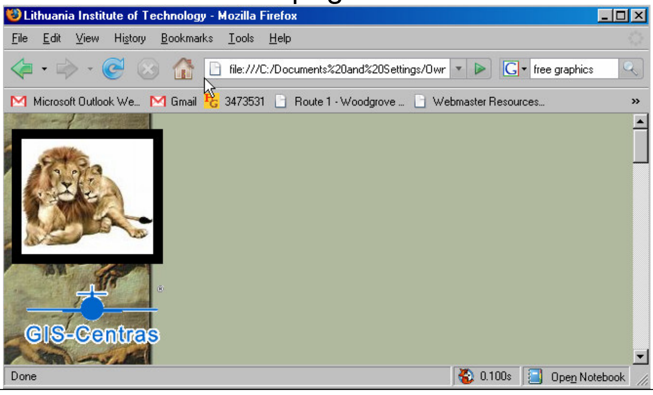
Now let's add this image to your firstpage.html document. Here is the code you will use.

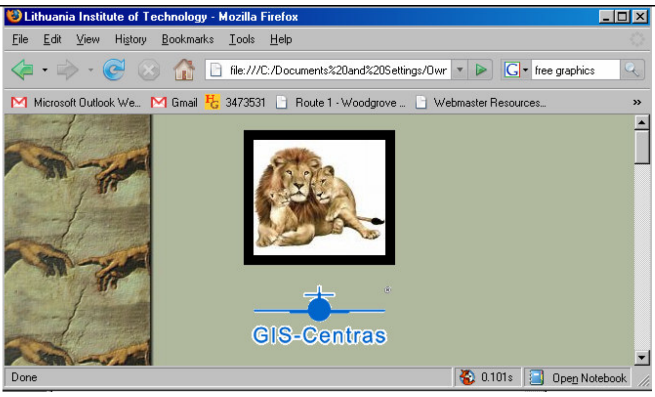
<body background=" tile_002.jpg">

And here's what webpage looks like now:


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%20and%20Settings/Own
free graphics
Microsoft Outlook We... Gmail 3473531 Route 1 - Woodgrove ... Webmaster Resources...
GIS-Centras
Email Me
Done
1.101s
Open Notebook

You have to be very careful with backgrounds. Unless you get the color and size of the text bold enough to stand out against the background, it could be very difficult to read the text on the background you have chosen. Another popular style of background tile is the "left border style." The left-side border will keep repeating all the way down your page. You will always need to keep your text and other images off the border.

Download a “left border”image from this website http://www.boogiejack.com/left_border_backgrounds_page02.html I will use the first background image as an example - bjack006.jpg **<body background=" bjack006.jpg ">**	And here's what webpage looks like now:  Lithuania Institute of Technology - Mozilla Firefox File Edit View History Bookmarks Tools Help file:///C:/Documents%20and%20Settings/Owr free graphics Microsoft Outlook We... Gmail 3473531 Route 1 - Woodgrove ... Webmaster Resources... GIS-Centras Done 0.100s Open Notebook

If you are going to use a “left hand” style border it is best to keep all your text and objects to the center of your webpage. You can simply do this by placing a center <center> tag right after the body tag like this: <body background=" bjack006.jpg "> <center>	 Lithuania Institute of Technology - Mozilla Firefox File Edit View History Bookmarks Tools Help file:///C:/Documents%20and%20Settings/Owr free graphics Microsoft Outlook We... Gmail 3473531 Route 1 - Woodgrove ... Webmaster Resources... GIS-Centras Done 0.101s Open Notebook

1.4.8 Images for Bulleted Lists

In the previous topic we learned how to create lists with a set of pre-defined bullet types. You can also use images as bullets by using the following code.

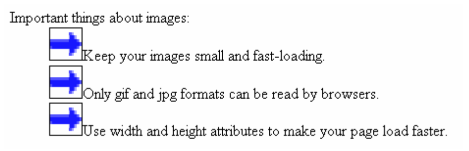
Firstly, let’s download a bullet image from here - http://www.boogiejack.com/bullet01.html

For this example I will use the “blue arrow” image - **arrow2.gif**

Here's the code for this example I have removed the background image. Here is a hint for you. If you would like your browser to ignore code you simply place an exclamation mark **!** in front of the tag or attribute. In this example the browser will not read the background image so the background should default to white

```
<body !background=" bjack006.jpg ">
<dl>
<dt>Important things about images:
<dd><img alt=" arrow " border="1" src="
arrow2.gif " width="32" height="32" />Keep your
images small and fast-loading.</dd>
<dd><img alt=" arrow " border="1" src="
arrow2.gif " width="32" height="32" />Only gif
and jpg formats can be read by browsers.</dd>
<dd><img alt=" arrow " border="1" src="
arrow2.gif " width="32" height="32" />Use width
and height attributes to make your page load
faster.</dd>
</dl>
```

And, here's what your webpage looks like:



Important things about images:

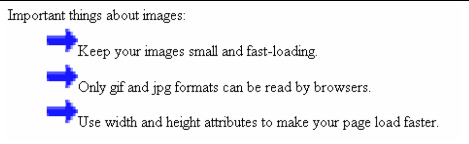
Keep your images small and fast-loading.

Only gif and jpg formats can be read by browsers.

Use width and height attributes to make your page load faster.

Now if you want to take it one step further you change the border attribute to “0” and the image bullets will appear to float on the page like this.

```
<dl>
<dt>Important things about images:
<dd><img alt="arrow" border="0" src="
arrow2.gif " width="32" height="32" />Keep your
images small and fast-loading.</dd>
<dd><img alt=" arrow " border="0" src="
arrow2.gif " width="32" height="32" />Only gif
and jpg formats can be read by browsers.</dd>
<dd><img alt=" arrow " border="0" src="
arrow2.gif " width="32" height="32" />Use width
and height attributes to make your page load
faster.</dd>
</dl>
```



Important things about images:

Keep your images small and fast-loading.

Only gif and jpg formats can be read by browsers.

Use width and height attributes to make your page load faster.

1.4.9 Creating Thumbnail Images

Thumbnail images are very small (viewing and file) size images i.e. they are resized images that are not only small physical size but small file size. Real thumbnails are **NOT** just images that look smaller because the are being resized with the width and height attributes of the img tag.

The benefit of a real thumbnail is that it loads a small size 'preview' images very fast. This will only happen if the file size is small. There is no real advantage using the image tag to make a thumbnail because the file size is exactly as the full image.

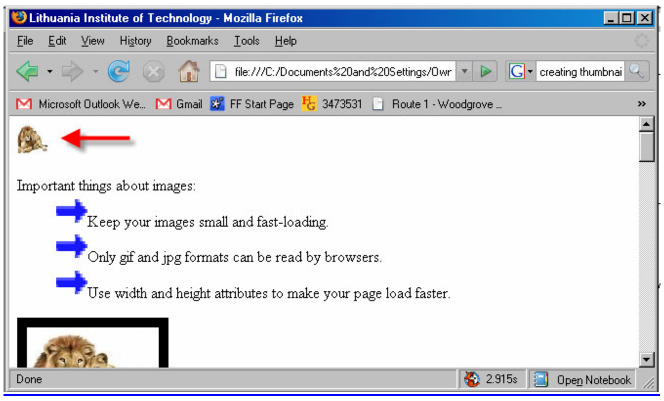
Many people prefer to use their favorite graphics program to make thumbnails. Making a thumbnail with a 'general' graphics program is simple:

Open the full size image in a graphics program (For example…Photoshop, Paint Shop Pro, Gimp, etc)

1. Then you will use the “Resize” function of the program. Make the 'Perserve aspect ratio' is set. A typical thumbnail is about 200 x 200 or smaller. And the file size is less than 3 - 10k (depending on the original resolution / optimizing)
2. Use Save As to save the thumbnail with a NEW name. Typically, you would append the name of the thumbnail image with **_tn**

As example here is a thumbnail image of the lion image. I resized the image using Paint Shop Pro X.

Normal Size = 135 X 118 Filename: lion.jpg	Thumbnail Size = 50 X 44 Filename: lion_tn.jpg

Here is the code to create a clickable Thumbnail image
 	
`<a href="http://www.gis-centras.lt/" target="_blank">`
 	
`<img alt="Lion Thumbnail" border="0" src=" lion_tn.jpg" width="32" height="32" />`
 	
`</a>`
	
	
	 Lithuania Institute of Technology - Mozilla Firefox
	File Edit View History Bookmarks Tools Help
	file:///C:/Documents%20and%20Settings/Owr
	creating thumbnai
	Microsoft Outlook We... Gmail FF Start Page 3473531 Route 1 - Woodgrove ...
	Done 2.915s Open Notebook
	Important things about images:
	Keep your images small and fast-loading.
	Only gif and jpg formats can be read by browsers.
	Use width and height attributes to make your page load faster.

1.4.10 Creating Imagemaps

An image map is just a kind of clickable image, which you can make to have different areas link to different websites. To make an image map, you need to add 2 groups of codes into your web page:

First is a map section, which contains area definitions, and the
Second is a special img tag, with an add-on to make it use the map.

When you create an image map as part of your document, you should always create text links to compliment the image links. This allows users of non-graphic browsers or those with the images turned off to fully access your page. It also acts as a back-up should your image not load up.

The descriptions below are way of an explanation on how the tags operate but in practical terms you will be using the ArcMap software to automatically generate your imagemaps for you.

Map Tag

This is where you assign a name to your image map, which you will need to use later in the **img** tag. The name is a required parameter of the map tag, and the map will not work without it. When used, a map tag will look like this and please note the map tag must be closed at the end	**<map name="Map of Lithuania">** Area Tags Go In Here... **</map>**

Area Tag

This is where you assign the individual clickable areas of your image. The area tag has 3 required parameters - the shape, the coordinates (cords), and the href parameter. If an area tag is made without all three, your image map will not work! Here is an example area tag:	<area shape="put shape type here" coords="put coordinates here" href="put URL here">

Shape and Coords:

There are **three** shapes that can be used for an image map - **circle**, **rect**, and **poly**. Each of the shapes requires a different number of numbers in the coords parameter. The order of the area tags is very important - the first area tag that mentions an area is the one which is used, so you should make sure you set your small areas first, and the large ones (which may wrap around small ones) last.	

Circle

For a **circle**, you must specify three numbers: The X and Y coordinates of the center of the circle, and the radius.	<area shape="circle" coords="284,131,60" href="link.htm">

Rectangle

rect makes a rectangular area clickable, and for it, you need 4 numbers - the X and Y coordinate of the top left corner, and the X and Y coordinate of the bottom right corner.	<area shape="rect" coords="91,148,194,262" href="link.htm">

Polygon

poly is the last type of area tag. It is used to create a polygon, or multi-sided shape, which the viewer can click on. For coords, it requires a list of X and Y coordinates of all of the points along the edge of the polygon. For complicated poly shapes, the coords parameter can be very long, and it becomes easy to make mistakes. It is usually best to keep your poly areas down to 8 points or less.	<area shape="poly" coords="203,213,203,387,325,336" href="link.htm">

href or nohref

This is the destination for your clickable area, which is specified by the href field, in exactly the same manner as the href parameter of the <a> tag. Optionally, if you wish to specify an area, which does not react to being clicked on, you can replace the href parameter with nohref.	An example of nohref: <area shape="default" nohref> Thus, any area not defined with – nohref - will not react to being clicked on.

img tag

The <img> tag used here is exactly the same as the normal one, with one addition - you must add a usemap parameter to it, which points to the map you created previously in your document. For example, an image map that uses a map named "navigation" would look like this:	<img src="navigation.gif" alt="Navigation" usemap="#navigation">

1.5. *Tables*

1.5.1 Creating a Simple Table

Tables were first designed to display tabular information. Later, they evolved into the main tool for designing a webpage, spreading the content out over the entire screen. Have you ever noticed how HTML code wants to jam everything up against the left margin? Tables are a way of getting margins on the page, that so-important 'white space' in designing.

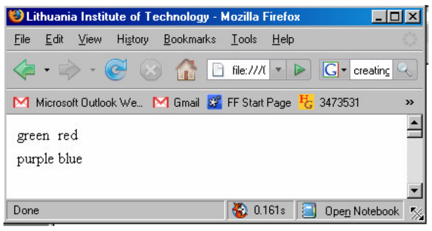
To make a table, you will use these basic tags:

<table>	opens the table.
<th>	opens and displays a table header.
</th>	closes the header line.
<tr>	opens a row.
<td>	opens a cell.
</td>	closes the cell.
</tr>	closes the row.
</table>	closes the table.

Here is an example of a simple table

```
<table>
<!First Row>
<tr>
<td>green</td>
<td>red</td>
</tr>

<!Second Row>
<tr>
<td>purple</td>
<td>blue</td>
</tr>

</table>
```

And, here's what the table would look like:


Lithuania Institute of Technology - Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///
creating
Microsoft Outlook We...
Gmail
FF Start Page
3473531
Done
0.161s
Open Notebook

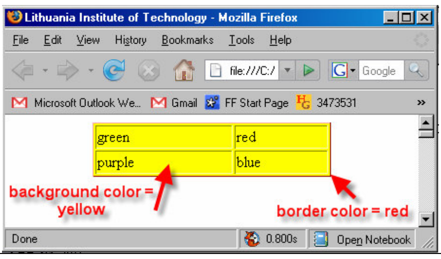
```
green  red
purple blue
```

1.5.2 Table Attributes

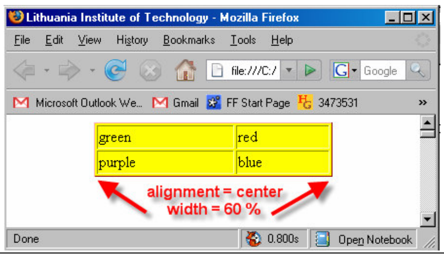
The simple table we created wasn't very exciting but let's see what we can do by adding attributes to the table:

```
<table width="60%" border="1" align="center" bordercolor="#FF0000" bgcolor="#FFFF00">

<tr><td>green</td><td>red</td></tr>

<tr><td>purple</td><td>blue</td></tr>

</table>
```

And, here's what the table would look likes now:



Lithuania Institute of Technology - Mozilla Firefox
green
red
purple
blue
background color = yellow
border color = red

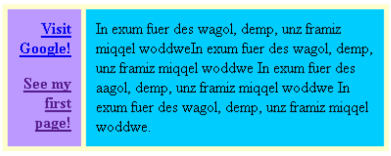


Depending on where you put the bgcolor attribute you are able to color the entire row, the entire table, or one cell at a time, Notice that you can put the attributes in any order within the tag, and they are always in this format:

- The attribute name with the equal sign (bgcolor=   align=   width=).
- The value, enclosed in quotation marks ("center" "#CC99FF"   "60%").
- There are no spaces between the attribute name and the final closing quotation mark.

1.5.3 Links in Tables

You can create navigation links in a column using tables, like this:

Explanation of Code	Table Code
The first line opens a table that will be centered, set to take up 80% of the screen (leaving 10% blank on each side), and will have a dark purple background color. The **cellpadding** attribute moves the contents 10 pixels from the border of the cell, and the **cellspacing** keeps the individual cells 5 spaces apart. Then, the row is opened. Within the first row, the first cell is opened, set to take up 20% of the width of the table. This cell will have a lavender background and the contents will be placed at the top of the cell, on the right side. Next comes a list of 2 active links, one to Google, one to your first page. All the links are in the first cell, placed one on top of the other. ● And then, the cell is closed. ● However, the row is not closed. Next, you'll open up the 2nd cell, set it to 80% of the width of the table, give it a light blue background, and align the contents to the top left corner. And then, there are is a paragraph of text and at the end of the paragraph, you will close the 2nd cell, then the row, and finally the table.	<table width=80% align="center" bgcolor="#ffffcc" cellpadding="10" cellspacing="5"> <tr> <td width="20%" align="right" valign="top" bgcolor="#BB99ff"> <p><b><a href="http://www.google.com/">Visit Google!</a></b></p> <p><b><a href="firstpage.html">See my first page!</a></b></p></td> <td width="80%" align="left" valign="top" bgcolor="#HHccff"> <p>In exum fuer des wagol, demp, unz framiz miqqel woddweIn exum fuer des wagol, demp, unz framiz miqqel woddwe In exum fuer des aagol, demp, unz framiz miqqel woddwe In exum fuer des wagol, demp, unz framiz miqqel woddwe.</p></td> </tr> </table> …and the table will look like this  Visit Google! See my first page! In exum fuer des wagol, demp, unz framiz miqqel woddweIn exum fuer des wagol, demp, unz framiz miqqel woddwe In exum fuer des aagol, demp, unz framiz miqqel woddwe In exum fuer des wagol, demp, unz framiz miqqel woddwe.

1.5.4 Cellpadding and Cellspacing

Cellpadding creates more white space between the cell edges and the contents.

Cellspacing increases the distance between cells. Values for both of these are given in pixels.

Both of these attributes are added to the <table> tag.

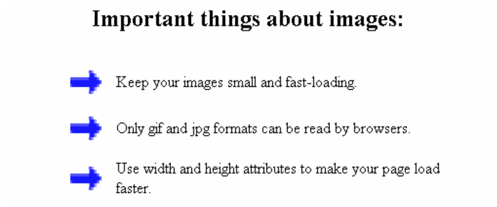
To see more examples of how cellpadding and cellspacing work visit the following websites:

Cellpadding - http://www.w3schools.com/html/tryit.asp?filename=tryhtml_table_cellpadding

Cellspacing - http://www.w3schools.com/html/tryit.asp?filename=tryhtml_table_cellspacing

To see another example of the use of tables let's re-format the Definition list we created in the previous topic into a table:

```
<table align="center" cellpadding="5" cellspacing="5" width="100%">

<tr><td colspan="2" align="center"><h2> Important things about images:</h2></td></tr>

<tr>
<td width="20%" align="right"><img alt="arrow" border="0" src="arrow2.gif"
width="32"height="32"></td>
<td width="80%">Keep your images small and fast-loading.</td></tr>

<tr>
<td align="right"><img alt="arrow" border="0" src="arrow2.gif" width="32"height="32"></td>
<td>Only gif and jpg formats can be read by browsers.</td>
</tr>

<tr>
<td align="right"><img alt="arrow" border="0" src="arrow2.gif" width="32"height="32"> </td>
<td> Use width and height attributes to make your page load faster. </td>
</tr>

</table>
```



Important things about images:

Keep your images small and fast-loading.

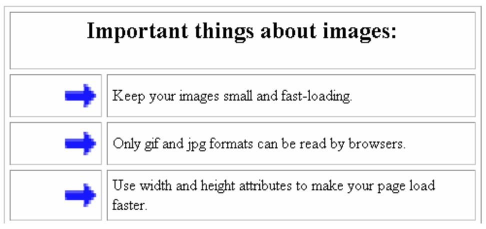
Only gif and jpg formats can be read by browsers.

Use width and height attributes to make your page load faster.

Now let's add the border attribute:

```
<table align="center" cellpadding="5" cellspacing="5" width="100%" border="1">
```

…and the table should look like this



Important things about images:

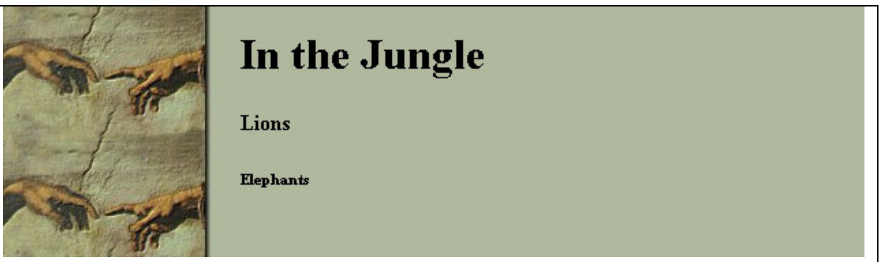
Keep your images small and fast-loading.

Only gif and jpg formats can be read by browsers.

Use width and height attributes to make your page load faster.

1.5.5 Using Color and Images in Tables

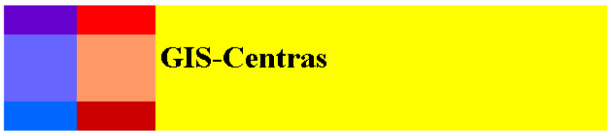
You can use tables to position your text onto the left handed background as shown below. The first cell is set for 25% and contains no text and the second cell is set for 75% and contains the text that shows up on the right hand side:

`<table width="100%"`
`cellpadding="5" cellspacing="5">`
`<tr>`
`<td width="25%">&nbsp;</td>`
`<td width="75%" align="left">`
`<h1>In the Jungle</h1>`
`<h4>Lionsh4>`
`<h6>Elephants</h6>`
`</td>`
`</tr>`
`</table>`
	 In the Jungle
	Lions
	Elephants

You can also use tables to create a graphic design for our webpages by adding color to each of the cells. For example,

```
<table width="100%" border="0"
cellpadding="5" cellspacing="0">

<tr>
<td bgcolor="#6600CC"
width="12%">&nbsp;</td>
<td bgcolor="#FF0000"
width="13%">&nbsp;</td>
<td bgcolor="#FFFF00"
width="75%">&nbsp;</td>
</tr>

<tr>
<td bgcolor="#6666FF"
width="12%">&nbsp;</td>
<td bgcolor="#FF9966"
width="13%">&nbsp;</td>
<td bgcolor="#FFFF00"
width="75%"><h1>GIS-Centras</h1></td>
</tr>

<tr>
<td bgcolor="#0066FF"
width="12%">&nbsp;</td>
<td bgcolor="#CC0000"
width="13%">&nbsp;</td>
<td bgcolor="#FFFF00"
width="75%">&nbsp;</td>
</tr>

</table>
```

This is what your webpage would look like:



1.6. Frames

1.6.1 Introduction to Frames

The use of frames, allows you to display more than one Web page in the same browser window. For example, you can a **vertical frameset** or a **horizontal frameset.**

With frames, you can display more than one HTML document in the same browser window. Each HTML document is called a frame, and each frame is independent of the others.

The disadvantages of using frames are:

- The web developer must keep track of more HTML documents
- It is difficult to print the entire page

For this topic we will be using examples from the W3schools website. This will better illustrate the various aspects of using Frames.

http://www.w3schools.com/html/html_frames.asp

1.6.2 Frame Tags

The Frameset Tag

The <frameset> tag defines how to divide the window into frames.Each frameset defines a set of rows or columns. The values of the rows/columns indicate the amount of screen area each row/column will occupy

The Frame Tag

The <frame> tag defines what HTML document to put into each frame

In the example below we have a frameset with two columns. The first column is set to 25% of the width of the browser window. The second column is set to 75% of the width of the browser window. The HTML document "frame_a.htm" is put into the first column, and the HTML document "frame_b.htm" is put into the second column:

```
<frameset cols="25%,75%">
  <frame src="frame_a.htm">
  <frame src="frame_b.htm">
</frameset>
```

1.6.3 Frame Attributes

If a frame has visible borders, the user can resize it by dragging the border. To prevent a user from doing this, you can add noresize="noresize" to the <frame> tag.

Add the <noframes> tag for browsers that do not support frames.

Please Note: You cannot use the <body></body> tags together with the <frameset></frameset> tags! However, if you add a <noframes> tag containing some text for browsers that do not support frames, you will have to enclose the text in <body></body> tags! See how it is done in the first example below.

How to use the <noframes> tag

This example demonstrates how to use the <noframes> tag.

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_noframes

Mixed frameset

This example demonstrates how to make a frameset with three documents, and how to mix them in rows and columns.

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_frame_mix

Frameset with noresize="noresize"

This example demonstrates the noresize attribute. The frames are not resizable. Move the mouse over the borders between the frames and notice that you can not move the borders.

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_frame_noresize

Navigation frame

This example demonstrates how to make a navigation frame. The navigation frame contains a list of links with the second frame as the target. The file called "tryhtml_contents.htm" contains three links. The source code of the links:

```
<a href ="frame_a.htm" target ="showframe">Frame a</a><br>
<a href ="frame_b.htm" target ="showframe">Frame b</a><br>
<a href ="frame_c.htm" target ="showframe">Frame c</a>
```

The second frame will show the linked document.

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_frame_navigation

Inline frame

This example demonstrates how to create an inline frame (a frame inside an HTML page).

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_iframe

Jump to a specified section within a frame

This example demonstrates two frames. One of the frames has a source to a specified section in a file. The specified section is made with <a name="C10"> in the "link.htm" file.

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_frame_jump

Jump to a specified section with frame navigation

This example demonstrates two frames. The navigation frame (content.htm) to the left contains a list of links with the second frame (link.htm) as a target. The second frame shows the linked document. One of the links in the navigation frame is linked to a specified section in the target file. The HTML code in the file "content.htm" looks like this: <a href ="link.htm" target ="showframe">Link without Anchor</a><br><a href ="link.htm#C10" target ="showframe">Link with Anchor</a>.

Frame Tags

Tag	Description
<frameset>	Defines a set of frames
<frame>	Defines a sub window (a frame)
<noframes>	Defines a noframe section for browsers that do not handle frames
<iframe>	Defines an inline sub window (frame)

http://www.w3schools.com/html/tryit.asp?filename=tryhtml_frame_navigation2

1.6.4 Vertical Frames

This example demonstrates how to make a vertical frameset with three different documents (http://www.w3schools.com/html/tryit.asp?filename=tryhtml_frame_cols).

1.6.5 Horizontal Frames

This example demonstrates how to make a horizontal frameset with three different documents (http://www.w3schools.com/html/tryit.asp?filename=tryhtml_frame_rows).

1.6.6 Navigation Frame

This example demonstrates how to make a navigation frame. The navigation frame contains a list of links with the second frame as the target. The file called "tryhtml_contents.htm" contains three links.

The source code of the links:
```
<a href ="frame_a.htm" target ="showframe">Frame a</a><br>
<a href ="frame_b.htm" target ="showframe">Frame b</a><br>
<a href ="frame_c.htm" target ="showframe">Frame c</a>
```

The second frame will show the linked document (http://www.w3schools.com/html/tryit.asp?filename=tryhtml_frame_navigation).

1.7. Divisions in HTML

The **<div>** tag defines logical divisions defined in your Web page. It acts a lot like a paragraph tag, but it divides the page up into larger sections.

<div> also gives you the chance to define the style of whole sections of HTML. You could define a section of your page as a call out and give that section a different style from the surrounding text.

The **<div>** tag gives you the ability to name certain sections of your documents so that you can affect them with style sheets or Dynamic HTML.

One thing to keep in mind when using the <div> tag is that it breaks paragraphs. It acts as a paragraph end/beginning, and while you can have paragraphs within a <div> you can't have a <div> inside a paragraph.

The primary attributes of the <div> tag are:
- style
- class

id

The <div> tag is **not** a replacement <p> tag. The <p> tag is for paragraphs, only, while the <div> tag defines more general divisions within a document. Don't replace <p> tags with <div> tags.

It's a good idea to label your <div> tags as you place them in your document. For example, if you're defining the main content area of your site, you should name that DIV tag: "maincontent". <div id="maincontent">

It's always a good idea to close your <div> tags as soon as you open them. Then place the contents within the element.

If you nest your <div> tags, be sure that you know where your content is going (in other words, which DIV it should be part of).

Style division

<div style="color : #ff0000">red text</div>

Justified text

<div align="justify">The following text will be justified across the width of the browser. This may not work reliably across all browsers, and can also be defined with style tags. Lorem ipsum sit dolor amet. Lorem ipsum sit dolor amet. Lorem ipsum sit dolor amet. Lorem ipsum sit dolor amet. Lorem ipsum sit dolor amet. Lorem ipsum sit dolor amet.</div>

- As this is a block-level tag, most browsers will add extra spaces before and after the element, similar to the paragraph element.
- This element is commonly used to layout Web pages with style sheets.

To investigate the further use of the <div> tag visit the following website:
http://www.w3schools.com/tags/tag_div.asp

1.8. CSS

1.8.1 Introduction to CSS

Cascading Style Sheets (CSS) give you the ability to simplify the creation of webpages. One major advantage to CSS is that it provides a method to maintain a consistent look and feel of your webpages. At the same time CSS also gives you much more control over the layout and design of your webpage. More control than HTML code can provide

Let us take a look at one of the more common styles applied to HTML - the color and size of text. In HTML, you would create a red <h1> text like this

<font color="#FF0000"><h1>Red Text</h1></font>	**Red Text**

However, if you wanted all your <h1> headings to be red, you would have to put those font tags before and after each <h1> tag.

This process could take a great deal of time and would provide you with many chances to make mistakes in the coding.

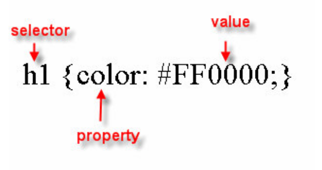
With CSS you can create a style sheet that will turn all <h1> headlines into the color red. That style sheet will look like this:

```
h1 {color: #FF0000;}
```

Now with the above style in place any <h1> tags that use that style sheet will automatically become red.

Let us look closer at the structure of the style sheet:

1. Select the element or tag that you want your style sheet to affect. For example, **h1**
2. Insert the opening curly bracket **{**
3. Name the property and the value, separated by a colon. For example, color: **#FF0000**
4. Insert the semi-colon **;**
5. Finally, close the style tag with a closing curly bracket **}**

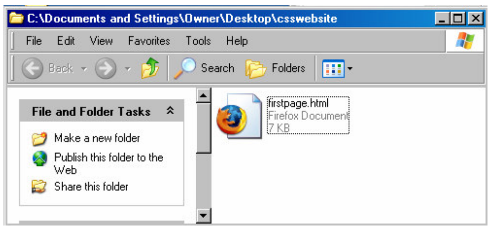
You can add as many property/value pairs as you want. Each pair needs to be followed by a semi-colon, and the two elements in each pair need to be separated by a colon. There are lots of details, but remember: you only have to write your style code once, and then all those <h1> tags will be taken care of.	

We can carry this further and add the **font-size** and font-weight attributes to the style as well.	h1 {color: #FF0000;
	font-size: 24px;
	font-weight: bold;
	}

1.8.2 First Webpage Using CSS

Before we begin to create our first css based webpage you firstly have to create a webpage that will be associated with the CSS file.

Create a new folder on your desktop and name it **csswebsite**. Place a copy of your **firstpage.html** file into the **csswebsite** folder.


C:\Documents and Settings\Owner\Desktop\csswebsite
File Edit View Favorites Tools Help
Back Search Folders
File and Folder Tasks
Make a new folder
Publish this folder to the Web
Share this folder
firstpage.html
Firefox Document
7 KB

Open the **firstpage.html** file and delete all the code in the file and replace it with this code:

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">

<html>

<head>

<title>My First CSS Webpage</title></head>

<body>

<h1>GIS-CENTRAS</h1>

<p>In exum fuer des wagol, demp, unz framiz miqqel woddweIn exum fuer des wagol, demp, unz framiz miqqel woddwe In exum fuer des aagol, demp, unz framiz miqqel woddwe In exum fuer des wagol, demp, unz framiz miqqel woddwe.</p>

<p>In exum fuer des wagol, demp, unz framiz miqqel woddweIn exum fuer des wagol, demp, unz framiz miqqel woddwe In exum fuer des aagol, demp, unz framiz miqqel woddwe In exum fuer des wagol, demp, unz framiz miqqel woddwe.</p>

```
</body>

</html>
```

Now, open a new document in your text editor – Notepad - and copy the code shown below.

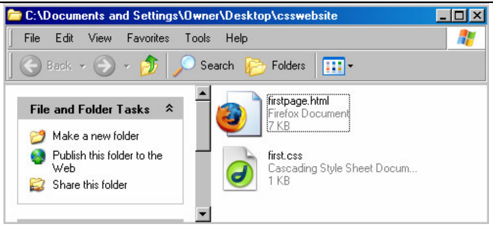
```
        body { background-color: #FFFFFF;
        margin-top: 2%;
        margin-right:10%;
        margin-bottom: 2%;
        margin-left: 10%; }

        h1 { font-family: Georgia, Times, serif;
        font-size: 24px;
        font-weight: bold;
        color: #000033; }
```

In the **body** style, the background color has been set to white - #FFFFFF - and margins have been set all the way around the page.

In the h1 style, the font type, size, weight and color has been set.

Save this page as **first.css**, and make sure that it's in your same folder, **csswebsite**.

You should now have two files in that folder, **first.css** and **firstpage.html. Please note that Windows has recognized the file type CSS and has labeled the file as a Cascading Style Sheet Document**



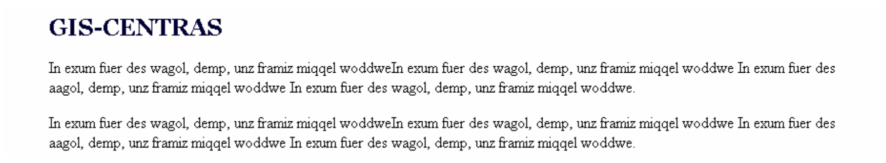
Now we have to link your **HTML** file to the **css** file before it can actually be used.

Open your **firstpage.html** file again and add the <link> tag between the opening and closing <head> </head> tags.

```
<head>
<link href="first.css" rel="stylesheet" type="text/css">
<title>My First CSS Webpage</title></head>
```

- The href attribute is the same in any <a> tag, containing the URL of your style sheet.
- The rel attribute specifies that the link is to a style sheet.
- The type attribute specifies the type of style sheet you're linking to.

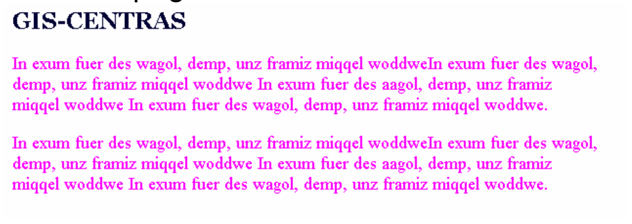
Save your **firstpage.html** file and try running it through a browser now and you should see this.



GIS-CENTRAS

In exum fuer des wagol, demp, unz framiz miqqel woddweIn exum fuer des wagol, demp, unz framiz miqqel woddwe In exum fuer des aagol, demp, unz framiz miqqel woddwe In exum fuer des wagol, demp, unz framiz miqqel woddwe.

In exum fuer des wagol, demp, unz framiz miqqel woddweIn exum fuer des wagol, demp, unz framiz miqqel woddwe In exum fuer des aagol, demp, unz framiz miqqel woddwe In exum fuer des wagol, demp, unz framiz miqqel woddwe.

Now let us make changes to the webpage by adding a style for the <p> tag. Add the following code to your **first.css** file
p { color: #FF00FF;
font-size: 18px;
}
	Your webpage should now look like this:
	
	GIS-CENTRAS
	In exum fuer des wagol, demp, unz framiz miqqel woddweIn exum fuer des wagol, demp, unz framiz miqqel woddwe In exum fuer des aagol, demp, unz framiz miqqel woddwe In exum fuer des wagol, demp, unz framiz miqqel woddwe.
	In exum fuer des wagol, demp, unz framiz miqqel woddweIn exum fuer des wagol, demp, unz framiz miqqel woddwe In exum fuer des aagol, demp, unz framiz miqqel woddwe In exum fuer des wagol, demp, unz framiz miqqel woddwe.

1.8.3 Navigation Lists

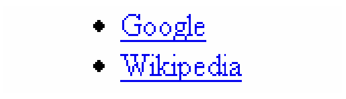
A navigation menu can really be looked at simply as a list of links. Whether your navigation menu is a horizontal row across the top or a vertical row down the side, it's still just a list. And, if you program your navigation using XHTML plus CSS, you can create a menu that is small to download (the XHTML) and easy to customize (the CSS).

To start designing a list for navigation, you need to have a list. All that is being used here is a standard unordered list where each list item has been linked with <a> tags.

Add the code below to **firstpage.html** page located in your **csswebsite** folder, place the code directly under the opening <body> tag.

```
<ul>
<li><a href=http://www.google.com/>Google</a></li>
<li><a href=http://www.wikipedia.org/>Wikipedia</a></li>
</ul>
```

Without any CSS styling, this menu does not look very exciting, but with just a few CSS styles added, you can create a menu to be proud of. Null or # link has been used so that these menu items will appear as links when you roll the mouse over the text, but they are not actually linked to any other pages.



- Google
- Wikipedia

1.8.4 Adding Divisions

Open your **first.css** style sheet and add the following style code to turn this into a website with a little more 'design'. You're going to divide your page into two divisions, one for the left-side navigation panel and one for the main body or center of your page.

Call your first division .navigation and position your cell starting from the absolute position (0,0 on an x-y axis with the starting point at the top left), give it a top margin of 10 pixels, a border along the right side that's a solid light green color, 1 pixel wide. The entire cell will take up 20% of the width of the screen.

In the second division .centerdoc, give the cell a padding. With all those numbers, list the number of pixels for the top, right, bottom and left margins, in that order.
top = 0 pixels right = 0 pixels bottom = 20 pixels left = 200 pixels

It's those 200 pixels that will make the browser skip over your first .navigation division and print the main body 200 pixels into the page.

```
.navigation

        {position: absolute;
        margin-top: 10px;
        border-right: 1px solid #C6EC8C;
        width: 15%;}

.centerdoc

        {padding: 0 0 20px 200px;
        margin-top: 10px;
        z-index: 20;}
```

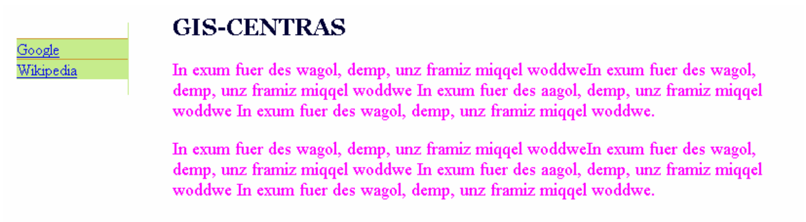
Add the above code to your own **first.css** document and be sure to **save the file**
Now open your **firstpage.html** file and add the following **divisions** as shown below

```
<div class="navigation">

<ul>
<li><a href=http://www.google.com/>Google</a></li>
<li><a href=http://www.wikipedia.org/>Wikipedia</a></li>
</ul>

</div>

<div class="centerdoc">

<h1>GIS-CENTRAS</h1>

<p>In exum fuer des wagol, demp, unz framiz miqqel woddweIn exum fuer des wagol, demp, unz framiz miqqel woddwe In exum fuer des aagol, demp, unz framiz miqqel woddwe In exum fuer des wagol, demp, unz framiz miqqel woddwe.</p>

<p>In exum fuer des wagol, demp, unz framiz miqqel woddweIn exum fuer des wagol, demp, unz framiz miqqel woddwe In exum fuer des aagol, demp, unz framiz miqqel woddwe In exum fuer des wagol, demp, unz framiz miqqel woddwe.</p>

</div>
```

Now let's add one more style to the style sheet place the following code in your **first.css** file

```
li { list-style-type: none;
display: block;
text-align: left;
background-color: #C6EC8C;
border-top: solid 1px #cc9933; }
```

Save your **first.css** file and open your **firstpage.html** in your browser and you should see the following webpage.



Google
Wikipedia

GIS-CENTRAS

In exum fuer des wagol, demp, unz framiz miqqel woddweIn exum fuer des wagol, demp, unz framiz miqqel woddwe In exum fuer des aagol, demp, unz framiz miqqel woddwe In exum fuer des wagol, demp, unz framiz miqqel woddwe.

In exum fuer des wagol, demp, unz framiz miqqel woddweIn exum fuer des wagol, demp, unz framiz miqqel woddwe In exum fuer des aagol, demp, unz framiz miqqel woddwe In exum fuer des wagol, demp, unz framiz miqqel woddwe.

To continue to experiment and learn more about Cascading Style Sheets please visit the following website. This website has more than 70 CSS examples and references.

http://www.w3schools.com/css/default.asp

1.9. Forms

A form is simply a webpage that can collect information from your readers and send that information to an email address. This information could be in the form of a survey, contact us form, an order form, or a customer feedback form from your customers.

1.9.1 Form Tags

Before you can set up your forms, you need to add certain lines of code to your webpage. These lines of code will process your form and send the results to your email address. For this example we will be using a fictitious website address. For the form to actually work you will need to get the necessary information (highlighted in bold below) from the webmaster of your website.

```
<body>

<form action="http://www.somecompany.com/cgi-bin/formmail.cgi" method="post">
<input type="hidden" name="emailto" value="anyone@somecompany.com">
<input type="hidden" name="nextpage" value="http://www.somecompany.com/thanks.html">
<input type="hidden" name="subject" value="Information Needed">
</form>

</body>
```

The first <form> tag opens up the form. At the end of your page, you'll have to remember to add a closing </form> tag.

The action attribute - **action="http://www.somecompany.com/cgi-bin/formmail.cgi" method="post"** - lets the browser know where to send the input of the form. In your case, you would want to send it to a **cgi** script that your webmaster has set up for you on your website. This cgi script will read the input from the <form> and format it. CGI stands for Common Gateway Interface.

The next line - **<input type="hidden" name="emailto" value="anyone@somecompany.com">** - will be hidden from the user, but will tell the browser to send the results to the email address that is placed in the **value** attribute

The next two lines are also inputs, and are also hidden:

```
<input type="hidden" name="nextpage" value="http://www.somecompany.com/thanks.html">
```

This inpute tells the browser that once the **Submit** button is clicked, the next page loaded should be at **http://www.somecompany.com/thanks.html**. This page tells the web visitor

```
<input type="hidden" name="subject" value="Information Needed">
```

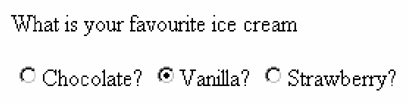
The next hidden input statement - **name="subject"** - tells the browser what you want in the subject line of your email message.

Now, you'll know that everything with the subject line **Information Needed** will be coming from this form.

Every <input> tag needs a
A type - the type could be – hidden, radio, checkbox, text, textarea, submit and/or reset
A name - the name is what will come back to you as a label in your email message.
A value - the value is the content supplied by the user filling out your form. In the case of radio buttons and checkboxes, your code will supply the value. In the case of text and textareas, the user will type in the values.

1.9.2 Radio Buttons

Each input in your form will require a type, a name, and a value. Here's a list of <input> tags for your radio buttons:

<p>What is your favorite ice cream</p> <p> <input type="radio" name="ice cream" value="Chocolate">Chocolate? <input type="radio" name="ice cream" value="Vanilla" checked="checked">Vanilla? <input type="radio" name="ice cream" value="Strawberry">Strawberry? </p>	And, here's what your webpage will look like when run through a browser:  What is your favourite ice cream Chocolate? Vanilla? Strawberry?

The checked="checked" attribute added in the list of attributes pre-selects that particular selection

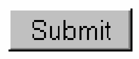
1.9.3 Submit Button

Once the form has been completed the website visitor must click the submit button. Here is the code for the submit button

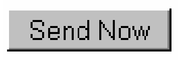
<input type="submit" value="Submit">

You don't need a name for this one. You just have to tell the browser that this is the signal to send off those values to the cgi-bin address in your <form> tag.

The value attribute is what will show up on the button when you run the document through a browser.



If you changed the value to something like this: <input type="submit" value="Send Now"> your button would look like this:



1.9.4 Checkboxes

Radio buttons are designed to give the user one choice, and only one choice. If they try to click on more than one radio button, the first one clicked will be emptied.

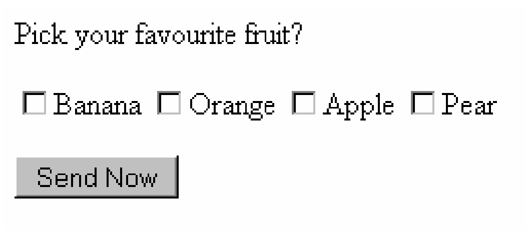
Checkboxes are designed to let your users click as many options.

Each <input> tag needs a type, a name, and a value attribute.

```
<p>Pick your favorite fruit?</p>
<p>
<input type="checkbox" name="fruit1" value="banana">Banana
<input type="checkbox" name="fruit2" value="orange">Orange
<input type="checkbox" name="fruit3" value="apple">Apple
<input type="checkbox" name="fruit4" value="pear">Pear
</p>
```

Notice that all the **names** for the checkboxes have to be different (unlike the radio buttons). This is because the radio buttons will return only one answer, while the checkboxes can return more than one.

And, here's what it will look like in a browser window:



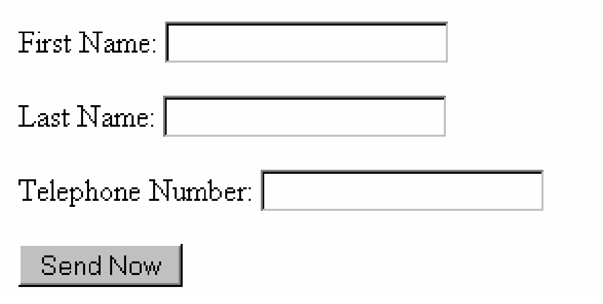
1.9.5 Input Tag

To gather text information you need places for web visitors to type in information. One of the methods used is the “input” tag.

Here's what the code will look like for a couple of text boxes that will collect name and phone number of the customers and the number of seed packets they would like to order:

```
<p>First Name: <input type="text" name="first_name"></p>

<p>Last Name: <input type="text" name="last_name"></p>

<p>Telephone Number: <input type="text" name="phone"></p>
```

The form should look like this:

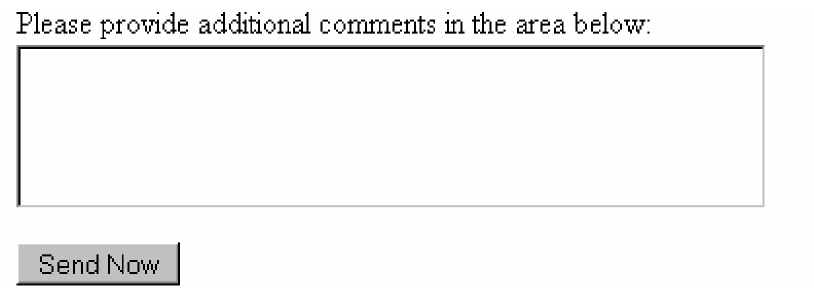


1.9.6 Text Area

Text areas are larger areas in the form where readers can type multiple lines of comments or questions. These text boxes don't use the <input> tags, and they need extra information assigned by the attritubes to set the number of rows and columns of the text area. In the line below, open a text area that will be 44 characters wide and 5 lines long.

```
<p>Please provide additional comments in the area below:
<textarea name="comments" rows="4" cols="45"></textarea>
</p>
```

Here's what it will look like:



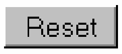
1.9.7 Reset Button

If one of your visitors fills out all the information, and then changes his mind, you can provide a Reset button. With one click, all the input in the form disappears. Here's the code:

```
<input type="reset" value="Reset">
```

Like the Submit button, you don't need a name for this one. You just have to tell the browser that this is the signal to clear out all the input values and start over.

The value attribute is what will show up on the button when you run the document through a browser. Like this:



If you changed the value to something like this:

```
<input type="reset" value="Try Again">
```

Your button would look like this:

Form Examples at W3 Schools Website

Here are a number of examples of forms from W3 Schools:

Form with input fields and a submit button
This example demonstrates how to add a form to a page. The form contains two input fields and a submit button: http://www.w3schools.com/html/tryit.asp?filename=tryhtml_form_submit

Form with checkboxes
This form contains two checkboxes, and a submit button:
http://www.w3schools.com/html/tryit.asp?filename=tryhtml_form_checkbox

Form with radio buttons
This form contains two radio buttons, and a submit button:
http://www.w3schools.com/html/tryit.asp?filename=tryhtml_form_radio

Send e-mail from a form
This example demonstrates how to send e-mail from a form:
http://www.w3schools.com/html/tryit.asp?filename=tryhtml_form_mail

Module self-study questions:

1. What are the 7 layers of the Internet and what are there functions?
2. Describe the building block of the Internet TCP/IP and explain how it works with FTP and HTTP?
3. Explain the purposes of HTML tags and HTML attributes?
4. Identify 3 reasons why you would use a list on your webpage and explain the difference between an ordered and unordered list?
5. What are your major considerations when downloading images from the Internet?
6. Describe the major purpose and usage of an imagemap?
7. What were tables originally designed for?
8. What are the 2 major disadvantages of using frames?
9. What is the major advantage of using CSS?
10. What can forms do for you on your webpage?

References

- http://www.wikipedia.org/
- http://www.w3.org
- http://www.w3schools.com/
- http://www.htmlcodetutorial.com/
- http://www.htmlgoodies.com/
- http://free-html-tutorials.com/
- http://webdesign.about.com/od/htmlxhtmltutorials/HTML_Tutorials_Web_Design_Tutorials_Beginning_HTML_and_Web_Design.htm
- http://www.mcli.dist.maricopa.edu/tut/lessons.html

Terms used

- TCP/IP
- FTP
- HTTP
- Telnet
- SMTP
- Bittorrent
- ARPANET
- OSI
- HTML
- Webpage
- Tags
- Attributes
- DTD
- W3C
- Browser
- Links
- Lists
- Target
- Anchor
- Relative
- Absolute
- Email
- Download
- Image
- Imagemap
- Table
- Cellpadding
- Cellspacing
- Frames
- Frameset
- Divisions
- CSS
- Forms

2. Web Languages for User Interface, Communication, Data and Schema Descriptions

Scripting languages and associated technologies, such as DHTML, HTML Document Object Models (DOMs), JavaScript XML, XSL and XSLT are very important, especially for applications to web creation development and applications. This module provides an introduction to these web application technologies, building on topics already considered in Module 1, and describing how recent developments in web application technology has facilitated an increased efficiency of internet technology. Methods discussed include the links between DHTML DOMs, and W3C standards, JavaScript methods, XML encoding, and methods of XSL and XSLT. The advantages and disadvantages of these techniques are also discussed.

2.1. DHTML, W3C Standards, HTML Document Object Model

2.1.1 Dynamic HTML or DHTML

Dynamic HTML or DHTML is a collection of methods for the creation of interactive and animated web sites. These allow changes in the appearance and content of web pages (hence such pages may be described as interactive or animated). DHTML uses several languages (for example HTML), a client-side scripting language (such as JavaScript), a presentation definition language (Cascading Style Sheets, CSS), and the Document Object Model. A DHTML webpage is one where scripting changes variables of the presentation definition language. This affects the look and function of the HTML page content. DHTML is often used to make rollover buttons or drop-down menus on a web page.

On the other hand, a dynamic web page is a broader concept - any web page generated differently for each user, load occurrence, or specific variable values. This includes pages created by client side scripting, and ones created by server-side scripting (such as PHP or Perl) where the web server generates content before sending it to the client.

Disadvantages of DHTML include:

- In some cases, scripts did not function effectively between different web browsers, so newer techniques (e.g., JavaScript coding and DOM Scripting) were used. These other techniques were more accessible through Progressive Enhancement (a strategy for web design that emphasizes accessibility and external stylesheet and scripting technologies).
- Difficulties in the development and debugging due to varying degrees of technological support among web browsers. Due to variable screen sizes, DHTML techniques may work on a limited number of browser and screen-size combinations. More recent browsers, such as Internet Explorer 5.0+, Mozilla Firefox 2.0+, and Opera 7.0+, usually combine DHTML with a shared Document Object Model.

Typically, a web page using DHTML is set up the following way:

```
<!DOCTYPE         html         PUBLIC         "-//W3C//DTD         XHTML         1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>DHTML example</title>
    <script type="text/javascript">
      function init() {
      myObj = document.getElementById("navigation");
      // .... more code
      }
      window.onload=init;
    </script>
  </head>
  <body>
    <div id="navigation"></div>
  </body>
</html>
<pre>
Often the code is stored in an external file; this is done by linking the file that
contains the JavaScript.
This is helpful when several pages use the same script:
</pre>
<script type="text/javascript" src="myjavascript.js"></script>
```

The following example will be displaying an additional block of text. The following code illustrates an often-used function. An additional part of a web page will only be displayed if the user requests it.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <title>Test</title>
    <style type="text/css">
      h2 {background-color: lightblue; width: 100%}
      a {font-size: larger; background-color: goldenrod}
      a:hover {background-color: gold}
      #example1 {display: none; margin: 3%; padding: 4%; background-color: limegreen}
    </style>
    <script type="text/javascript">
      function changeDisplayState (id) {
        d=document.getElementById("showhide");
        e=document.getElementById(id);
        if (e.style.display == 'none' || e.style.display == "") {
          e.style.display = 'block';
          d.innerHTML = 'Hide example';
        } else {
          e.style.display = 'none';
          d.innerHTML = 'Show example';
        }
      }
    </script>
  </head>
  <body>
    <h2>How to use a DOM function</h2>
    <div><a     id="showhide"     href="javascript:changeDisplayState('example1')">Show
example</a></div>
    <div id="example1">
      This is the example.
      (Additional information, which is only displayed on request)...
    </div>
    <div>The general text continues...</div>
  </body>
</html>
```

2.1.2 The World Wide Web Consortium (W3C)

The World Wide Web Consortium (***W3C***) is the main international standards organization for the World Wide Web (abbreviated WWW or W3). W3C was created to ensure compatibility and agreement among industry members in the adoption of new standards. The consortium was created to get all those vendors to agree on a set of core principles and components that would be supported by everyone, e.g., scripting. However, there is no W3C specification that formally defines DHTML. Most guidelines may be applicable to applications using DHTML, however the following guidelines focus on issues related to scripting and style sheets. W3C/IETF Standards (over Internet protocol suite) include the following: CSS, CGI, DOM, GRDDL, HTML , OWL, RDF, SVG, SOAP, SMIL, SRGS, SSML, VoiceXML , WSDL, XACML , XHTML, XML , XML Events, XForms, XML and XSLT. In 1998, W3C published the Level 1 DOM specification, which allowed access to and manipulation of every single element in an HTML page. All browsers have implemented this recommendation, and therefore, incompatibility problems in the DOM are now rare.

The W3C technologies that are important for this course are:

- Cascading Style Sheets (CSS), as was discussed in the first Module, is a stylesheet language used to describe the presentation of a document written in a markup language. Its most common application is to style web pages written in HTML and XHTML (http://en.wikipedia.org/wiki/XHTML), but the language can be applied to any kind of XML document, including SVG (http://en.wikipedia.org/wiki/SVG).
- The Common Gateway Interface (CGI) is a standard protocol for interfacing external application software with an information server, commonly a web server. The task of such an information server is to respond to requests (in the case of web servers, requests from client web browsers) by returning output. Each time a request is received, the server analyzes what the request asks for, and returns the appropriate output.
- The Document Object Model (DOM) is a platform- and language-independent standard object model for representing HTML or XML and related formats. A web browser is not obliged to use DOM in order to render an HTML document. However, the DOM is required by JavaScript scripts that wish to inspect or modify a web page dynamically. In other words, the Document Object Model is the way JavaScript sees its containing HTML page and browser state.
- Scalable Vector Graphics (SVG) is an XML specification and file format for describing two-dimensional vector graphics, both static and animated. SVG can be purely declarative or may include scripting. Images can contain hyperlinks using outbound simple XLinks.
- SOAP is a protocol for exchanging XML-based messages over computer networks, normally using HTTP/HTTPS. SOAP forms the foundation layer of the Web services stack, providing a basic messaging framework upon which abstract layers can be built.
- The Web Services Description Language (WSDL) is an XML-based language that provides a model for describing Web services. The WSDL defines services as collections of network endpoints, or ports. WSDL specification provides an XML format for documents for this purpose.
- The Extensible HyperText Markup Language, or XHTML, is a markup language that has the same depth of expression as HTML, but also conforms to XML syntax. Whereas HTML is an application of Standard Generalized Markup Language (SGML), a very flexible markup language, XHTML is an application of XML, a more restrictive subset of SGML. Because they need to be well-formed, true XHTML documents allow for automated processing to be performed using standard XML tools—unlike HTML, which requires a relatively complex, lenient, and generally custom parser. XHTML can be thought of as the intersection of HTML and XML in many respects, since it is a reformulation of HTML in XML.
- The Extensible Markup Language (XML) is a general-purpose markup language.[1] It is classified as an extensible language because it allows its users to define their own tags. Its primary purpose is to facilitate the sharing of structured data across different information systems, particularly via the Internet.

These the W3C technologies will be discussed in this and/or the following modules of this course.

2.1.3 Document Object Model (DOM)

The ***Document Object Model*** (DOM) is a platform- and language-independent standard object model for representing HTML or XML and related formats. The DOM is separated into different parts (Core, XML, and HTML) and different levels (DOM Level 1/2/3). The Document Object Model allows access to all elements in an HTML document, and also the methods and properties to add, move, change, or remove HTML elements. A web browser is not obliged to use DOM in order to render an HTML document. However, the DOM is required by JavaScript scripts that wish to inspect or modify a web page dynamically. In other words, the Document Object Model is the way JavaScript sees its containing HTML page and browser state. To change anything on a page, JavaScript needs access to all elements in the HTML document. With JavaScript you can restructure an entire HTML document. You can add, remove, change, or reorder items on a page. The DOM can also be used by JavaScript to read and change HTML, XHTML, and XML documents.

Because the DOM supports navigation in any direction (e.g., parent and previous sibling) and allows for arbitrary modifications, an implementation must at least buffer the document that has been read so far (or some parsed form of it). Hence, the DOM is likely to be best suited for applications where the document must be accessed repeatedly or out of sequence order. If the application is strictly sequential and one-pass, the SAX model (http://en.wikipedia.org/wiki/SAX) is likely to be faster and use less memory. In addition, non-extractive XML parsing models, such as VTD-XML (http://en.wikipedia.org/wiki/VTD-XML), provide a new memory-efficient option.

W3C began development of the DOM in the mid-1990s. Although the W3C never produced a specification for DOM 0, it was a partially documented model and was included in the specification of HTML 4. By October 1998, the first specification of DOM (DOM 1) was released. DOM 2 was issued in November 2000, with specifics on the style sheet object model and style information manipulation. DOM 3 was released in April 2004 and is the current release of the DOM specification.

The W3C DOM specifications are divided into levels, each of which contains required and optional modules. To claim to support a level, an application must implement all the requirements of the claimed level and the levels below it. An application may also support vendor-specific extensions that do not conflict with the W3C standards. As of 2005, Level 1, Level 2, and some modules of Level 3 are W3C Recommendations that means they have reached their final form.

Level 0

The application supports an intermediate DOM, which existed before the creation of DOM Level 1. Examples include the *DHTML Object Model* or the Netscape intermediate DOM. Level 0 is not a formal specification published by the W3C but rather a shorthand that refers to what existed before the standardization process. Level 0 DOM was invented by Netscape at the same time JavaScript was invented and was first implemented in Netscape 2. It offers access to a few HTML elements, forms and images. To ensure compatibility with established systems, the more recent, advanced browsers (including supporters of Level 1 DOM, also support the older Level 0 DOM, as not supporting it would prevent the function of most common scripts.

Level 1

Navigation of DOM (HTML and XML) document (tree structure) and content manipulation (includes adding elements). HTML-specific elements are included as well. This not only gives an exact model for the entire HTML (or XML) document, it also allowed more changes in the document such as changing paragraphs or tables.

Level 2

The Level 2 specification contains six different specifications:

- The DOM2 Core,
- Views,
- Events,
- Style,
- Traversal and Range,
- Level 2 HTML.

The Level 2 Core extends the functionality of the Level 1 Core. It also contains specialized interfaces for XML Level of support. The Level 2 Views allows programs and scripts to dynamically access and update the content of a representation of a document. The Level 2 Events gives a generic event system to programs and scripts. It introduces the concepts of event flow, capture, bubbling, and cancellation. The Level 2 CSS, or Level 2 Style, allows programs and scripts to dynamically access and update the content of style sheets. It has interfaces for Style Sheets and Cascading Style Sheets. The Level 2 Traversal and Range allow programs and scripts to dynamically traverse and identify a range of content in a document. The Level 2 Range allows the creation, insertion, modification, and deletion of a range of content in a Document. The Level 2 HTML allows programs and scripts to dynamically access and update the content and structure of HTML documents. It extends the interfaces defined in the Level 1 HTML, using the Level 2 Core possibilities.

Level 3 consists of 6 different specifications:

- DOM Level 3 Core;
- DOM Level 3 Load and Save;
- DOM Level 3 XPath;
- DOM Level 3 Views and Formatting;
- DOM Level 3 Requirements.
- DOM Level 3 Validation, which further enhances the DOM

The DOM3 Core extends the functionality of the DOM1 and DOM2 Core specifications, with new methods. The DOM3 Load and Save allows programs and scripts to dynamically load the content of an XML document into a DOM document, and serialize a DOM document into an XML document. The DOM3 XPath provides simple functionalities to access a DOM tree using XPath 1.0. The Level 3 Views and Formating mainly deal with the formatting and construction of pages, and the links of these pages with other pages and functions. The Level 3 requirements allow the links with other systems, to enable faster functions in the creation and use of web pages. The Level 3 Validation allows programs and scripts dynamically update the content and the structure of documents while ensuring that the document remains valid, or to ensure that the document becomes valid.

The DOM structure may be represented in a tree schema. This is based on the fact that in the DOM technology, everything in an HTML document is a node, the entire document is a document node, every HTML tag is an element node, the texts contained in the HTML elements are text nodes, every HTML attribute is an attribute node and comments are also comment nodes. The HTML text below illustrates the relationship between these nodes.

```
<html>
  <head>
    <title>DOM Tutorial</title>
  </head>
  <body>
    <h1>DOM Lesson one</h1>
    <p>Hello world!</p>
  </body>
</html>
```

In these text lines, all the nodes are related to the other nodes, and every node has a parent node (except for the document node). For example, the parent nodes of the <head> and <body> nodes are the <html> node, and the parent node of the "Hello world!" text node is the <p> node. The element nodes may also have child nodes. For example, the <head> node has one child node: the <title> node. The <title> node also has one child node: the text node "DOM Tutorial". In such cases, nodes may be considered siblings when they share a parent node. In this example, the <h1> and <p> nodes are siblings, as their common parent is the <body> node. In this sense, nodes can have descendants and ancestors. Descendants are all the nodes that are children of a node, or children of those children in continual sequence. Therefore, all text nodes may be descendants of the <html> node. The first text nodes are also descendants of the <head> node. Ancestors are those nodes that are parents of a node, or parents of this parent, in a continual sequence. For example, all text nodes will have the <html> node as an ancestor. Nodes also have a hierarchical relationship with each other. All nodes in an HTML document form a document tree (or node tree). The tree starts at the document node and continues to branch out until it has reached all text nodes at the lowest level of the tree. A tree schema for these nodes is shown below.

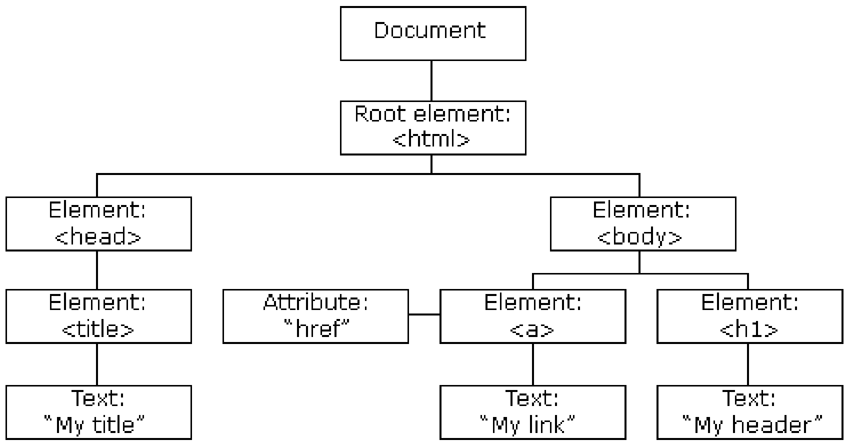


Figure: an HTML document form a document tree (or node tree). Source: http://www.w3schools.com/htmldom/dom_nodes.asp

2.2. Scripting Basics

2.2.1 Introduction to Scripting Languages

Scripting languages, also called script languages, are programming languages that control applications. They have many functions, one important one being the creation, operation maintenance and modification of web pages. Some of their main attributes are the following.

- They are executed directly from their source code (any sequence of statements and/or declarations written in some human-readable computer programming language), usually text files containing language specific markup, and are therefore different from programmers (markup is defined combine a text and extra information, such as symbols representing structure and layout).
- They are used to connect pre-existing components to accomplish a new related task. They are often used for creating graphical user interfaces (e.g. dynamic web pages) or executing a series of commands that might otherwise have to be entered interactively through keyboard at the command prompt.
- Some scripting languages allow end-users to write and debug short, simple, and sometimes domain-specific programs (such as web pages with changing formats).
- Another aim of the scripting language is to make it easy for a user with domain knowledge (an engineer, statistician, economist, etc.) to accomplish given tasks (e.g. creation of relevant web pages).
- Scripts are usually written as plain text and interpreted or compiled each time just before being invoked (both letters and symbols such as < or > may be used).
- Some scripting languages are designed for a specific domain or function, but often it is possible to write more general programs in that language (for example JavaScript may be used for several web applications).
- A scripting language and a lower level programming language may be used together, each lending its particular strengths to solve specific problems.
- Scripting languages may allow faster programming. Some scripting languages may be significantly slower to execute and consume more memory, but in most cases especially with small scripts, the write-time advantage far outweighs the run-time disadvantage.

There are many types of scripting languages. One important group of application-specific scripting languages is are used to develop web pages. These include JavaScript, JScript, VBScript Python PHP and Perl.

Some scripting languages are used for client side and others for server side development. Client side scripting is performed client-side because there may be a need for information or functionality that is available on the client but not on the server, because the user needs to observe them or provide input, or because the server lacks the processing power to perform the operations for the linked clients. In addition, if operations are done by clients, without sending data over the network, they may be faster, use less bandwidth, and have a lower security risk.

Server-side scripting is executed directly by the web server itself or by extension modules. A common scripts usually used for client side scripting are Javascript and VBScript, but both may also be used for server-side scripts. Perl, PHP, and server-side VBScript are used for server side scripting.

- **JavaScript** is a scripting language most often used for client-side web development. The primary use of JavaScript is to write functions that are embedded in or included from HTML pages and interact with the Document Object Model (DOM) of the page. JavaScript may be used for opening a new window with changes in the size, position and appearance of the window (i.e. whether the menus, toolbars, etc. are visible). It may also be used to validate the web form input values before submission to the server. JavaScript may also enable the changing of images as the mouse cursor moves over them, largely to draw the user's attention to important links displayed as graphical elements. As JavaScript code can run locally in a user's browser (rather than on a remote server), it responds fast to user actions. It can also detect user actions such as individual keystrokes (unlike HTML). JavaScript may enable applications such as ArcIMS, as most of the user-interface logic is written in JavaScript, which dispatches requests for information (such as the coordinates to identify a feature) to the server.

- **JScript** is an active scripting engine. It can be linked to any application that supports Windows Script, such as Internet Explorer, Active Server Pages (http://en.wikipedia.org/wiki/ASP.NET), and Windows Script Host. Any application supporting Windows Script can use multiple languages - JScript, VBScript, Perl, and others. JScript (and the other languages) can be used for both simple tasks (such as mouse movements on Web pages) and for more complex tasks (such as updating a database or running logon scripts for Windows NT).

- **VBScript** (short for Visual Basic Scripting Edition) is an Active Scripting language developed by Microsoft. A VBScript script must be executed within a host environment, of which there are several provided on a standard install of Microsoft Windows (Windows Script Host, Windows Internet Explorer). When employed in Microsoft Internet Explorer, VBScript is similar in function to JavaScript, as a language to write functions that are embedded in or included from HTML pages and interact with the Document Object Model (DOM) of the page, to perform tasks not possible in HTML alone. VBScript can also be used to create applications that run directly on a person's computer running Microsoft Windows. The simplest example of this is a script that makes use of the Windows Script Host (WSH) environment. Such a script is usually in a stand-alone file with the file extension .vbs. VBScript provides basic date/time, string manipulation, math, user interaction, error handling, and regular expressions.

- **Python** is a multi-paradigm programming language (primarily functional, object-oriented and imperative) which uses automatic memory management. A multi-paradigm programming language is a programming language that supports more than one programming paradigm. A programming paradigm is a fundamental style of programming regarding how solutions to problems are to be formulated in a programming language. Python was designed to be a highly readable language, with a simple visual layout, using English keywords where other languages use punctuation. Python requires less boilerplate (any text that is or can be reused in new contexts or applications without being changed much from the original) than traditional statically-typed structured languages such as C. It also has a smaller number of syntactic exceptions and special cases than either of these languages. Python uses indentation/whitespace, rather than curly braces or keywords, to delimit statement blocks. An increase in indentation comes after certain statements; a decrease in indentation signifies the end of the current block.

- **PHP** (Hypertext Preprocessor) is a computer programming language for producing dynamic web pages, used mainly in server-side scripting. It can also be used from a command line interface or in standalone graphical applications. PHP is a widely-used general-purpose scripting language that is suited for web development and can be embedded into HTML. PHP usually runs on a web server, taking PHP code as its input and creating web pages as output. However, it can also be used for command-line scripting and client-side GUI applications. PHP only parses code within its delimiters.

- **Perl** (Practical Extraction and Reporting Language) is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming and general user interface development. Its major features include support for multiple programming paradigms (procedural, object-oriented, and functional styles), built-in support for text processing, and a large collection of third-party modules. In contrast to Python, Perl emphasizes support for common application-oriented tasks, e.g. by having built-in regular expressions, file scanning and report generating features. It is therefore more application oriented and has a slightly narrower applicability. In contrast, Python emphasizes support for common programming methodologies such as data structure design and object-oriented programming.

2.2.2 HTML and the DOM

As it was discussed in the Module 1, HTML, is the predominant markup language for web pages. HTML is also often used to refer to content of the MIME type text/html or even more broadly as a generic term for HTML whether in its XML-descended form (such as XHTML 1.0 and later) or its form descended directly from SGML (such as HTML 4.01 and earlier). There are several HTML versions, the latest being HTML 4.01, published as a W3C Recommendation and ISO/IEC 15445:2000 are the most recent and final versions of HTML (as of January 2008).

HTML defines several data types for element content, such as script data and stylesheet data, and a plethora of types for attribute values, including IDs, names, URIs, numbers, units of length, languages, media descriptors, colors, character encodings, dates and times, and so on. All of these data types are specializations of character data. The Document Object Model (DOM) allows access to all elements in an HTML document, and also the methods and properties to add, move, change, or remove HTML elements.

HTML documents can start with a Document Type Declaration (informally, a "DOCTYPE"). The DTD to which the DOCTYPE refers contains machine-readable grammar specifying the permitted and prohibited content for a document conforming to such a DTD. Browsers do not necessarily read the DTD. The most popular graphical browsers use DOCTYPE declarations (or the lack thereof) and other data at the beginning of sources to determine which mode to use.

For example:

```
<!DOCTYPE          html          PUBLIC          "-//W3C//DTD          HTML          4.01//EN"
"http://www.w3.org/TR/html4/strict.dtd">
```

This declaration references the Strict DTD of HTML 4.01, which does not have presentational elements like <font>, leaving formatting to Cascading Style Sheets and the span and div tags. SGML-based validators read the DTD in order to properly parse the document and to perform validation. In modern browsers, the HTML 4.01 Strict doctype activates standards layout mode for CSS as opposed to quirks mode.

HTML 4.01 also provides Transitional and Frameset DTDs. The Transitional DTD was intended to gradually phase in the changes made in the Strict DTD, while the Frameset DTD was intended for those documents that contained frames. HTML documents can be delivered by the same means as any other computer file; however, HTML documents are most often delivered in one of the following two forms: over HTTP servers and through email.

The World Wide Web is primarily composed of HTML documents transmitted from a web server to a web browser using the Hypertext Transfer Protocol (HTTP).However, HTTP can be used to serve images, sound and other content in addition to HTML. To allow the web browser to know how to handle the document it received, an indication of the file format of the document must be transmitted along with the HTML document.

2.2.3 JavaScript Basics

This section gives a description of JavaScript. The following subsections are included.

- Description of JavaScript
- JavaScript and Web Browsers
- JavaScript Statements
- JavaScript Comments
- JavaScript Variables
- Basic Data Types in JavaScript
- Operators
- Statements
- Functions
- JavaScript Events

After this section, two sections on Javascript object hierarchy, and samples of JavaScript objects for JavaScript Image maps will be also considered,

Description of JavaScript: JavaScript is a scripting language most often used for client-side web development, consisting of lines of executable computer code. The language is best known for its use in websites (as client-side JavaScript), but is also used to enable scripting access to objects embedded in other applications. As of 2006, the latest version of the language is JavaScript 1.7.

Because JavaScript runs on the client rather than the server, it can respond to user actions quickly, making an application feel more responsive. Furthermore, JavaScript code can detect user actions that HTML alone cannot, such as individual keystrokes. Applications such as ArcIMS attempt to take advantage of this: much of the user-interface logic is written in JavaScript, and JavaScript dispatches requests for information (such as the coordinates for an identification of feature) to the server.

JavaScript and Web Browsers: JavaScript is used, among other things, to improve the design, validate forms, detect browsers, create cookies, and much more. It works with all the major web browsers, e.g. Internet Explorer, Mozilla, Firefox, Netscape and Opera. JavaScript was designed to add interactivity to HTML pages, and to write functions that are embedded in or included from HTML pages and interact with the DOM of the page. Some simple examples of this usage are:

- Opening or popping up a new window with programmatic control over the size, position and 'look' of the new window (i.e. whether the menus, toolbars, etc. are visible).
- Validation of web form input values to make sure that they will be accepted before they are submitted to the server.
- Changing images as the mouse cursor, to draw the user's attention to important links displayed as graphical elements.

JavaScript gives HTML designers a programming tool, by putting dynamic text into an HTML page. It can react to events: for example, set to execute after an action, such as when a page is loaded or a user clicks on an HTML element. It can also read, write and hence change HTML. It can be used to validate form data before submission to a server, saving the server from processing. It can detect a visitor's browser and load pages designed the specific browser. JavaScript can also be used to create cookies and to store and retrieve information on the visitor's computer

The most common host environment for JavaScript is a web browser. Web browsers usually use the public API to create "host objects" responsible for reflecting the DOM into JavaScript. Another host environment is the web server. The JavaScript web server exposes host objects that represent HTTP requests and response objects, which a JavaScript program could use to create web pages.

A simple example of a Web page containing JavaScript (using HTML syntax) would be:

```
<!DOCTYPE html>
<html>
  <head><title>simple page</title></head>
  <body>
    <script>
      document.write('Hello World!');
    </script>
    <noscript>
      Your browser does not support JavaScript.
    </noscript>
  </body>
</html>
```

In terms of where to put the JavaScript, the JavaScripts in a page are executed as the page loads into the browser. In some cases, the user will execute a script when a page loads, or when user triggers an event. When scripts are to be executed when they are called, or after the triggering of an event, they go to the head section. This ensures that the script is loaded before it is used. Where scripts are to be executed when the page loads, they are located in the body section, this generating the content of the page. Scripts may also be in both the body and the head sections, where such functions are combined.

The syntax of JavaScript is a set of rules that defines what constitutes a valid program in the JavaScript language. Variables have no type attached, and any value can be stored in any variable. Variables can be declared with a var statement. These variables are lexically scoped and once a variable is declared, it may be accessed anywhere inside the function where it is declared. Variables declared outside any function, and variables first used within functions without being declared with 'var', are global. Here is an example of variable declarations and global values:

```
x = 0; // A global variable
var y = 'Hello!'; // Another global variable

function f(){
  var z = 'foxes'; // A local variable
  twenty = 20; // Global because keyword var is not used
  return x; // We can use x here because it is global
}
// The value of z is no longer available
```

JavaScript is case sensitive. It is common to start object names with a capitalized letter and functions or variables with a lower-case letter. Spaces, tabs and newlines used outside of string constants are called whitespace. Unlike C, whitespace in JavaScript source can directly impact semantics. Because of a technique called "semicolon insertion", any statement that is well formed when a newline is parsed will be considered complete (as if a semicolon were inserted just prior to the newline). Programmers are advised to supply statement-terminating semicolons explicitly to enhance readability and lessen unintended effects of the automatic semicolon insertion.

Unnecessary whitespace, whitespace characters that are not needed for correct syntax, can increase the amount of wasted space, and therefore the file size of .js files. The easiest way to address the problem of file size is to set the server to use zip compression. This compression will work far better than any whitespace parser and will reduce the size of all other source your server uploads. This method will work with or without semi-colons.

JavaScript Statements: JavaScript statements are commands to the browser, telling it what to do, for example in the following format, telling the browser to write “Hi there”:

```
document.write("Hi There");
```

Usually, a semi colon is written at the end of the sentence, but this is optional as the browser interprets the end of the line as the end of the statement.

JavaScript Comments: Comment syntax of JavaScript is the same as in C++, as below.

```
// single line

/* multi-line
   comment */
```

Comments can be used to make the code more readable, and also to explain the JavaScript. Single line comments start with //. For example:

```
<script type="text/javascript">
// This will write a header:
document.write("<h1>This is a header</h1>");
// This will write two paragraphs:
document.write("<p>This is a paragraph</p>");
document.write("<p>This is another paragraph</p>");
</script>
```

Multi line comments start with /* and end with */. For example:

```
<script type="text/javascript">
/*
The code below will write one header and two paragraphs
*/
document.write("<h1>This is a header</h1>");
document.write("<p>This is a paragraph</p>");
document.write("<p>This is another paragraph</p>");
</script>
```

Comments may also be used to prevent execution of a single or multiple code lines. Comments may also be placed at the end of the line

JavaScript Variables: These may described as "containers" for storing information, for example:

```
x = 8;  length =79.40;  truckname = "Ford"
```

The value of a variable can be changed during the script. The user may refer to the variable by name to read the value or change the value. Variable names are case sensitive and must begin with a letter or the underscore character. Variables are used to store data. For example:

```
<html>
<body>

<script type="text/javascript">
var name="Hege";
document.write(name);
document.write("<br>");
name="Tove";
document.write(name);
</script>

<p>The script above declares a variable, assigns a value to it, change the value,and
displays the value again.</p>
</body>
</html>
```

A variable may be created with the statement, for example:

```
var strname = some value
```

Assigning a Value to a Variable may be done in several ways, for example below the variable strname has the value "Hege":

```
var strname="Hege"; //or
strname="Hege";
```

Basic Data Types in JavaScript:

Numbers: Numbers in JavaScript are represented in binary as IEEE-754 Doubles, which provides an accuracy to about 14 or 15 significant digits JavaScript. Because they are binary numbers, they do not always exactly represent decimal numbers, particularly fractions. This becomes an issue when formatting numbers for output, which JavaScript has no built-in methods for. For example:

```
alert(0.94 - 0.01); // displays 0.9299999999999999
```

As a result, rounding should be used whenever numbers are formatted for output.

Numbers may be specified in any of these notations:

```
345;    // an "integer", although there is only one numeric type in JavaScript
34.5;   // a floating-point number
3.45e2; // another floating-point, equivalent to 345
0377;   // an [[octal]] integer equal to 255
0xFF;   // a [[hexadecimal]] integer equal to 255, the letters A-F may be upper- or
lowercase
```

When used as a constructor, a numeric wrapper object is created (see the description for object's type below):

```
myNumericWrapper = new Number( 123.456 );
```

Arrays: An array is a data structure consisting of a group of elements accessed by indexing. Each element may have the same data type and the array occupies a contiguous area of storage. Most programming languages have a built-in array data type. In JavaScript, arrays have a length property that is larger than the largest integer index used in the array. It is automatically updated if one creates a property with an even larger index. Writing a smaller number to the length property will remove larger indices. This length property is the only special feature of Arrays that distinguishes it from other objects.

Elements of Arrays may be accessed using normal object property access notation:

```
myArray[1];
myArray["1"];
```

The above two are equivalent. It is not possible to use the "dot"-notation or strings with alternative representations of the number:

```
myArray.1;     // syntax error
myArray["01"]; // not the same as myArray[1]
```

Declaration of an array can use either an Array literal or the Array constructor:

```
myArray = [0,1,,,4,5];             // array with length 6 and 4 elements
myArray = new Array(0,1,2,3,4,5); // array with length 6 and 6 elements
myArray = new Array(365);          // an empty array with length 365
```

Arrays are implemented so that only the elements defined use memory; they are "sparse arrays". Setting myArray[10] = 'someThing' and myArray[57] = 'somethingOther' only uses space for these two elements, just like any other object. The length of the array will still be reported as 58.

It can be used the object declaration literal to create objects that behave much like associative arrays in other languages:

```
dog = {"color":"brown", "size":"large"};
dog["color"]; // this gives you "brown"
```

It can be used the object and array declaration literals to quickly create arrays that are associative, multidimensional, or both.

```
cats = [{"color":"brown", "size":"large"},
        {"color":"black", "size":"small"}];
cats[0]["size"]; // this gives you "large"

dogs = {"rover":{"color":"brown", "size":"large"},
        "spot":{"color":"black", "size":"small"}};
dogs["spot"]["size"]; // this gives you "small"
```

Strings: Strings in Javascript are a sequence of characters. Strings can be created directly by placing the series of characters between double or single quotes, for example:

```
var greeting = "Hello, world!";
var another_greeting = 'Greetings, people of Earth.';
```

In Mozilla based browsers, individual characters within a string can be accessed (as strings with only a single character) through the same notation as arrays:

```
var h = greeting[0]; // Now h contains 'H' - Works in Mozilla based browsers
```

But, for Internet Explorer, the individual characters can be accessed using the charAt() method (provided by String class). This is the preferred way when accessing individual characters within a string, as it also works in Mozilla based browsers:

```
var h = greeting.charAt(0); // Now h contains 'H' - Works in both Internet Explorer
                            // and Mozilla based browsers
```

However, JavaScript strings are immutable; the meaning that they cannot be modified after it is created, and the result may be an error message if this attempted:

```
greeting[0] = "H"; // ERROR
```

Applying the equality operator ("==") to two strings returns true if the strings have the same contents, which means: of same length and same cases (for alphabets). Thus:

```
var x = "world";
var compare1 = ("Hello, " + x == "Hello, world"); // Now compare1 contains true
var compare2 = ("Hello, " + x == "hello, world"); // Now compare2 contains false since
the first characters of both operands are not of the same case
```

Objects: Types are normally subdivided into primitives and objects. Objects are entities that have an identity (they are only equal to themselves) and that map property names to values, ("slots" in prototype-based programming terminology). JavaScript objects are often mistakenly described as associative arrays or hashes, but they are neither.

JavaScript has several kinds of built-in objects, namely Array, Boolean, Date, Function, Math, Number, Object, RegExp and String. Other objects are "host objects", defined not by the language but by the runtime environment. For example, in a browser, typical host objects belong to the DOM (window, form, links etc.).
Objects can be created using a declaration, an initializer or a constructor function:

```
// Declaration
var anObject = new Object();

// Initialiser
var objectA = {};
var objectB = {'index1':'value 1','index2':'value 2'};

// Constructor (see below)
```

The most basic objects in JavaScript act as dictionaries. These dictionaries can have any type of value paired with a key, which is a string. Objects with values can be created directly through object literal notation:

```
var o = {name: 'My Object', purpose: 'This object is utterly without purpose.', answer:
42};
```

Properties of objects can be created, set, and read individually using the familiar dot ('.') notation or by a similar syntax to arrays:

```
var name = o.name; // name now contains 'My Object'
var answer = o['answer']; // answer now contains 42
```

Object literals and array literals allow one to create flexible data structures:

```
var myStructure = {
  name: {
    first: "Mel",
    last: "Smith"
  },
  age: 33,
  hobbies: [ "chess", "jogging" ]
};
```

This is the basis for JSON (JavaScript Object Notation, http://en.wikipedia.org/wiki/JSON), which is a simple notation that uses JavaScript-like syntax for data exchange.

JavaScript Operators:

Arithmetic binary operators are:

```
+     Addition
-     Subtraction
*     Multiplication
/     Division (returns a floating-point value)
%     Modulus (returns the integer remainder)
```

The + operator is overloaded, that means it is used for string concatenation and arithmetic addition and also to convert strings to numbers. It also has special meaning when used in a regular expression. For example:

```
// Concatenate 2 strings
var a = 'This';
var b = ' and that';
alert(a + b);  // displays 'This and that'

// Add two numbers
var x = 2;
var y = 6;
alert(x + y); // displays 8

// Adding a string and a number results in concatenation
alert( x + '2'); // displays 22

// Convert a string to a number
var z = '4';   // z is a string (the digit 4)
alert( z + x); // displays 42
alert( +z + x);// displays 6
```

Arithmetic unary operators are:

```
-     Unary negation (reverses the sign)
++    Increment (can be prefix or postfix)
--    Decrement (can be prefix or postfix)
```

JavaScript Assignment operators are:

```
=     Assign
+=    Add and assign
-=    Subtract and assign
*=    Multiply and assign
/=    Divide and assign

var x = 1;
x *= 3;
document.write( x );  // displays: 3
x /= 3;
document.write( x );  // displays: 1
x -= 1;
document.write( x );  // displays: 0
```

JavaScript Comparison operators are:

```
==    Equal
!=    Not equal
>     Greater than
>=    Greater than or equal to
<     Less than
<=    Less than or equal to

===   Identical (equal and of the same type)
!==   Not identical
```

JavaScript ***boolean comparison*** operators: JavaScript has three logical boolean operators: && (logical AND), || (logical OR), and ! (logical NOT):

```
&&    and
||    or
!     not (logical negation)
```

In the context of a boolean operation, all JavaScript values evaluate to true unless the value is the boolean false itself, the number 0, a string of length 0, or one of the special values null, undefined, or NaN. The Boolean function can be used to explicitly perform this conversion:

```
Boolean( false );     // returns false
Boolean( 0 );         // returns false
Boolean( 0.0 );       // returns false
Boolean( "" );        // returns false
Boolean( null );      // returns false
Boolean( undefined ); // returns false
Boolean( NaN );       // returns false
```

The unary NOT operator ! first evaluates its operand in a boolean context, and then returns the opposite boolean value:

```
var a = 0;
var b = 9;
!a; // evaluates to true,  same as (Boolean( a ) == false)
!b; // evaluates to false, same as (Boolean( b ) == true)
```

In the earliest implementations of JavaScript and JScript, the && and || operators behaved in the same manner as their counterparts in other C derived programming languages, in that they always returned a boolean value:

```
x && y; // returns true if x AND y evaluate to true: (Boolean( x ) == Boolean( y ) ==
true), false otherwise
x || y; // returns true if x OR y evaluates to true, false otherwise
```

In the newer implementations, these operators return one of their operands:

```
expr1 && expr2; // returns expr1 if it evaluates to false, otherwise it returns expr2
expr1 || expr2; // returns expr1 if it evaluates to true, otherwise it returns expr2
```

This novel behavior is little known even among experienced JavaScripters, and can cause problems if one expects an actual Boolean value.

Short-circuit logical operations means the expression will be evaluated from left to right until the answer can be determined. For example, a || b is automatically true if a is true and a && b is false if a is false. There is no reason to evaluate b.

Bitwise *binary* operators are:

```
&      And
|      Or
^      Xor

<<     Shift left  (zero fill)
>>     Shift right (sign-propagating); copies of the leftmost bit (sign bit) are shifted
in from the
       left.
>>>    Shift right (zero fill)

       For positive numbers, >> and >>> yield the same result.
```

Bitwise unary operator is:

```
~      Not (inverts the bits)
```

String operators are:

```
=      Assignment
+      Concatenation
+=     Concatenate and assign
```

Examples are:

```
str = "ab" + "cd";    // "abcd"
str += "e";           // "abcde"
```

JavaScript Statements:

Conditional statements: In Javascript, these allow different actions for different decisions that can be accomplished with the following statements:

- `if` statement allows the execution of some code, only if a specified condition is true
- `if...else` statement allows the execution of some code if the condition is true and another code if the condition is false
- `if...else if....else` statement allows the selection of one of many blocks of code to be executed

Examples of the statements are:

```
<script type="text/javascript">
//Write a "Good morning" greeting if the time is less than 10var d=new Date();
var time=d.getHours();
if (time<10)
{
document.write("<b>Good morning</b>");
}
</script>
```

When comparing variables two equals signs next to each other must be used (==).

If...else statement syntax is:

```
if (condition)
{
code to be executed if condition is true
}
else
{
code to be executed if condition is not true
}
```

Example of If...else statement is:

```
<script type="text/javascript">
//If the time is less than 10, you will get a "Good morning" greeting.
//Otherwise you will get a "Good day" greeting.var d = new Date();

var time = d.getHours();
if (time < 10)
{
document.write("Good morning!");
}
else
{
document.write("Good day!");
}
</script>
```

`If...else if...else` statement syntax is:

```
if (condition1)
{
code to be executed if condition1 is true
}
else if (condition2)
{
code to be executed if condition2 is true
}
else
{ code to be executed if condition1 and condition2 are not true
}
```

Example of `If...else if...else` statement is:

```
<script type="text/javascript">
var d = new Date()
var time = d.getHours()
if (time<10)
{
document.write("<b>Good morning</b>");
}
else if (time>10 && time<16)
{
document.write("<b>Good day</b>");
}
else
{
document.write("<b>Hello World!</b>");
}
</script>
```

Switch statement allows the selection of one of many blocks of code to be executed. The syntax is:

```
switch (expr)
{
  case VALUE:
    statements;
    break;
  case VALUE:
    statements;
    break;
  default:
    statements;
    break;
}
```

`The parameter break` is optional; however, it's recommended to use it in most cases, since otherwise code execution will continue to the body of the next `case` block. Add a break statement to the end of the last case as a precautionary measure, in case additional cases are added later. Strings can be used for the case values. Braces `{}` are required.

For loop statement with expression has the following syntax:

```
for (initial-expression; cond-expression; expression evaluated after each loop-round)
{
  statements;
}
```

For example, for loop can be used to print digits 1 to 10 in an alert box:

```
var msg = "";

for (var x = 1; x <= 10; x++)
   {
   msg = msg + x + "\n";
   }

alert(msg);
```

For ... in loop statement has the following syntax:

```
for (var property-name in object-name)
{
  statements using object-name[property-name];
}
```

For ... in loop iterates through all enumerable properties of an object, or the objects at all indices of an array. For example, it can be used to identify a type of used web browser. There are differences between the various web browsers with regard to which properties will be reflected with the for...in loop statement. In theory, this is controlled by an internal state property defined by the ECMAscript standard called "DontEnum", but in practice, each browser returns a slightly different set of properties during introspection.

While loop statement has the following syntax:

```
while (cond-expr)
{
  statements;
}
```

For example, *while loop* cab be used to print digits 1 to 10 in an alert box:

```
var msg = "";
var x = 1;
while (x <= 10)
   {
   msg = msg + x + "\n";
   x++;
   }
alert(msg);
```

Do ... while statement has the following syntax:

```
do {
  statements;
} while (cond-expr);
```

With statement has the following syntax:

```
with(document) {
  var a = getElementById('a');
  var b = getElementById('b');
  var c = getElementById('c');
};
```

Note the absence of `document` before each `getElementById()` invocation. The *with* statement indicates an object that will be used implicitly inside the statement body. The *with* statement is convenient because it avoids typing the same object names repeatedly. Using the *with* statement, the reference to the object may be shortened as below:

```
with (document)
{
  write("Hello from JavaScript");
  write("<<br />>");
  write("You can write what you like here");
}
```

JavaScript Functions: A function is a block with a (possibly empty) parameter list that is normally given a name. A function may give back a return value. The syntax of JavaScript function is:

```
function function-name(arg1, arg2, arg3)
{
  statements;
  return expression;
}
```

Anonymous functions are also possible such as the following:

```
var fn = function(arg1, arg2)
{
  statements;
  return expression;
};
```

For example, Euclid's original algorithm of finding the greatest common divisor that is a geometrical solution which subtracts the shorter segment from the longer:

```
function gcd(segmentA, segmentB) {
  while (segmentA != segmentB) {
    if (segmentA > segmentB)
      segmentA -= segmentB;
    else
      segmentB -= segmentA;
  }
  return segmentA;
}
```

The number of arguments (e.g., segmentA, segmentB) given when calling a function may not necessarily correspond to the number of arguments in the function definition; a named argument in the definition that does not have a matching argument in the call will have the value undefined.

Within the function the arguments may also be accessed through the arguments list; this provides access to all arguments using indices (e.g. arguments[0], arguments[1], ... arguments[n]), including those beyond the number of named arguments. Note that while the arguments list has a .length property, it is not an instance of Array; it does not have methods such as .slice(), .sort(), etc. Basic data types (strings, integers, etc.) are passed by value whereas objects are passed by reference.

JavaScript Events: These are the second main property of JavaScript apart of object-orientation, that it supports event-driving programming. Events are actions that can be detected by JavaScript. Every element on a web page has certain events that can trigger JavaScript functions. Examples of events are: mouse clicks, web page or an image loading, moving a mouse over a hot spot on the web page, selecting an input box in an HTML form, submitting an HTML form or a keystroke.

2.2.4 Javascript Object Hierarchy

Many JavaScript objects are contained within each other. JavaScript objects have a container to contained object relationship rather than a class and subclass relationship. Properties are not inherited from one type of object to another. There are two main types of JavaScript objects.

- Language Objects - objects provided by the language and are not dependent on other objects.
- Navigator - objects provided by the client browser. These objects are all sub objects to the navigator object.

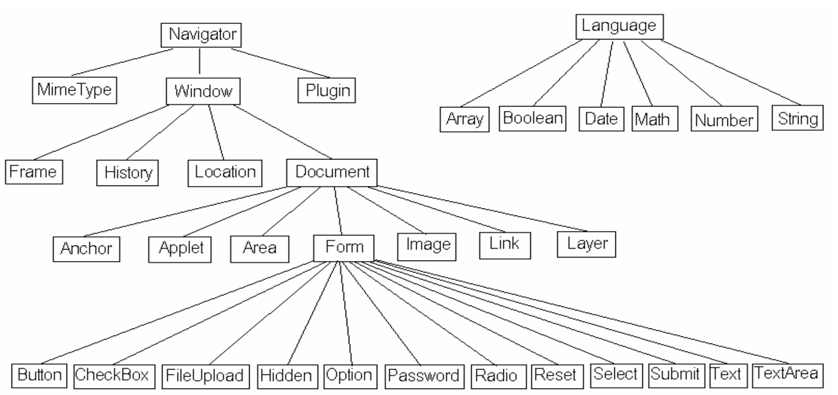
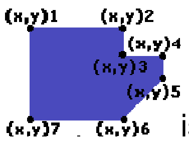


Figure: JavaScript Object Hierarchy (http://www.w3schools.com/default.asp)

2.2.5 JavaScript Image Map' Sample

An image map is a list of coordinates relating to a specific image, created in order to hyperlink areas of the image to various destinations (as opposed to a normal image link, in which the entire area of the image links to a single destination). For example, a map of the world may have each country hyperlinked to further information about that country. The intention of an image map is to provide an easy way of linking various parts of an image without dividing the image into separate image files. See Module 1, the Imagemaps section about HTML <map> tag and respective parameters.

The HTML syntax of the image map for the polygon  is (see Module 1 for argument's explanation if necessary):

```
<map name="poly">
<area shape="poly" coords="x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6, x7, y7"
href="page.html" alt="Alternate Text Here">
</map>
```

This example can be enxanced by using JavaScript. The events that can call a JavaScript can be added to the <area> tags inside the image map. The `<area>` tag supports the `onClick`, `onDblClick`, `onMouseDown`, `onMouseUp`, `onMouseOver`, `onMouseMove`, `onMouseOut`, `onKeyPress`, `onKeyDown`, `onKeyUp`, `onFocus`, and `onBlur` events.

Here is the above example, with some JavaScript added:

```
<html>
<head>
        <script type="text/javascript">
        function writeText(txt)
        {
        document.getElementById("desc").innerHTML=txt;
        }
        </script>
</head>

<body>

<img  src="SamplePolygon.gif"  width="145"  height="126"  alt="  Sample  Polygon”  usemap
="#SamplePolygon"/>

<map id ="SamplePolygon" name="SamplePolygon">
<area shape ="poly" coords =" x1, y1, x2, y2, x3, y3, x4, y4, x5, y5, x6, y6, x7, y7"
onMouseOver="writeText('Sample of clickable polygon.')" href ="Code_Sample.htm" target
="_blank" alt="Code_Sample" />
<p id="desc"></p>

</body>
</html>
```

Many examples of Javascript are included in the online manual at the http://www.w3schools.com/js/js_examples.asp and students can go and have a look at the JavaScript in action there.

2.3. *XML Encoding Basics*

2.3.1 Definition of XML

The Extensible Markup Language (XML) is a general-purpose markup language. It is classified as an extensible language because it allows its users to define their own tags. Its primary purpose is to facilitate the sharing of structured data across different information systems, particularly via the Internet. It is used both to encode documents and serialize data. XML is recommended by the World Wide Web Consortium. It is a fee-free open standard. The W3C recommendation specifies the XML lexical grammar, and the requirements for parsing. It started as a simplified subset of the Standard Generalized Markup Language (SGML), and is designed to be relatively human-legible. By adding semantic constraints, application languages can be implemented in XML. These include XHTML (XML-based HTML), GML (Geographic Markup Language), ArcXML (ESRI Arc Extensible Markup Language), KLM (Google Keyhole Markup Language) and many others. XML is sometimes used as the specification language for such application languages. XML is also language that is recommended by ISO and OGC standardization organizations for encoding geographic information schemas and data.

Advantages of XML are:

- It is text-based. It manifests as plain text files, which are less restrictive than other proprietary document formats.
- It supports Unicode, allowing almost any information in any written human language to be communicated.
- It can represent common computer science data structures: records, lists and trees. The hierarchical structure is suitable for most (but not all) types of documents.
- Its self-documenting format describes structure and field names as well as specific values.
- The strict syntax and parsing requirements make the necessary parsing algorithms extremely simple, efficient, and consistent.
- XML is heavily used as a format for document storage and processing, both online and offline.
- It is based on international standards.
- It can be updated incrementally.
- It allows validation using schema languages such as XSD and DTD, which makes effective unit-testing, firewalls, acceptance testing, contractual specification and software construction easier. Forward and backward compatibility are relatively easy to maintain despite changes in DTD or Schema.
- It is platform-independent, thus relatively immune to changes in technology.
- Its predecessor, SGML, has been in use since 1986, so there is extensive experience and software available.
- An element fragment of a well-formed XML document is also a well-formed XML document.

Disadvantages of XML are:

- XML syntax is redundant or large relative to binary representations of similar data. The redundancy may affect application efficiency through higher storage, transmission and processing costs.
- XML syntax is verbose, especially for human readers, relative to other alternative text-based data transmission formats.

- The hierarchical model for representation is limited in comparison to an object-oriented graph. Expressing overlapping (non-hierarchical) node relationships requires extra effort.
- XML namespaces are problematic to use and namespace support can be difficult to correctly implement in an XML parser.
- XML is commonly depicted as "self-documenting" but this depiction ignores critical ambiguities.
- The distinction between content and attributes in XML seems unnatural to some and makes designing XML data structures harder.
- Linking between XML documents requires the use of XLink, which is complex compared to hyperlinks
- It's hard to find an XML parser that is complete, correct, and efficient.

XML supports the direct use of almost any Unicode character (other than the ones that have special symbolic meaning in XML, itself, such as the open corner bracket, "<") in element names, attributes, comments, character data, and processing instructions. Therefore, the following is a well-formed XML document, even though it includes both Chinese and Cyrillic characters:

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

XML documents do not carry information about how to display the data.

2.3.2 XML Writing

There are two levels of correctness of an XML document:

- ***Well-formed*** XML document conforms to all of XML's syntax rules. For example, if a start-tag appears without a corresponding end-tag, it is not well-formed. A document that is not well-formed is not considered to be XML; a conforming parser is not allowed to process it.

- ***Valid*** XML document additionally conforms to some semantic rules. These rules are either user-defined, or included as an XML schema or DTD. For example, if a document contains an undefined element, then it is not *valid*; a *validating parser* is not allowed to process it.

2.3.3 XML Syntax Rules

As long as only well-formedness is required, XML is a generic framework for storing any amount of text or any data whose structure can be represented as a tree. The only indispensable syntactical requirement is that the document has exactly one *root element* (alternatively called the *document element*). This means that the text must be enclosed between a root start-tag and a corresponding end-tag. The following is a "well-formed" XML document:

```
<book>This is a book.... </book>
```

The root element can be preceded by an optional XML declaration. This element states what version of XML is in use (normally 1.0); it may also contain information about character encoding and external dependencies.

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

The W3C specification *requires* that processors of XML support the pan-Unicode character encodings UTF-8 and UTF-16 (UTF-32 is not mandatory). The use of more limited encodings, such as those based on ISO/IEC 8859, is acknowledged and is widely used and supported.

Comments can be placed anywhere in the tree, including in the text if the content of the element is text or `#PCDATA (#PCDATA` is a declaration of the element that has mixed content: character data or character data mixed with other elements).

XML comments start with `<!--` and end with `-->`. Two dashes (`--`) may not appear anywhere in the text of the comment.

```
<!-- This is a comment. -->
```

In most applications, additional markup is used to structure the contents of the XML document. The text enclosed by the root tags may contain an arbitrary number of XML elements. The basic syntax for one element is:

```
<name attribute="value">content</name>
```

The two instances of name are referred to as the start-tag and end-tag, respectively. Here, content is some text that may again contain XML elements. Therefore, a generic XML document contains a tree-based data structure. Here is an example of a structured XML document:

```
<recipe name="bread" prep_time="5 mins" cook_time="3 hours">
   <title>Basic bread</title>
   <ingredient amount="3" unit="cups">Flour</ingredient>
   <ingredient amount="0.25" unit="ounce">Yeast</ingredient>
   <ingredient amount="1.5" unit="cups" state="warm">Water</ingredient>
   <ingredient amount="1" unit="teaspoon">Salt</ingredient>
   <instructions>
     <step>Mix all ingredients together.</step>
     <step>Knead thoroughly.</step>
     <step>Cover with a cloth, and leave for one hour in warm room.</step>
     <step>Knead again.</step>
     <step>Place in a bread baking tin.</step>
     <step>Cover with a cloth, and leave for one hour in warm room.</step>
     <step>Bake in the oven at 350°F for 30 minutes.</step>
   </instructions>
 </recipe>
```

Attribute values must always be quoted, using single or double quotes; and each attribute name should appear only once in any element.

XML requires that elements be properly nested - elements may never overlap. For example, the code below is not well-formed XML, because the `em` and `strong` elements overlap:

```
<!-- WRONG! NOT WELL-FORMED XML! -->
<p>Normal <em>emphasized <strong>strong emphasized</em> strong</strong></p>

<!-- Correct: Well-formed XML. -->
<p>Normal         <em>emphasized         <strong>strong         emphasized</strong></em>
<strong>strong</strong></p>

<p>Alternatively      <em>emphasized</em>      <strong><em>strong      emphasized</em>
strong</strong></p>
```

XML provides special syntax for representing an element with empty content. Instead of writing a start-tag followed immediately by an end-tag, a document may contain an empty-element tag. An empty-element tag resembles a start-tag but contains a slash just before the closing angle bracket. The following three examples are equivalent in XML:

```
<foo></foo>
<foo />
<foo/>
```

An empty-element may contain attributes:

```
<info author="John" genre="science-fiction" date="2009-Jan-01" />
```

Entity references: An *entity* in XML is a named body of data, usually text. Entities are often used to represent single characters that cannot easily be entered on the keyboard. They are also used to represent pieces of standard (boilerplate) text that occur in many documents, especially if there is a need to allow such text to be changed in one place only.

Special characters can be represented either using entity references, or by means of numeric character references. An example of a numeric character reference is "`&#x20AC`;" that refers to the Euro symbol by means of its Unicode code-point in hexadecimal.

An entity reference is a placeholder that represents that entity. It consists of the entity's name preceded by an ampersand ("&") and followed by a semicolon (";"). XML has five predeclared entities:

`&amp;`	`&`	ampersand
`&lt;`	`<`	less than
`&gt;`	`>`	greater than
`&apos;`	`'`	apostrophe
`&quot;`	`"`	quotation mark

Here is an example using a predeclared XML entity to represent the ampersand in the name "AT&T":

```
<company_name>AT&amp;T</company_name>
```

Additional entities (beyond the predefined ones) can be written in the Document Type Definition (***DTD***) or XML Schema (***XSD***) documents. A basic example of doing so in a minimal internal DTD follows. Declared entities can describe single characters or pieces of text, and can reference each other.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE example
[
    <!ENTITY copy "&#xA9;">
    <!ENTITY copyright-notice "Copyright &copy; 2006, XYZ Enterprises">
]>
<example>
    &copyright-notice;
</example>
```

When viewed in a suitable browser, the XML document above appears as:

```
<example> Copyright © 2006, XYZ Enterprises </example>
```

Numeric character references: Numeric character references look like entity references, but instead of a name, they contain the `#` character followed by a number. The number (in decimal or "x"-prefixed hexadecimal) represents a Unicode code point. Unlike entity references, they are neither predeclared nor do they need to be declared in the DTD or XSD documents. They have typically been used to represent characters that are not easily encodable, such as an Arabic character in a document produced on a European computer. The ampersand in the "AT&T" example could also be escaped like this (decimal 38 and hexadecimal 26 both represent the Unicode code point for the "&" character):

```
<company_name>AT&#38;T</company_name>
<company_name>AT&#x26;T</company_name>
```

Well-formed documents: A well-formed document must conform to the following rules, among others:

- Non-empty elements are delimited by both a start-tag and an end-tag.
- Empty elements may be marked with an empty-element (self-closing) tag, such as `<IAmEmpty />`. This is equal to `<IAmEmpty></IAmEmpty>`.
- All attribute values are quoted with either single (`'`) or double (`"`) quotes. Single quotes close a single quote and double quotes close a double quote.
- Tags may be nested but must not overlap. Each non-root element must be completely contained in another element.
- The document complies with its declared character encoding. The encoding may be declared or implied externally, such as in `"Content-Type"` headers when a document is transported via HTTP, or internally, using explicit markup at the very beginning of the document. When no such declaration exists, a Unicode encoding is assumed, as defined by a Unicode Byte Order Mark before the document's first character. If the mark does not exist, UTF-8 encoding is assumed.

Element names are case-sensitive. For example, the following is a well-formed matching pair:

```
<Step> ... </Step>
```

Whereas this is not:

```
<Step> ... </step>
```

The careful choice of names for XML elements will convey the meaning of the data in the markup. This increases human readability while retaining the rigor needed for software parsing. Choosing meaningful names implies the semantics of elements and attributes to a human reader without reference to external documentation. However, this can lead to verbosity, which complicates authoring and increases file size.

Automatic verification: It is relatively simple to verify that a document is well-formed or validated XML, because the rules of well-formedness and validation of XML are designed for portability of tools. The idea is that any tool designed to work with XML files will be able to work with XML files written in any XML language (or XML application). One example of using an independent tool follows:

- Load XML document into an XML-capable browser, such as Firebox or Internet Explorer
- Use a tool like `xmlwf` (usually bundled with expat, stream-oriented XML 1.0 parser library, written in C)
- Parse the document, for instance in `Ruby`:

```
irb> require "rexml/document"
irb> include REXML
irb> doc = Document.new(File.new("test.xml")).root
```

XML semantics and validatation: By leaving the names, allowable hierarchy, and meanings of the elements and attributes open and definable by a customizable schema or DTD, XML provides a syntactic foundation for the creation of purpose specific, XML-based markup languages. The general syntax of such languages is rigid - documents must adhere to the general rules of XML, ensuring that all XML-aware software can at least read and understand the relative arrangement of information within them.

The language schema merely supplements the syntax rules with a set of constraints. Schemas typically restrict element and attribute names and their allowable containment hierarchies, such as only allowing an element named `'birthday'` to contain 1 element named `'month'` and 1 element named `'day'`, each of which has to contain only character data. The constraints in a schema may also include data type assignments that affect how information is processed; for example, the `'month'` element's character data may be defined as being a month according to a particular schema language's conventions, perhaps meaning that it must not only be formatted a certain way, but also must not be processed as if it were some other type of data.

An XML document that complies with a particular schema/DTD, in addition to being well-formed, is said to be ***valid***.

An XML schema is a description of a type of XML document, typically expressed in terms of constraints on the structure and content of documents of that type, above and beyond the basic constraints imposed by XML itself. A number of standard XML schema languages have emerged for the purpose of formally expressing such schemas, and some of these languages are XML-based, themselves.

Before, generalized data description languages such as SGML and XML, software designers had to define special file formats or small languages to share data between programs. This required writing detailed specifications and special-purpose parsers and writers.

XML's regular structure and strict parsing rules allow software designers to leave parsing to standard tools. As XML provides a general, data model-oriented framework for the development of application-specific languages, software designers need only concentrate on the development of rules for their data, at relatively high levels of abstraction.

Well-tested tools exist to validate an XML document "against" a schema: the tool automatically verifies whether the document conforms to constraints expressed in the schema. Some of these validation tools are included in XML parsers, and some are packaged separately.

Other usages of schemas exist: XML editors, for instance, can use schemas to support the editing process (by suggesting valid elements and attributes names, etc).

2.3.4 XML Schema (DTD and XSD)

The oldest schema format for XML is the Document Type Definition (DTD). While DTD support is ubiquitous due to its inclusion in the XML 1.0 standard, it several limitations:

- DTD has no support for newer features of XML, most importantly namespaces.
- It lacks expressiveness. Certain formal aspects of an XML document cannot be captured in a DTD.
- It uses a custom non-XML syntax, inherited from SGML, to describe the schema.
- DTD is still used in many applications because it is considered the easiest to read and write.

***Document Type Definition** (**DTD**)* is one of several SGML and XML schema languages, and is also the term a document that is authored in the DTD language. A DTD is primarily used for the expression of a schema via a set of declarations that conform to a particular markup syntax and that describe a class, or *type*, of SGML or XML documents, in terms of constraints on the structure of those documents. A DTD may also declare constructs that are not always required to establish document structure, but that may affect the interpretation of some documents.

DTD is native to the SGML and XML specifications, and since its introduction, other specification languages such as XML Schema (XSD) and RELAX NG have been released with additional functionality.

As an expression of a schema, a DTD specifies the syntax of an "application" of SGML or XML, such as the derivative language HTML or XHTML.

In a DTD, the structure of a class of documents is described via element and attribute-list declarations. Element declarations name the allowable set of elements within the document, and specify whether and how declared elements and runs of character data may be contained within each element. Attribute-list declarations name the allowable set of attributes for each declared element, including the type of each attribute value, if not an explicit set of valid value(s).

DTD is associated with an XML document via a Document Type Declaration, which is a tag that appears near the start of the XML document. The declaration establishes that the document is an instance of the type defined by the referenced DTD.

The declarations in a DTD are divided into an internal subset and an external subset. The declarations in the internal subset are embedded in the Document Type Declaration in the document itself. The declarations in the external subset are located in a separate text file. The external subset may be referenced via a public identifier and/or a system identifier.

Here is an example of a Document Type Declaration containing both public and system identifiers:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

Here is an example of a Document Type Declaration that encapsulates an internal subset consisting of a single entity declaration:

```
<!DOCTYPE foo [ <!ENTITY greeting "hello"> ]>

<!DOCTYPE bar [ <!ENTITY greeting "hello"> ]>
```

All HTML 4.01 documents are expected to conform to one of three SGML DTDs. The public identifiers of these DTDs are constant and are as follows:

- `-//W3C//DTD HTML 4.01//EN`
- `-//W3C//DTD HTML 4.01 Transitional//EN`
- `-//W3C//DTD HTML 4.01 Frameset//EN`

The system identifiers of these DTDs, if present in the Document Type Declaration, will be URI references. System identifiers can vary, but are expected to point to a specific set of declarations in a resolvable location. SGML allows for public identifiers to be mapped to system identifiers in *catalogs* that are optionally made available to the URI resolvers used by document parsing software.

A common misconception is that non-validating XML parsers are not required to read DTDs, when in fact, the DTD must still be scanned for correct syntax as well as for declarations of entities and default attributes. A non-validating parser may, however, elect not to read *external entities*, including the external subset of the DTD. If the XML document depends on declarations found only in external entities, it should assert `standalone="no"` in its XML declaration.

An example of a very simple XML DTD to describe a list of persons is given below:

```
<!ELEMENT people_list (person*)>
<!ELEMENT person (name, birthdate?, gender?, socialsecuritynumber?)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT birthdate (#PCDATA)>
<!ELEMENT gender (#PCDATA)>
<!ELEMENT socialsecuritynumber (#PCDATA)>
```

Taking this line by line, it says:

1.) `People_list` is a valid element name, and an instance of such an element contains any number of person elements. The `*` denotes there can be 0 or more person elements within the `people_list` element.

2.) `Person` is a valid element name, and an instance of such an element contains one element named `name`, followed by one named `birthdate` (optional), then `gender` (also optional) and socialsecuritynumber (also optional). The ? indicates that an element is optional. The reference to the name element name has no ?, so a person element *must* contain a name element.

3.) Name is a valid element name, and an instance of such an element contains "parsed character data" (#PCDATA).
4.) Birthdate is a valid element name, and an instance of such an element contains character data.
5.) Gender is a valid element name, and an instance of such an element contains character data.
6.) Socialsecuritynumber is a valid element name, and an instance of such an element contains character data.

An example of an XML file that makes use of and conforms to this DTD follows. It assumes the DTD is identifiable by the relative URI reference "example.dtd":

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE people_list SYSTEM "example.dtd">
```

```
<people_list>
  <person>
    <name>Fred Bloggs</name>
    <birthdate>27/11/2008</birthdate>
    <gender>Male</gender>
  </person>
</people_list>
```

It is possible to render this in an XML-enabled browser (such as Interner Explorer 5 or Mozilla) by pasting and saving the DTD component above to a text file named example.dtd and the XML file to a differently-named text file, and opening the XML file with the browser. The files should both be saved in the same directory. However, many browsers do not check that an XML document conforms to the rules in the DTD; they are only required to check that the DTD is syntactically correct. For security reasons, they may also choose not to read the external DTD.

While DTD supports in XML tools is widespread due to its inclusion in the XML 1.0 standard, it is seen as limited as described above. There newer XML schema languages that are much more powerful are increasingly favored over DTDs that are:

- XML Schema, also referred to as XML Schema Definition (XSD), has achieved Recommendation status within the W3C.
- RELAX NG, which is also a part of DSDL, is an ISO international standard.
- Document Structure Description (DSD), attempts to combine an expressive schema balanced with ease of use.

XML Schema (XSD) is a newer XML schema language, described by the W3C as the successor of DTDs, is XML Schema, or more informally referred to by the initialism for XML Schema instances, XSD (XML Schema Definition). XSDs are far more powerful than DTDs in describing XML languages. They use a rich data-typing system, allow for more detailed constraints on an XML document's logical structure, and must be processed in a more robust validation framework. XSDs also use an XML-based format that makes it possible to use ordinary XML tools to help process them, although XSD implementations require much more than just the ability to read XML.

XSDs were the first W3C-recommended XML schemas to provide a namespace and datatype aware alternative to using XML's native Document Type Definitions (DTDs).

Criticisms of XSD include the following:

- The specification is very large that makes it difficult to understand and implement.
- The XML-based syntax leads to verbosity in schema description that makes XSDs harder to read and write.
- Schema validation can be an expensive addition to XML parsing, especially for high volume systems.
- The modeling capabilities are very limited, with no ability to allow attributes to influence content models.
- The type derivation model is very limited; in particular, that derivation by extension is rarely useful.

Like all XML schema languages, XML Schema can be used to express a schema: a set of rules to which an XML document must conform in order to be considered *valid* according to that schema. An XSD defines a type of XML document in terms of constraints upon what elements and attributes may appear, their relationship to each other, what types of data may be in them, and other things. It can be used with validation software in order to ascertain whether a particular XML document is of that type, and to produce a PSVI (Post-Schema-Validation Infoset, where each element, attribute and in general, any node of the XML document has assigned the type from XML schema).

However, unlike most other schema languages, XML Schema was also designed with the intent that determination of a document's validity would produce a collection of information adhering to specific data types. Such a post-validation infoset can be useful in the development of XML document processing software, but the schema language's dependence on specific data types has provoked criticism.

An XML Schema instance is an XML Schema Definition (XSD) written in XML Schema and typically has the filename extension `.xsd`. The language itself is sometimes informally referenced as XSD. It has been suggested that WXS (for W3C XML Schema) is a more appropriate initialism though this acronym has not been in a widespread use and W3C working group rejected it. XSD is also an initialism for XML Schema Datatypes, the *datatype* portion of XML Schema.

There are 19 primitive datatypes specified. There are several proposed solutions on embedding binary data into xml (such as an image). Binary data could be considered an opaque data type. An opaque type gets its name from the fact that the data model knows nothing about the internal representation of the type. An example of a very simple XML Schema Definition to describe a country is given below.

```
<xs:schema
  xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="country" type="Country"/>
    <xs:complexType name="Country">
        <xs:sequence>
            <xs:element name="name" type="xs:string"/>
            <xs:element name="population" type="xs:decimal"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

An example of an XML document that conforms to this schema is given below (assuming the schema file name is `country.xsd`):

```
<country
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="country.xsd">
    <name>France</name>
    <population>59.7</population>
</country>
```

A unique Internet Media Type is not yet registered for XSDs, so "application/xml" or "text/xml" should be used, as per RFC 3023 (RFC refers to a Request for Comments, a series of memoranda encompassing new research, innovations, and methodologies applicable to Internet technologies, number 3023 refers to standards for XML and XSD).

2.3.5 XSL and XSLT

XML documents do not carry information about how to display the data. Without using CSS or XSL (eXtensible Stylesheet Language, a family of transformation languages which allows one to describe how files encoded in the XML standard are to be formatted or transformed), a generic XML document is rendered as raw XML text by most web browsers. Some display it with 'handles' (e.g. + and - signs in the margin) that allow parts of the structure to be expanded or collapsed with mouse-clicks. In order to style the rendering in a browser with CSS, the XML document must include a reference to the stylesheet.

The eXtensible Stylesheet Language (XSL) is a family of transformation languages that allows one to describe how files encoded in the XML standard are to be formatted or transformed. XSL is designed to be data driven and strongly encourages the inversion of control design pattern. There are several languages in the family that are available in the form of W3C recommendations. The XSL family includes the following:

- ***XSL Transformations** (**XSLT**)* is language for transforming XML documents**.** There are currently many XSLT implementations available. Several web-browsers including Internet Explorer, Firefox, Mozilla, and Netscape, and Opera, all support transformation of XML to HTML through XSLT. The figure below shows a transformation example.

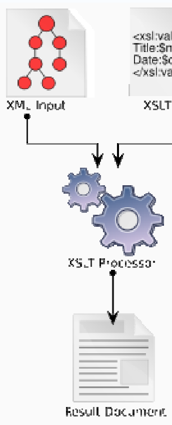


Figure: The basic elements and process flow of Extensible Stylesheet Language Transformations (Source: http://en.wikipedia.org/wiki/XSL_Transformations)

XSLT is developed by the W3C. As of 2007, XSLT 1.0 is more widely used and implemented. However, the most recent version is XSLT 2.0, which reached W3C recommendation status on 23 January 2007. Originally, XSLT was part of the W3C's Extensible Stylesheet Language (XSL) development effort of 1998-1999, a project which also produced XSL Formatting Objects and the XML Path Language, XPath.

As a language, XSLT is influenced by functional languages, and by text-based pattern matching languages like SNOBOL and awk. Its most direct predecessor was DSSSL, a language that performed the same function for SGML that XSLT performs for XML. XSLT can also be considered as a template processor.

- ***XSL Formatting Objects (XSL-FO)*** is language for specifying the visual formatting of an XML document and it is becoming more widespread.

- ***The XML Path Language (XPath)*** is used by XSLT as a means of navigating an XML document and is part of the XSL family.

- ***The XML Query Language (XQuery)*** is another W3C project that is intended to provide similar capabilities for querying XML documents using XPath.

Extensible Stylesheet Language (XSL) can be used to alter the format of XML data, either into HTML or into other "human-readable" formats that are suitable for a browser to display. The original document is not changed; rather, a new document is created based on the content of an existing one. The new document may be serialized (output) by the processor in standard XML syntax or in another format, such as HTML or plain text.

XSLT is most often used to convert data between different XML schemas or to convert XML data into HTML or XHTML documents for web pages, creating a dynamic web page, or into an intermediate XML format that can be converted to PDF documents. An XSLT processor can use an XSL *stylesheet* as a guide for such conversions.

The XSLT processing model involves:

- One or more XML *source* documents
- One or more XSLT *stylesheet* modules
- The XSLT template processing engine (the *processor*); and
- One or more *result* documents

The XSLT processor ordinarily takes two input documents - an XML *source* document, and an XSLT *stylesheet* - and produces an *output* document. The XSLT stylesheet contains the XSLT program text (or source code in other languages) and is itself an XML document. It describes a collection of template rules: instructions and other directives that guide the processor in the production of the output document.

The XSLT language is declarative - rather than listing an imperative sequence of actions, template rules only define how to handle a node matching a particular XPath-like pattern if the processor should happen to encounter one, and the contents of the templates effectively comprise functional expressions which directly represent their evaluated form: the result tree that is the basis of the processor's output.

The processor follows a fixed algorithm: Assuming a stylesheet has already been read and prepared, the processor builds a *source tree* from the input XML document. It then starts by processing the source tree's root node, finding in the stylesheet the best-matching template for that node, and evaluating the template's contents. Instructions in each template generally direct the processor to either create nodes in the result tree, or process more nodes in the source tree in the same way as the root node. Output is derived from the result tree.

XSLT processor implementations fall into two main categories: server-side, and client-side. To specify *client-side* XSLT, the following processing instruction is required in the XML:

```
<?xml-stylesheet type="text/xsl" href="myTransform.xslt"?>
```

XSLT processors may be delivered as standalone products, or as components of other software including web browsers, application servers, frameworks such as Java and .NET, or even operating systems. For example, Windows XP comes with the MSXML3 library, which includes an XSLT processor. Most of the earlier XSLT processors were interpreters.

Client-side XSLT is supported by many web browsers with XSLT processors. An alternative, rather than being dependent on the end-user's browser capabilities, is to use XSL to convert XML into a displayable format *on the server*. The end-user is not aware of what has gone on behind the scenes; all they see is well-formatted, displayable data.

Example of incoming XML document is:

```
<?xml version="1.0" ?>
<persons>
  <person username="JS1">
    <name>John</name>
    <family_name>Smith</family_name>
  </person>
  <person username="MI1">
    <name>Morka</name>
    <family_name>Ismincius</family_name>
  </person>
</persons>
```

Example of transforming XML to XML with the XSLT stylesheet that provides templates to transform the XML document is:

```
<?xml version="1.0" ?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
<xsl:output method="xml" indent="yes"/>

<xsl:template match="/">
       <root> <xsl:apply-templates/> </root>
</xsl:template>

<xsl:template match="//person">
        <name username="{@username}">
           <xsl:value-of select="name" />
        </name>
</xsl:template>

</xsl:stylesheet>
```

Its evaluation results in a new XML document, having another structure:

```
<?xml version="1.0" encoding="UTF-8"?>
<root>
      <name username="JS1">John</name>
      <name username="MI1">Morka</name>
</root>
```

Example of transforming XML to XHTML) with XSLT stylesheet is:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

<xsl:template match="/persons">
        <html xmlns="http://www.w3.org/1999/xhtml">
        <head> <title>Testing XML Example</title> </head>
        <body>
                <h1>Persons</h1>
                <ul>
                <xsl:apply-templates select="person">
                        <xsl:sort select="family_name" />
                </xsl:apply-templates>
                </ul>
        </body>
        </html>
</xsl:template>

<xsl:template match="person">
        <li>
                <xsl:value-of select="family_name"/>,
                <xsl:value-of select="name"/>
        </li>
</xsl:template>

</xsl:stylesheet>
```

XSLT relies upon the W3C's XPath language for identifying subsets of the source document tree, as well as for performing calculations. XPath also provides a range of functions, which XSLT itself further augment. This reliance upon XPath adds a great deal of power and flexibility to XSLT.

XSLT 2.0 relies on XPath 2.0; both specifications were published on the same date. Similarly, XSLT 1.0 works with XPath 1.0.

XPath is an expression language for portions of XML documents, or computing values (such as strings, numbers, or boolean values) based on the content of an XML document. The XPath language is based on a tree representation of the XML document. It enables navigation around the tree. In this, nodes are selected by a several criteria. The commonest XPath expression (after which XPath is named) is a path expression. This is written as a sequence of steps to get from one XML node (the current context node) to another node or set of nodes. The steps are separated by "/" (i.e. path) characters. Each step has three components:

- Axis Specifier
- Node Test
- Predicate

For example, given source a XML containing at least:

```
<A><B><C></C></B></A>
```

The simplest XPath is of the form:

```
/A/B/C
```

This selects C elements that are children of B elements that are children of the A element that is the outermost element of the XML document.

2.4. Conclusion

This module has examined scripting languages and several associated technologies, including DHTML, HTML Document Object Models (DOMs), JavaScript, XML, XSL and XSLT. These applications were examined as tools for the development, maintenance and modification of webpages. These topics build on the earlier modules, especially topics covering HTML, simple web creation and GIS applications. This module complements the topics of the earlier modules, as web creation is one of the most important aspects of information management, and a vital aspect of information infrastructure at international, national, local and individual user levels.

Several key issues must be noted to fully benefit from the topics covered in this module:

- Scripting languages are undergoing constant change, the main focus being on accuracy, ease of use, application to client and server-side uses, flexibility and sophistication.
- These techniques involve both software development and standards. Standards are also important, because of security and consumer needs. Therefore, understanding of software developments must acknowledge the importance of regulations.
- Although scripting languages may seem quite complicated, they have a basic logic which allows the creation of webpages and a degree of flexibility and creativity for the user.
- No scripting language is perfect, but as has been seen in this module, advantages and disadvantages are universal. Therefore, a critical assessment is necessary to understand these languages.
- JavaScript is one of the most important scripting languages, but it also works in a larger environment, which requires knowledge of other web related techniques and languages such as HTML and DHTML.
- XML is also important, its key issues concerning the text-based format, which is less restrictive than other document formats, the near universal applicability of Unicode, and the generally useful hierarchical structure, also its simple and consistent algorithms and possibilities for incremental updating. These are important, despite the disadvantages of verbosity, problems of relation to binary representations and the limitations of the hierarchical model for representation. Therefore, as with JavaScript, critical assessment of this language is necessary.

Module self-study questions:

1. Describe the difference between DHTML and HTML. Do you think that the DHTML is a more effective web page manager and creator than HTML?
2. Describe the main characteristics of the World Wide Web Consortium (W3C). What are the main impacts that this organization has had on the development of the Internet?
3. Describe the functions of the Document Object Model. What specifications and items are included in Levels 0, 1, 2 and 3?
4. What are the main functions of JavaScript in web programming and website creation? Describe with examples the main principles of JavaScript Statements.
5. What are the main advantages and disadvantages of XML? Describe the differences between the XML Schemas (DTD and XSD)?

Suggested Readings:

[1] JavaScript Tutorial at the http://www.w3schools.com/js/
[2] XML Tutorial at the http://www.w3schools.com/xml/
[3] XSL Tutorial at the http://www.w3schools.com/xsl/

References:

[1] W3Schools, http://www.w3schools.com/default.asp
[2] Flanagan, D., JavaScript: The Definitive Guide, 5th Edition, O'Reilly & Associates, 2006
[3] Duffy, S., How to do Everything with JavaScript, Osborne, 2003
[4] Vander Veer, Emily A., JavaScript For Dummies, 4th Edition, Wiley Pub, 2004
[5] Powell, T., Schneider, F., JavaScript: The Complete Reference. McGraw-Hill Companies, 2001
[6] XML on Open Directory Project, http://www.dmoz.org/Computers/Data_Formats/Markup_Languages/XML/
[7] van der Vlist, E., Van Comparing XML Schema Languages, 2001, http://www.xml.com/pub/a/2001/12/12/schemacompare.html
[8] XHTML2 Working Group Home Page, W3C, http://www.w3.org/MarkUp/
[9] Ousterhout, John K. Scripting: Higher Level Programming for the 21st Century, http://home.pacbell.net/ouster/scripting.html
[10] Prechelt, L., Are Scripting Languages Any Good? A Validation of Perl, Python, Rexx, and Tcl against C, C++, and Java, 2002, http://page.mi.fu-berlin.de/prechelt/Biblio/jccpprt2_advances2003.pdf
[11] The Extensible Stylesheet Language Family (XSL), W3C, http://www.w3.org/Style/XSL/
[12] XSL, The Cover Pages Web Site, http://xml.coverpages.org/xsl.html

Terms used

- Active Server Pages
- Client Side Scripting
- DHTML
- Document Object Model (DOM)
- Document Type Declaration (DTD)
- Encoding
- JavaScript
- Jscript
- Markup
- Object Hierarchy,
- Object-Oriented Programming
- Perl
- PHP
- Python
- Scripting Languages
- Server-Side Scripting
- Structuring Components
- Vbscript
- W3C Recommendation
- W3C Standards
- XHTML
- XML
- XSD
- XSL
- XSLT

3. Web Mapping and Related Technologies

Internet mapping technology has opened new paths for disseminating, sharing, displaying, and processing spatial information on global and local networks. Web-based solutions provide a low-cost, efficient way to deliver map products to users. This module discusses web mapping and related technology and specifications. The first to be discussed is multi-tiered architecture that serves as the environment for web-mapping services. AJAX web technology, often used for web mapping applications on the front client tier, is also overviewed. Next, the three-tiered architecture is analyzed for web mapping applications and systems with thin (Web Mapping Server - WMS) and thick (Web Feature Server - WFS) web mapping clients are also introduced. Further, the OGC standard implementation specifications that defined functionalities of Web Mapping Server, Web Feature Server, and Web Coverage Server are discussed in detail. These specifications have to be used by developers to build interoperable distributed web mapping systems. The OGC Web Map Service specification defines a set of functions that clients may use to interact with WMS/WFS/FCS providers (servers).

Module Outline

Topic 1: Multi-tiered Architecture
Topic 2: AJAX Technology
Topic 3: Web Mapping Software Architecture
Topic 4: WMServices
Topic 5: WFServices
Topic 6: Web Coverage Service
Topic 7: Filter Encoding
Topic 8: Styled Layer Descriptor and Symbology Encoding
Topic 9: Conclusion

3.1. Multi-tiered Architecture

Software architecture, described as strategic software design, includes architectural styles. There are a few software architectural (strategic) styles including standalone, client-server, peer-to-peer, etc. In general, a stand-alone, single-tier application would be something that runs only on the client's machine.

Client-server or two-tier is a computing architecture that separates a client from a server and can be implemented over a computer network. The most basic type of client-server architecture employs only two types of nodes: clients and servers. Client-server applications consist of a client that provides a user interface and a server that provides persistent storage, licensing, heavy processing, etc.

Each instance of the client software can send requests to one or more connected servers. The servers can accept these requests, process them, and return the requested information to the client. This concept can be applied to many different kinds of applications. The fundamentals of client-server architecture remain the same. Nowadays, the typical client is a web browser and servers include web servers, database servers, and email servers.

Some client-server architecture designs can be more sophisticated and can contain a number of nodes. Multi-tier client-server architecture makes use of three (three-tiered) or more than three tiers (multi-tiered or n-tier architecture), each of which may make use of several applications or services. A ***multi-tiered*** application is run by more than one distinct software agent that may reside on a few hardware and software platforms. For example, it can include *clients*, *application servers,* which process data for the clients, and *database servers*, which store data for the application servers. The *three-tiered* architecture is the most commonly used.

The advantage of *n-tiered* architectures is that they are far more *scalable*, since they balance and distribute the processing load among multiple, often redundant, specialized server nodes. This, in turn, improves overall system performance and reliability, since more of the processing load can be accommodated simultaneously.

Software nodes or instances of ***peer-to-peer*** *(P2P)* architecture can simultaneously act as both a client and a server; each has equivalent responsibilities and status. Both client-server and P2P architectures are in wide usage today.

One of the major initiatives in the development of GIS technology is the adaptation of IT technology that directly uses multi-tier software architecture for operational systems organization. Many spatial solutions started with the transition to web-based distributive and open n-tier architecture in the late 1990s. However, most of the current map web applications provide only cartographic renderings and simple processing analyses, with most of the computational functionalities still placed in the client tier. Most of the current web map servers are a simplification of full functional application servers at the middle of the three-tier industry-standard architecture.

Map application servers provide remote access to GIS data for users or applications through the internet/intranet with secure access using simple point-and-click interfaces. Through an application server, the following map services are available:

- Access and visualization of spatial data

- Remote submission of spatial data processing
- Dynamical check on the job status
- Spatial analysis
- Security control over data access and transmission
- Data integration from heterogeneous sources
- Plugging in a legacy application

These map services can be deployed on different layers of n-tier architecture and be distributed across the networks. Currently there are two major application development models (API) - Java 2 Enterprise Edition (J2EE, http://en.wikipedia.org/wiki/Enterprise_application) and .NET (http://en.wikipedia.org/wiki/.NET) - both used to build n-tier mapping applications. They are founded upon similar architectural principles, and both are based on the concept of multi-tiered architecture. The feasible mapping n-tier architecture is shown in the figure below.

GIS functionality, data, and metadata can be assigned to various tiers (that can be subdivided by layers) along a network and can be found on the server side (in one or more intermediate middleware layers), either on the back-end or client side. All tiers can be independently configured to meet the users' requirements and scaled to meet future requirements.

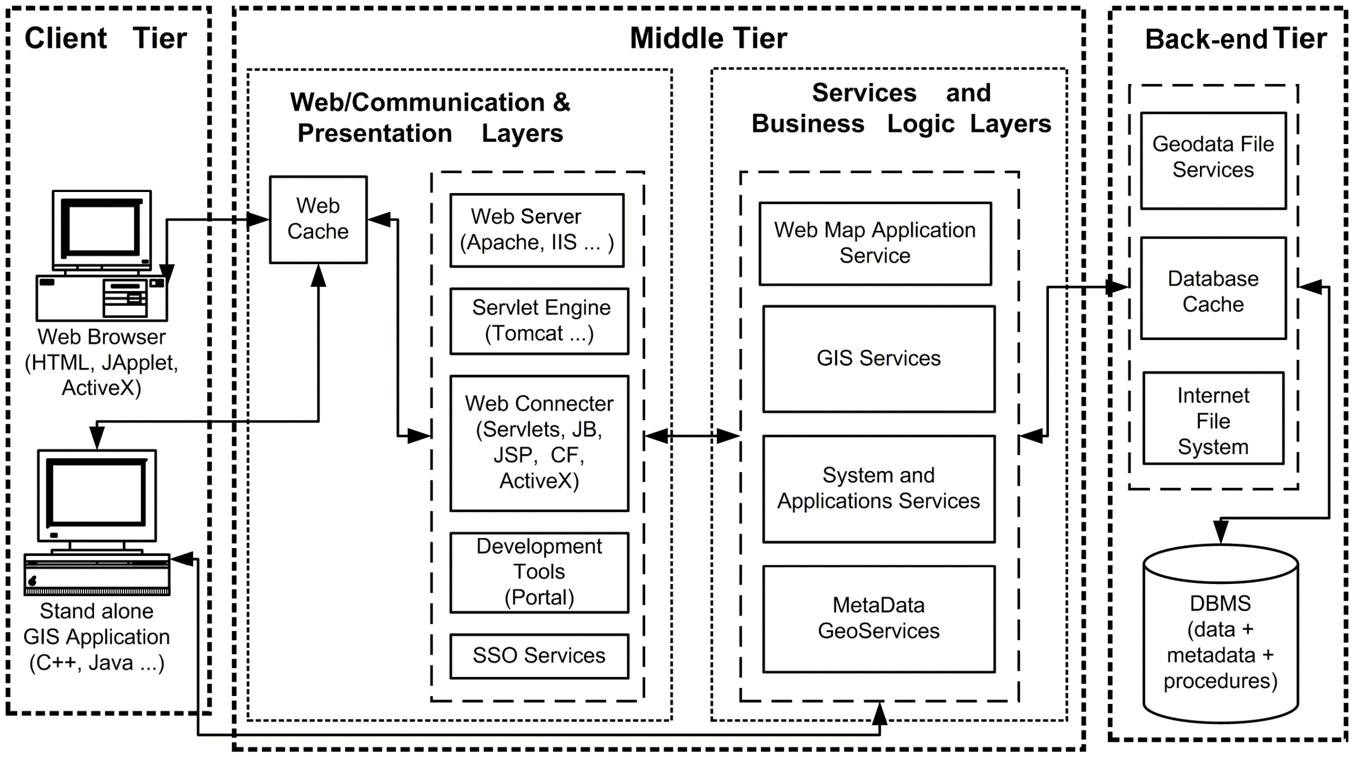

Client Tier
Middle Tier
Back-end Tier
Web/Communication & Presentation Layers
Services and Business Logic Layers
Web Browser (HTML, JApplet, ActiveX)
Stand alone GIS Application (C++, Java ...)
Web Cache
Web Server (Apache, IIS ...)
Servlet Engine (Tomcat ...)
Web Connecter (Servlets, JB, JSP, CF, ActiveX)
Development Tools (Portal)
SSO Services
Web Map Application Service
GIS Services
System and Applications Services
MetaData GeoServices
Geodata File Services
Database Cache
Internet File System
DBMS (data + metadata + procedures)

Figure: The feasible GIS n-tier architecture

A three-tier distributed client-server architecture (as shown in Figure) includes a user system interface *client tier* where user services reside. Client tiers can be represented by web browsers only (thin client), either web browser with rendering add-ons, such as SVG (medium client); or thick client such as Java applets, ActiveX components (http://en.wikipedia.org/wiki/ActiveX) or GIS applications; or wireless device (could be a thin client). Distributed systems with web browsers can employ web servers using HTTP, HTTPS or SOAP protocols. The web server forwards the request to presentation and business logic tier through an appropriate web connector that may support the CGI (http://en.wikipedia.org/wiki/Common_Gateway_Interface), NSAPI, and ISAPI methods of interfacing. The thick client can use Object Request Brokers (ORB, http://en.wikipedia.org/wiki/Object_request_broker) based on the CORBA, RMI or SOAP. Either DCOM specifications are used to communicate with the application server.

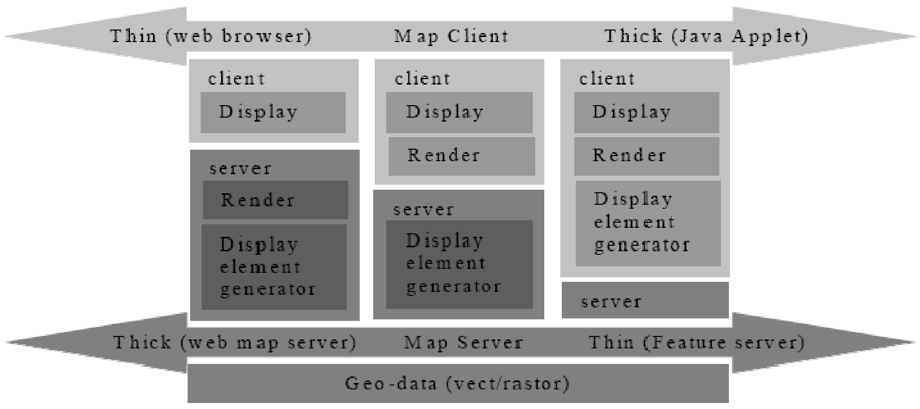


Figure: Three kinds of map servers and corresponding map clients (Zhang S. and Goddard S., 2003)

The standard request-response protocol used between the *client* and web server of *middle tiers* is ***HTTP/HTTPS (Hypertext Transfer Protocol/Secure Hypertext Transfer Protocol)***. Web services most commonly rely on the HTTP protocol. Other protocols are layered on it (e.g., SOAP messages transmitted over HTTP). When a client is makes an HTTP request the responding server stores or creates *resources* such as HTML files and images. In between the user's client and origin server several intermediaries may be found, such as proxies, gateways, and tunnels. HTTP is not constrained to using TCP/IP and its supporting layers (see Module 1). HTTP can be implemented on top of any other protocol on the Internet, or on other network.

Typically, an HTTP client initiates a request by establishing a Transmission Control Protocol (TCP) connection to a particular port on a host (port 80 by default). An HTTP server listening on that port waits for the client to send a request message. Upon receiving the request, the server sends back a status line, such as "HTTP/1.1 200 OK", and a message that can include the requested file, an error message, or some other information.

Resources to be accessed by HTTP are identified using Uniform Resource Identifiers (URIs) or Uniform Resource Locators (URLs) using the http or https URI schemes.

Often, the middle tier is divided in two or more subsystems or layers with different functions and security features. A server tier can be presented by a web server with web server connectors, such as JSP (http://en.wikipedia.org/wiki/JSP), Servlets (http://en.wikipedia.org/wiki/Servlets), or ASP (http://en.wikipedia.org/wiki/ASP)); system, application, and transaction (e.g. Enterprise JavaBeans or .NET) components. The thick client can directly access EJB components by passing a web server by using the RMI-JRPM or CORBA-IIOP standards (http://en.wikipedia.org/wiki/CORBA).

The middle tier provides process management services such as protocol management, HTML/XML data streaming, system, administrative, and applications services (e.g. load balancing, data access management, cashing, etc.) and transactions that are shared by multiple applications. Mapping web services perform specific mapping and GIS functions that can include cartographic image rendering, spatial feature data streaming, geo-coding and routing, map metadata publishing and browsing, spatial data integration from different formats, data download, spatial and cartographic generalization, and spatial queries - all can be integrated as part of a larger application server.

Communication and presentation layer logic controls the application interface with the user by processing requests, generating content in response, and formatting and delivering (image maps, data streams etc.) returns to the user*.* Servlet, JSP or ASP process incoming requests and handles the response. Business logic is normally off-loaded to EJB or .NET business components and output is usually offloaded to JSP or ASP pages.

Within business logic layer, EJB or .NET, business components are invoked by a servlet, JSP or ASP to handle most of the application's business logic and data processing. The components enable business logic to be persistent across calls and are designed to work closely with JDBC API (http://en.wikipedia.org/wiki/JDBC) or ADO.NET (http://en.wikipedia.org/wiki/ADO) for database transactions. Business logic describes the activities that involve the generation of specific content: storing and retrieving data and performing computations on that data. A component of this layer handles the load distribution of incoming spatial requests, and catalogs, which services are running on which spatial components. Spatial components have capabilities for accessing and bundling maps and data into the appropriate format before sending the data back to a client. These components can support different functionalities: generate image maps and stream vector spatial data for the client; return attribute data for spatial and tabular queries; execute geo-coding and routing functions; extract and return spatial data in appropriate format; search a spatial metadata repository for documents related to spatial data and services; run spatial and cartographic generalization techniques; etc.

Data management layer controls database storage and retrieval. The JDBC API or ADO.NET is available to components, as are all APIs, although database transactions are usually controlled by EJB or .NET components in the model. Data access logic describes transactions with a database. Data access is normally performed as a functionality of business logic. As many spatial data are still stored in file format, the management of this data may be significantly improved by storing data within corporate database systems.

For the classic three-tier architecture, the third *back-end tier* represents a single or multiple database, file server, or other back-end enterprise information source – all provide database and file management functionality. Hybrid approaches can be offered when database management components provide possibilities to store and manage spatial data files within a single repository. The data management component ensures the integrity, consistency, and security of data, especially in distributed environment.

The database layer contains a collection of relational database tables, including LOB objects (http://en.wikipedia.org/wiki/BLOB) stored flat files, metadata tables, and stored procedures. In the latter case, all functions and procedures can have fast and efficient access to the information stored inside the database.

Alternatively, the back-end tier may be designed for High Performance Computing (e.g. geoprocessing, see at the http://en.wikipedia.org/wiki/High_Performance_Computing). It is the classic three-tier architecture except that the back-end can provide access to operations such as topology “cleaning”, generalization, overlays, data conversion, transformation, and other heavy geo-processing services, which results in a more efficient workflow. Mapping application servers offer flexible ways of integrating legacy systems into any new development effort by using a variety of methods for integrating legacy systems into web-based solutions.

3.2. AJAX Technology

Another web technology that is often used for web mapping applications is ***AJAX*** - Asynchronous JavaScript plus the XML approach (http://www.adaptivepath.com/ideas/essays/archives/000385.php).

The Internet was designed for browsing HTML documents and the classic web application model adopts a "click, wait, and refresh" user interaction paradigm and a synchronous request-response communication mechanism. This user interaction paradigm results in slow, unreliable, low productivity, and inefficient web applications.

In the AJAX approach a "click, wait, and refresh" user interaction model is replaced with "*partial screen update*". The AJAX-based application updates only user interface elements that contain new information; the rest of the user interface remains displayed without refreshing.

A synchronous request-response communication mechanism is replaced by an *asynchronous* communication model. For an AJAX-based application, the request-response can be an asynchronous interaction between user and a client browser and interaction between browser and server. Thus, the user can continue to interact with the application while the client program requests information from the server in the background. When a server response arrives, only the related user interface portion is updated.

However, despite the abbreviated meaning (Asynchronous JavaScript + XML approach), AJAX is not specific to a particular programming language, data exchange format, or network communications object. AJAX is only a web application model that employs *partial screen update* and *asynchronous communication*. There are three technologies for building the AJAX client engine, of which DHTML/JavaScript (asynchronous JavaScript + XML), Java (asynchronous Java + XML), and Flash (asynchronous ActionScript + SWF) are the most commonly used.

AJAX differs from a web multi-tired software architecture in that it adds a client-side engine as an intermediate layer between the user interface and the server. The user interacts with the client-side engine; the client-side engine interacts with the server.

AJAX technology is adapted to web mapping purposes (e.g. Google Map). This provides an enhanced performance, thus only map image from whole application is updated when user request to update a map, the rest of application does not require the refresh. By using AJAX, a web map application draws and zooms quickly, pans smoothly, and can be extended to display a wide variety of information.

3.3. Web Mapping Software Architecture

The first known geographic web site, Xerox, was published at the http://mapweb.parc.xerox.com/map in 1993. The world map had been put in this site with multiple zoom level options. A large number of map images, for each case scale and extent scenario, were pre-built. This was not a real web map server, as understood today, however the conceptual idea was similar.

The first middle-tier map web application, ArcView Internet Map Server, was built by ESRI. Since then, ESRI has created a few generations of its IMS product and now offers ArcGIS Server to provide not only web mapping capabilities, but also togeoprocess, geocode, network, etc. The immense popularity and acceptability that the Internet has gained over the last decade has attracted GIS vendors throughout the world. Other major GIS commercial vendors and their web mapping products include MapInfo with MapExtrem, Autodesk with MapGuide, and Intergraph with GeoMedia WebMap. In addition, open source and non-commercial organizations are offering their products for developers and users, such as MapServer, GeoServer, etc.

Strictly speaking, web-mapping applications are not GIS. Functionality of current IMS is much poorer compared to desktop GIS. Only ESRI's ArcServer could be considered a web application capable to doing a real GIS job.

In general, the geo-spatial data presentation (portrayal) can be divided into four processes (Cuthbert, 1997):

1. The filtering or selection of geo-spatial data to be displayed
2. The generation of display elements from the selected geo-spatial data
3. The rendering of display elements into a rendered map
4. The display of the rendered map to the user

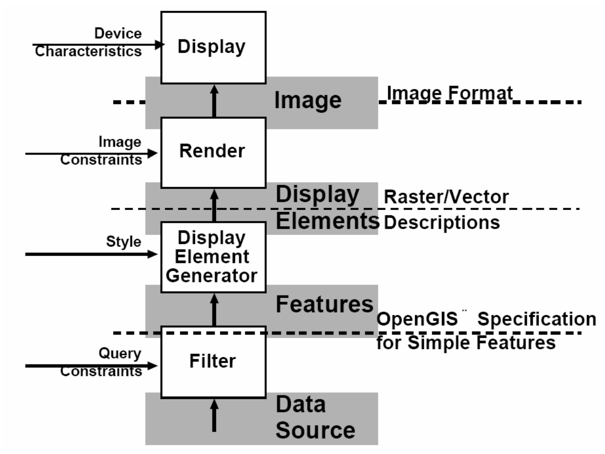


Figure: OGC Portrayal model. Portrayal is the presentation of information to humans, e.g., a map.

For web mapping applications, this portrayal model is implemented as a multi-tired architecture. The filtering or selection of geo-spatial data to be displayed is implemented on the *back-end* or *data storage tier* that consists of the file servers that hold the spatial data (e.g. from shape and image files, as well as from the server running Oracle or other RDBMS, and SDE). The process of selection is initiated from the *middle* or *business tier* that consists of the server-side components, including the web server, application server, spatial server, etc. This tier is where all of the request processing and server administration takes place.

The display of the rendered map to the user is implemented to be on the client's machine. The *client* or *presentation tier* consists of client-side components which are used to send requests to the server and to view the maps and data. The display viewers consist of HTML/DHTML viewers, Java, other standalone application viewers, or XML clients. HTML/DHTML viewers are web browsers that are loaded (e.g. with JavaScript). Java viewers may be standalone programs or they can be implemented in a browser through an applet.

The second and third portrayal process (the generation of display elements from the selected geo-spatial data and he rendering of display elements into a rendered map) can be implemented in the *middle tier* or in the *client tier.* It depends on what type of services (WMS - Web Map Service or WFS – Web Feature Service) are implemented. The WMS corresponds to the thin client and WMS to the thick client architectures.

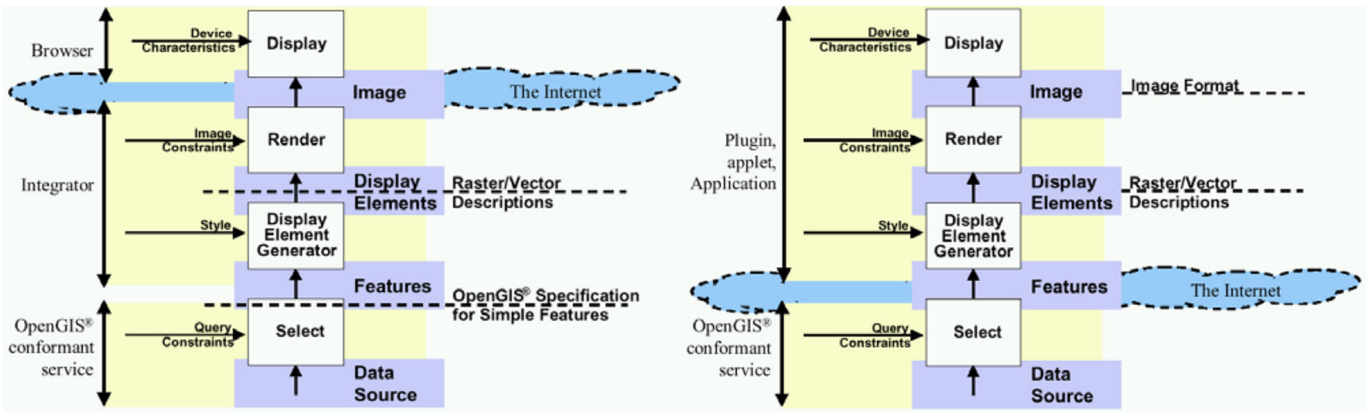


Figure: Thin vs. thick clients for portraying features over the Internet (OGC Reference Model ORM version 0.1.3)

WMS or image service publishes maps as raster images that are already pre-build on the middle tier for display, while a WFS or feature service organizes geographic features or vector data on the middle tier and render them on a client for display. WMS and WFS serve as brokers to map and feature databases. A rendered map is the result from a map server while features are returned from a feature server.

WMS delivers map content to a client application such as JPEG, PNG, or GIF images. A new map image is generated each time a client requests new information. WFS delivers map content to the user as streamed features or as XML/GML coded geometry. Feature geometry is streamed and remains only as long as the client is open. WFS requests are sent to the spatial server only when additional data are needed.

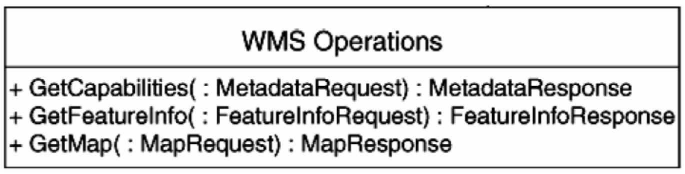
If the map client only has display capabilities, pixel data (raster data) will be transmitted to the client. All standard web browsers have such a capability. If the map client has map-rendering and display element generator capabilities, features are transmitted to the map client from the feature server. Compared to WFS, pixel data transmission has the advantage that it reduces network traffic by transmitting vector geo-data only once, especially for the *zoom in/zoom out* and *pan* operations in the client side. The disadvantage is that the process requirements for the vector data map client are much higher than those for the pixel data map client. A typical vector data map client needs a Java applet, Active-X component, or plug-in to manipulate the vector data.

3.4. WMServices

According to OpenGIS Web Map Server Implementation Specification (2006), a Web Map Service (WMS) produces maps of spatially referenced data dynamically from geographic information. WMS-produced maps are rendered in an ***image*** format such as PNG, GIF or JPEG, or as vector-based graphical elements in Scalable Vector Graphics (SVG) or Web Computer Graphics Metafile (WebCGM) formats.

The Open Geospatial Consortium Specification defines that an OGC web map server implements three functions or *operations*: *GetCapabilities*, *GetMap*, and *GetFeatureInfo*. The GetCapabilities function provides the client with a map server's service metadata, specifying its capabilities. The GetMap function specifies well-defined map request geographic and dimensional parameters that enable the client to request an image map. Finally, the GetFeatureInfo function allows the client to request more information about features at a specific location in the map. GetCapabilities and GetMap are the mandatory operations; GetFeatureInfo is optional.

In a typical OGC WMS client-server interaction, the client firstly requests GetCapabilities from the map server in order to determine what the map server can do and what maps the map server can provide. The client then requests GetMap with the map server's capabilities information in order to get a map image. Finally, the client can request GetFeatureInfo by specifying a point on the map to receive more geographic feature information.



WMS Operations

+ GetCapabilities(: MetadataRequest) : MetadataResponse
+ GetFeatureInfo(: FeatureInfoRequest) : FeatureInfoResponse
+ GetMap(: MapRequest) : MapResponse

Figure: The OGC WMS functions/operations

The WMS host has to support the HTTP protocol within distributed computing platform. HTTP supports two request methods: GET and POST (http://en.wikipedia.org/wiki/HTTP#Request_methods). A WMS server may offer one or both of these methods. Support for the GET method is mandatory; support for the POST method is optional. HTTP GET uses keyword-value pairs (*KVP*) to encode the various operations and parameters within URL as it is shown below. HTTP POST uses XML as the encoding language in the body of the POST document.

The WMS operations can be invoked using a standard web browser by submitting GET requests to the WMS host in the form of HTTP Uniform Resource Locators (URLs). An HTTP URL locates the online resource of each operation supported by a service instance. The content of such URLs depends on which operation is requested. A map request URL indicates the map content, geographic map extent, the desired coordinate reference system, and the output map image size.

Each URL should conform to the description in IETF RFC 2616 (section 3.2.2 "HTTP URL") but is otherwise implementation-dependent; only the query portion comprising the service request itself is defined by OGC WMS specification.

A URL string includes the protocol (“http” or “https”), hostname or IP address, optional port number, path, mandatory question mark “?” (a separator indicating start of query string), and optional string with one or more server-specified parameters ending in an ampersand “&” (Separator between parameters in query string). A URL string is constructed in conformance with IETF RFC 2616. Thus, a URL string defines the network address of a particular server to which request messages are to be sent for a particular operation.

The WMS OGC specification defines how to construct parameters or query part of the URL string in order to form a complete request message. Every WMS operation has several mandatory or optional request parameters. Each parameter is defined by name. Each parameter may have one or more legal values in the form “name=value&”. Values either are defined by the specification or are selected by the client based on service metadata.

Table: URL string for completion of WMS GET request message

URL component	Description
http://host[:port]/path[?{name[=value]&}]	URL prefix of service operation. [ ] denotes 0 or 1 occurrence of an optional part; {} denotes 0 or more occurrences.
name=value&	One or more standard request parameter name/value pairs as defined for each *operation* by the WMS OGC specification.

Upon receiving a valid request, the server sends a valid response as a file that may contain text, or a file that may represent a map image. The appropriate Multipurpose Internet Mail Extensions (MIME) type (IETF RFC 2045) accompanies these responses.

In response to a *GetCapabilities* request, the OGC web map server produces an XML document containing the web map server’s service metadata, describing all the operations it supports, and providing information about available maps. The client application has to parse the XML capabilities document to retrieve the necessary information used to request a map.

With the capabilities information, the client can request a map image from the map server. The *GetMap* request usually includes some necessary parameters such as Layers, Styles, Bounding Box (BBox), Spatial Reference System (SRS), map output Width and Height, and the Format of the image. Layers specify the information to be shown on the map (for example, roads, rivers, towns, and so on). Styles are defined to depict Layers, for example, lines represent roads, circles represent towns, and so on. BBox is a set of four coordinate values (minX, minY, maxX, maxY) indicating a rectangular area on the earth to be mapped. SRS is the projected coordinate system to be used (see the OGC specification for a detailed description at the http://www.opengeospatial.org/standards/wms).

```
http://yourfavoritesite.com/WMS?
VERSION=1.1.0&
REQUEST=GetMap&
BBOX=0.0,0.0,1.0,1.0&
LAYERS=Rivers,Roads,Houses&
STYLES=CenterLine,CenterLine,Outline
```

Figure: The incomplete example of HTTP-GET request for a map in which the appearance for a map portrayal is specified by the Layers and Styles parameters

Two or more maps can be overlaid on the same output map image by using image formats that support transparent backgrounds (e.g. GIF or PNG). These maps can be requested from different distributed map servers.

A basic WMS holds data as “Layers” - the basic unit of geographic information requested from a server and offer a finite number of predefined “Styles” in which to display those layers. A layer or a few layers that portray a digital image suitable for display on a computer screen is called “Map”.

The OGC WMS can support basic coordinate systems for a map and layers. During a portrayal operation, a WMS may convert geographic data from a layer or source data coordinate system into a map coordinate system. A layer may have an associated vertical, temporal, or other coordinate system.

After getting a map, the client can request feature data of a specific point on the map using the *GetFeatureInfo* function. The response from the WMS server will be one of the three output formats: a Geography Markup Language (GML) file, a plain text file, or a Hypertext Markup Language (HTML) file. GML is a type of encoding based on the XML specifying the geographic feature information. Geographic features include points, lines, polygons, and so on, which are defined in the OGC Simple Features Specification (http://www.opengeospatial.org/standards/sfa). The client application parses the GML and displays the feature information in the web browser.

3.5. WFServices

The Open Geospatial Consortium Web Map Service (WMS) allows one to display a map's images from multiple WMS on a client machine; whereas, the OGC Web Feature Service (WFS) allows one to work with ***features*** or vector representations of geospatial data encoded in Geography Markup Language (GML) from multiple WFS.

The OGC WFS specification defines interfaces and their functions or operations for data access and manipulation on geographic features using HTTP through the distributed computing platforms. The OGC WFS supports *insert, update, delete, lock, query,* and *discovery* operations. These operations can be applied for an individual feature: a new feature instance can be created, deleted, updated, and queried based on spatial and non-spatial constraints.

Feature is an abstraction of a real world phenomenon. According the OGC, *feature* is instance that is described by a set of properties where each property can be represented as a ***tuple*** {name, type, value}. The name and type of each feature property is determined by its type definition. A geographic feature has at least one property that is geometry related and associated with a location on the Earth. However, geographic features may not have geometric properties at all. The ISO defines a f*eature* instance to include feature attributes, feature relationships, and feature operations (defined mathematical operations for computing information about features). Individual feature instances are classified into feature types with common characteristics.

The OGC Geography Markup Language (GML) specification describes an encoding of geographic features in XML for the storage, transport, processing, and transformation of geographic data.

Therefore, a WFS requires what their interface be defined in XML. The extension of XML, GML must be used to express features within the interface for feature presentation. SQL filter language has to be used in XML for selections. The source data can only be accessible through the WFS interface and hidden for client applications. A subset of XPath expressions can be used for referencing properties.

WFS can be described by the same way as WMS, by defining a request and response processing from a client to WMS servers and back. Thus, a request can be processed in the following order:

1. At the first, a client application requests a capabilities document that contains a description of all the operations that the WFS supports and a list of all feature types that it can service.
2. On the next step, if required, a client application may request the definition of one or more of the feature or element types that the WFS can service.
3. Based on the definition of the feature type(s), the client application generates a request that is posted to a web server.
4. The WFS is invoked, read, and services the request. For example, WFS can retrieve feature instances with particular feature properties under spatially and/or non-spatially queries.
5. When the WFS request has been completed, a status report is generated by WMS and sent back to the client. In the event that an error has occurred, the status report will indicate that fact.

Similar to WMS, WFS can response to GET (keyword-value pairs) and/or POST/SOAP (XML) HTTP request methods, however, the construction of an URL request to WMS differs in each case. The construction of URL for WFS GET request is similar to that described above for WMS. In addition, web feature services can be offered using HTTPS (http://en.wikipedia.org/wiki/Https). HTTPS is HTTP over a secure communication channel that allows encrypted information to be transferred over the net.

The response to any request encoded using the GET method is identical to the responses generated for requests encoded in XML (the POST method).

The OGC WFS request can support the following operations:

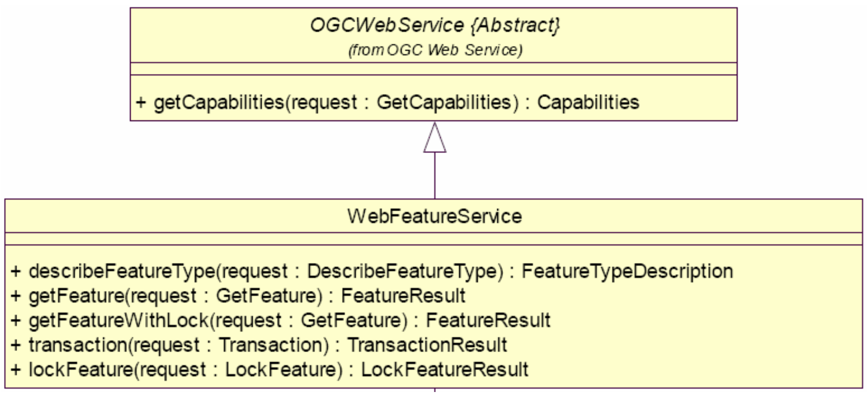

OGCWebService {Abstract}
(from OGC Web Service)
+ getCapabilities(request : GetCapabilities) : Capabilities
WebFeatureService
+ describeFeatureType(request : DescribeFeatureType) : FeatureTypeDescription
+ getFeature(request : GetFeature) : FeatureResult
+ getFeatureWithLock(request : GetFeature) : FeatureResult
+ transaction(request : Transaction) : TransactionResult
+ lockFeature(request : LockFeature) : LockFeatureResult

Figure: A UML diagram of the WFS interface. The WebFeatureService class inherits the *getCapabilities* operation from the abstract OGCWebService class that is common to all OGC Web Services (including WMS). The WebFeatureService class adds the *DescribeFeatureType*, *GetFeature*, *getFeatureWithLock, Transaction*, and *LockFeature* methods.

Let us briefly look on the WFS operations that are *getCapabilities* (mandatory), *DescribeFeatureType* (mandatory), *GetFeature* (mandatory), *GetGmlObject, Transaction – Insert, Update* and *Delete* (optional), and *LockFeature* (optional)*:*

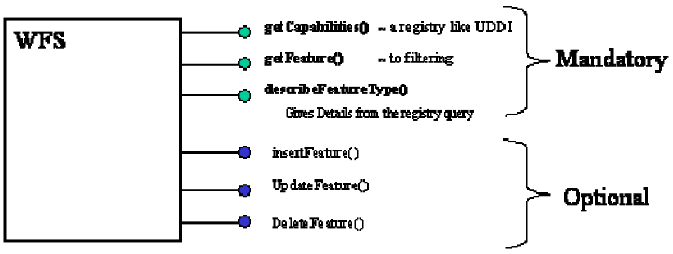

WFS
getCapabilities() -- a registry like UDDI
getFeature() -- to filtering
describeFeatureType()
Gives Details from the registry query
insertFeature()
UpdateFeature()
DeleteFeature()
Mandatory
Optional

Figure: The interface model of the OGC Web Feature Service (from http://en.wikipedia.org/wiki/Web_Feature_Service)

The WFS ***GetCapabilities*** service describes WFS capabilities that indicate available WFS feature types and its supported operations. In fact, this service is returning service metadata and has to support the HTTP GET request method (KVP encoding).

The following fragment defines the sample of the GetCapabilities GET URL request:

```
http://www.someWFSserver.com/wfs.cgi?
VERSION=1.1.0&
SERVICE=WFS&
REQUEST=GetCapabilities
```

Figure: The GET URL request for the capabilities document request from WFS

The capabilities response document contains the following sections:

- *Service Identification* section provides information about the WFS service itself, as defined in the OWS Common Implementation Specification clause 7.4.3 (http://www.opengeospatial.org/standards/common).
- *Service Provider* section provides metadata about the organization operating the WFS server, as defined in the OWS Common Implementation Specification, clause 7.4.4.
- *Operation Metadata* section provides metadata about the WFS operations. The contents of the operation metadata section are defined in the OWS Common Implementation Specification, clause 7.4.5. This metadata includes the parameters and constraints for each operation.
- *FeatureType list* section defines the list of feature types, derived from gml:AbstractFeatureType and operations on each feature type that are available from a web feature service. Additional information, such as the default Spatial Reference System (SRS), any other supported SRSs, or no SRS whatsoever (for non-spatial feature types), for WFS requests is provided for each feature type.
- *ServesGMLObjectType list* section lists GML Object types, which are not derived from gml:AbstractFeatureType but are available from a web feature service that supports the GetGMLObject operation. These types may be defined in a base GML schema or in an application schema using its own namespace.
- *SupportsGMLObjectType list* section defines the list of GML Object types that a WFS server would be capable of serving if it was deployed to serve data as

described by an application schema that either used those GML Object types directly (for non-abstract types) or defined derived types based on those types.

- *Filter capabilities* optional section that defines the list of WMS supported filter operations. If the section is not defined, then the server only supports the minimum default set of filter operators as defined in the OGC Filter Encoding Implementation Specification (http://www.opengeospatial.org/standards/filter). The schema of the section is also defined in the Filter Encoding Implementation Specification.

Detailed descriptions of response elements for this and other WFS operations could be found in the OGC WMS specification as well as detailed descriptions of the GET URL WFS request parameters using standard CGI style keyword-value pair encoding for the WFS operations are also defined at the OpenGIS Web Feature Service (WFS) Implementation Specification (http://portal.opengeospatial.org/files/?artifact_id=8339).

The WFS ***DescribeFeatureType*** service describes the structure of any feature type that the WFS can service. The operation generates a schema that defines how a WFS implementation expects feature instances to be encoded on input (via Insert and Update requests) and how feature instances will be generated on output (in response to GetFeature and GetGmlObject requests).

The following fragment defines the sample of the DescribeFeatureType GET URL request:

```
http://www.someserver.com/wfs.cgi?
SERVICE=WFS&
VERSION=1.1.0&
REQUEST=DescribeFeatureType&
TYPENAME=Buildings_100K
```

Figure: The GET URL request of schema description of the `Buildings_100K` feature type

The only mandatory output is a GML3 application schema, defined using XML Schema, and should be generated. Other schema languages may also be used to describe feature types as long as the MIME type value for the outputFormat attribute is advertised in the capabilities document.

In response to the above DescribeFeatureType request, a web feature service might generate an XML Schema document that looks like:

```
<?xml version="1.0" ?>
<xsd:schema
   targetNamespace="http://www.someserver.com/myns"
   xmlns:myns="http://www.someserver.com/myns"
   xmlns:gml="http://www.opengis.net/gml"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns="http://www.w3.org/2001/XMLSchema"
   elementFormDefault="qualified" version="0.1">

   <xsd:import
   namespace="http://www.opengis.net/gml"schemaLocation="../gml/3.1.1/base/gml.xsd"/>

    <!-- ============================================
    define global elements
    ============================================ -->
    <xsd:element name="Buildings_100K"
    type="myns:Buildings_100K_Type"
    substitutionGroup="gml:_Feature"/>

    <!-- ===========================================
    define complex types (classes)
    =========================================== -->
    <xsd:complexType name="Buildings_100K_Type">
       <xsd:complexContent>
             <xsd:extension base="gml:AbstractFeatureType">
                   <xsd:sequence>
                         <xsd:element name="shpGeom"
                               type="gml:PolygonPropertyType" nillable="false"/>
                         <xsd:element name="id" nillable="true" minOccurs="0">
                               <simpleType>
                                     <xsd:restriction base="integer">
                                           <xsd:totalDigits value="10"/>
                                     <xsd:/restriction>
                               <xsd:/simpleType>
                         <xsd:/element>
                         <xsd:element     name="mailAddress"     type="myns:AddressType"
                         nillable="true">
                   <xsd:/sequence>
             <xsd:/extension>
       <xsd:/complexContent>
       <xsd:/complexType>
       <xsd:complexType name="AddressType">
          <xsd:sequence>
                <xsd:element name="streetName" nillable="true">
                      <xsd:simpleType>
                            <xsd:restriction base="string">
                                  <xsd:maxLength value="50"/>
                            </xsd:restriction>
                      </xsd:simpleType>
                </xsd:element>
                <xsd:element name="streetNumber" nillable="true">
                      <xsd:simpleType>
                            <xsd:restriction base="string">
                                  <xsd:maxLength value="10"/>
                            </xsd:restriction>
                      </xsd:simpleType>
                </xsd:element>
                <xsd:element name="city" nillable="true">
                      <xsd:simpleType>
                            <xsd:restriction base="string">
                                  <xsd:maxLength value="30"/>
                            </xsd:restriction>
                      </xsd:simpleType>
                </xsd:element>
                <xsd:element name="municipality" nillable="true">
                      <xsd:simpleType>
                            <xsd:restriction base="string">
                                  <xsd:maxLength value="30"/>
                            </xsd:restriction>
                      </xsd:simpleType>
                </xsd:element>
          </xsd:sequence>
   </xsd:complexType>
</xsd:schema>
```

Figure: A sample of XML Schema document (xsd) that is generated on the DescribeFeatureType request

The WFS ***GetFeature*** service retrieves feature instances with specified feature properties. GML is used for the canonical representation of features. For this reason, one has to understand GML models for features that will be discussed in the next module.

In GML a feature is represented as an XML `element`. The name of the feature element indicates the feature type (e.g. `Buildings_100K` or `gml:Buildings_100K)` and conventionally given in UpperTitleCase. A feature element can contain a set of child elements that describes a set of the feature properties. The name of the property element indicates the property type, for example `streetName` or `gml:boundedBy`, and conventionally given in lowerCamelCase.

The value of a property can be given in-line, for example `<xsd:element name="location" type="gml:PointPropertyType" nillable="true"/>`, or by-reference as the value of a resource identified in a link carried as an XML attribute of the property element. If an in-line form is used, then the content may be structured using XML elements. For example:

```
<xsd:element name="municipality" nillable="true">
      <xsd:simpleType>
            <xsd:restriction base="string">
                  <xsd:maxLength value="30"/>
            </xsd:restriction>
      </xsd:simpleType>
</xsd:element>
```

The following fragments define the sample for a GetFeature GET URL request:

```
http://www.someserver.com/wfs.cgi&
SERVICE=WFS&
VERSION=1.1.0&
REQUEST=GetFeature&
TYPENAME=Buildings_100K,Roads_100K
```

Figure: The GET URL request of query all properties of all `Buildings_100K` and `Roads_100K` instances

The mandatory `typeName` attribute is used to indicate the name of one or more feature type classes to be queried. Their values have to be in the list of namespace-qualified names (XML Schema type QName, e.g. `myns:AddressType`) and match one of the feature types from the capabilities document of the WFS. Specifying more than one typename indicates that a join operation is being performed.

```
http://www.someserver.com/wfs.cgi&
SERVICE=WFS&
VERSION=1.1.0&
REQUEST=GetFeature&
PROPERTYNAME=Buildings_100K/buildingId&
TYPENAME= Buildings_100K
```

Figure: The GET URL request for query `buildingId` property of all `Buildings_100K` type instances

The value of each `wfs:PropertyName` element, e.g. `buildingId`, has to be a namespace-qualified name (XML Schema type QName, e.g. `myns:Buildings_100K`) and matches the name of one of the property elements in the GML representation of the relevant feature. The names of the property elements may be obtained as the result of a DescribeFeatureType request.

```
http://www.someserver.com/wfs.cgi&
SERVICE=WFS&
VERSION=1.1.0&
REQUEST=GetFeature&
PROPERTYNAME= Buildings_100K/buildingId&
FEATUREID= Buildings_100K.1366
```

Figure: The GET URL request for query `buildingId` property of `Buildings_100K` type instance with `1366` identifier

```
http://www.someserver.com/wfs.cgi&
SERVICE=WFS&
VERSION=1.1.0&
REQUEST=GetFeature&
TYPENAME=InWaterA_1M&
FILTER=<Filter><Within><PropertyName>Buildings_100K/wkbGeom<PropertyName>
<gml:Envelope><gml:lowerCorner>80 80<gml:lowerCorner>
<gml:upperCorner>200 200</gml:upperCorner></gml:Envelope>
</Within></Filter>
```

Figure: The GET URL request for query all property of `Buildings_100K` type instance with *spatial* filter. An XML encoded filter parameter value is specified in the OGC Filter Encoding Implementation Specification.

A WMS GetFeature may contain one or more `Query` descriptions that are concatenated to produce the result query set. The `Filter` element is used to define constraints on a query. Both spatial and/or non-spatial constraints can be specified as described in the Filter Encoding Specification.

If a web feature service implements Xlink traversal, a WFS client can use the traverseXlinkDepth and traverseXlinkExpiry attributes to request that nested property XLink linking element locator attribute (href) XLinks are traversed and resolved if possible. The XLink linking element is a part of the XML Path Language (XPath) specification that is addressing parts of a XML document is discussed in the previous Module. WFS implementation must support only the subset of the XPath language that predefined in the OGC WMS specification.

If WMS GetFeature `outputFormat` attribute is set up to `text/gml; subtype=gml/3.1.1` (that is the default value) then the client will receive a response from WMS GetFeature request in the form of a validated GML3 document.

In response to a GetFeature request, a WFS can either generate a complete response document or it may simply return a count of the number of features. The optional `resultType` attribute is used to control the type of WFS respond.

The following example illustrates a GetFeature request a instance with `1366` identifier using `Towns_100K/townId` as `typeName/PropertyName` elements. The request is:

```
<wfs:GetFeature
      xmlns:wfs="http://www.opengis.net/wfs"
      xmlns:ogc="http://www.opengis.net/ogc"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:myns="http://www.someserver.com/myns"
      xsi:schemaLocation="http://www.opengis.net/wfs ../../wfs/1.1.0/WFS.xsd"
      version="1.1.0"
      service="WFS"
      outputFormat="text/xml; subtype=gml/3.1.1">
      <Query typeName="myns:Towns_100K">
            <wfs:PropertyName>myns:townId</wfs:PropertyName>
            <wfs:PropertyName>gml:directedNode</wfs:PropertyName>
            <ogc:Filter>
                  <ogc:GmlObjectId myns:townId="1366"/>
            </ogc:Filter>
      </myns:Query>
</wfs:GetFeature>
```

Figure: The POST XML request for query `townId` property of `Towns_100K` type instance with `1366` identifier

The response contains an `xlink:href`, but it is returned as:

```
<?xml version="1.0" ?>
<wfs:FeatureCollection
      Xmlns:myns="http://www.someserver.com/myns"
      xmlns:wfs="http://www.opengis.net/wfs"
      xmlns:gml="http://www.opengis.net/gml"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://www.someserver.com/myns PopulatedPlaces.xsd
      http://www.opengis.net/wfs ../wfs/1.1.0/WFS.xsd">
      <gml:boundedBy>
            <gml:Envelope srsName="http://www.opengis.net/gml/srs/epsg.xml#63266405">
                  <gml:lowerCorner>90 100</gml:lowerCorner>
                  <gml:upperCorner>190,200</gml:upperCorner>
            </gml:Envelope>
      </gml:boundedBy>
      <gml:featureMember>
            <Towns_100K myns:townId="1366">
                  <myns:name>Vilnius</myns:name>
                  <gml:directedNode orientation="+" xlink:href="#n1"/>
            </Town>
      </gml:featureMember>
</wfs:FeatureCollection>
```

Figure: The POST GML respond for query `townId` property of `Towns_100K` type instance with `1366` identifier

The optional WMS ***GetGmlObject*** operation allows retrieval of features and elements by ID from a web feature service. An XML document fragment is returning as a response to this operation to the client. The WFS GetGmlObject service can process a request to retrieve element instances by traversing XLinks that refer to their XML ID.

The GetGmlObject requests WFS to return an element with a `gml:id` attribute value specified within an `ogc:GmlObjectId` element. The value of the `gml:id` attribute is used as a unique key to retrieve the complex element with the `gml:id` attribute value. The requested element could be any identified element of the GML data being served by a WFS, such as a feature, a geometry, a topology, or a complex attribute.

Every feature instance in a WFS implementation has to be uniquely identifiable. A feature identifier is unique to the server and can be used to reference the same feature instance. A feature identifier is encoded as described in the OGC Filter Encoding specification. A feature identifier can be used wherever a feature reference is required (e.g. for the GetGmlObject operation).

The WFS ***Transaction*** services execute operations that modify features: create, update, and delete feature instances. WFS will generate an XML response document indicating the completion status of the transaction.

A `Transaction` element may contain zero or more `Insert`, `Update`, or `Delete` elements that describe operations to create, modify, or destroy feature instances. A WFS processes `Insert`, `Update`, and `Delete` elements in the order in which they appear in the transaction request.

The `Insert` element is used to create new feature instances. By default, the initial state of a created feature is expressed by GML3 and must be validated against a GML3 application schema generated by the DescribeFeatureType operation.

The `Update` element describes one update operation that is applied to a feature or set of features of a single feature type. Multiple `Update` operations can be contained in a single Transaction request.

The `Delete` element is used to delete one or more feature instances. The following fragment defines the sample of the Transaction `Delete` GET URL request:

```
http://www.someserver.com/wfs.cgi?
SERVICE=WFS&
VERSION=1.1.0&
REQUEST=Transaction&
OPERATION=Delete&
TYPENAME=Buildings_100K&
BBOX=10,10,20,20
```

Figure: The GET URL request for the WMS transaction that deletes all `Buildings_100K` type features inside the specified `BBOX` box

Expressing lengthy INSERT or UPDATE requests is not convenient using keyword-value pair encoding.

The WFS ***LockFeature*** service locks on one or more feature instances for the duration of a transaction in order to support serializable transactions.

The following fragments define the sample for the LockFeature GET URL requests:

```
http://www.someserver.com/wfs.cgi?
SERVICE=WFS&
VERSION=1.1.0&
REQUEST=LockFeature&
TYPENAME=Buildings_100K
```

Figure: The GET URL request for locking of all instances of the `Buildings_100K` feature type

3.6. Web Coverage Service

Coverage is **a** feature that acts as a function to return values from its range for any direct position within its spatiotemporal domain (http://www.opengeospatial.org/standards/wcs). A coverage represents continues phenomena, as vector or individual features represent discrete objects in the form of a point, line or polygon. There are different types of coverage – a set of tiled polygons, a grid of values, a mathematical function, or a combination of these (http://www.opengeospatial.org/standards/gc).

Grid coverage represents the value (scalar or vector) in a grid's points. The value returned by the coverage for any point is that of the grid point's value for a location nearest the grid's point. Often grid coverages domain is comprised from regularly spaced locations along 0, 1, 2, or 3 axes of a spatial coordinate reference system. Value in a grid point can be scalar, such as elevation (one band) or vector, such as brightness values in different parts of the electromagnetic spectrum (multiple bands).

The OGC Web Coverage Service (WCS) specification describes how electronic retrieval of geospatial data, as multi-valued "coverages", is supported over the net. The current version of OGC WCS is limited to describing grid coverage only.

Like OGC WMS and WFS, WCS allows clients to retrieve portions of a server's spatial data based on spatial constraints. However, unlike the WMS, which portrays spatial data as a map's image, the WFS returns geospatial data together with their properties. This data, like that for the WFS, can be processed and not just portrayed. Unlike WFS that returns discrete geospatial features, the WCS returns multidimensional coverages representing space-varying phenomena.

The WCS provides three operations: *GetCapabilities*, *DescribeCoverage,* and *GetCoverage*. The *GetCapabilities* operation returns an XML document describing the service and brief descriptions of the coverages that clients may request. The *DescribeCoverage* operation lets clients request a full description of one or more coverages served by a particular WCS server. The *GetCoverage* operation returns a coverage, encoded in a well-known coverage format. The *GetCoverage* syntax and semantics have some resemblance to the WMS *GetMap* and WFS *GetFeature* requests, but several extensions support the retrieval of coverages rather than static maps or discrete features. The *GetCoverage* operation is normally run after *GetCapabilities* and *DescribeCoverage* operation responses have shown what requests are allowed and what data are available.

3.7. Filter Encoding

The OGC Filter Encoding implementation specification (http://www.opengeospatial.org/standards/filter) was originally part of version 0.0.10 of the Web Feature Server (WFS) specification. It was put into a separate document because the filter encoding can be used by a broad range of services (WFS, WCS, Gazetteer, Web Registries) that require the ability to express a query's constraint in XML.

A *filter* is used to define a set of feature instances that are to be operated upon. The operating set can be comprised of one or more enumerated features or a set of features defined by specifying spatial and non-spatial constraints on the geometric and scalar properties of a feature type. A *filter expression* uses property values of an object type to build constraints.

The OGC Filter Encoding specification describes an XML encoding of the OGC Common Catalog Query Language (CQL) as a system of a query predicate. Using the XML tools, an XML filter can be validated, parsed, and then transformed into other query language (e.g., a WHERE clause for a SQL SELECT statement or XPath expression for fetching data from XML documents) that is required to retrieve or modify object instances in databases.

This specification defines the expression within a root element `<Filter>`. The `<Filter>` element is defined by the following XML Schema fragment:

```
<xsd:element name="Filter" type="ogc:FilterType"/>
      <xsd:complexType name="FilterType">
      <xsd:choice>
            <xsd:element ref="ogc:spatialOps"/>
            <xsd:element ref="ogc:comparisonOps"/>
            <xsd:element ref="ogc:logicOps"/>
            <xsd:element ref="ogc:_Id" maxOccurs="unbounded"/>
      </xsd:choice>
</xsd:complexType>
```

Figure: The OGC XSD schema for `<Filter>` element

The elements `logicOps`, `comparisonOps`, and `spatialOps` are substitution groups for logical, spatial, and comparison operators. A spatial operator uses a spatial relationship to determine geometric arguments that satisfy the topological conditions.

This OGC specification also defines the encoding of fundamental arithmetic operations and functions.

The following fragments define a filter using the `IsBetween` operator. The filter identifies all features where the `FLOOR` is between 2 and 5:

```
<Filter>
  <PropertyIsBetween>
        <PropertyName>FLOOR</PropertyName>
        <LowerBoundary><Literal>2</Literal></LowerBoundary>
        <UpperBoundary><Literal>5</Literal></UpperBoundary>
  </PropertyIsBetween>
</Filter>
```

Figure: Comparison OGC filter operator

3.8. Styled Layer Descriptor and Symbology Encoding

The WMS produces a map's image as an output on a user request from the client machine. Users may require visualizing or portraying feature data from a server in the map's image in user-defined symbology style. The new OGC Styled Layer Descriptor Profile of the Web Map Service Implementation Specification (http://www.opengeospatial.org/standards/sld) and Symbology Encoding Implementation Specification (http://www.opengeospatial.org/standards/symbol) extend the OGC WMS specification in order to allow user-defined symbolization of feature and coverage data.

The current OGC WMS specification supports the ability to specify very basic styling options by using a basic collection of available visual styles for data set (see example above with `STYLES=CenterLine,CenterLine,Outline` parameters, WMS uses predefine basic style for each style code). The user cannot define individual styling rules. This requires a styling language that the client and server can both understand. The OGC defines such language and it is called ***Symbology Encoding*** (SE). The SE language is defined within the Symbology Encoding Implementation Specification. This language can be used not only for the WMS, but also for portrayal output of WFS and WCS.

There are different methods or techniques of cartographic symbolization (see GII-06 Module 6). In addition to the SE, the OGC offers the new operations for OGC services within the OGC ***Styled Layer Descriptor*** (SLD) profile specification. These operations support some cartographic visualization techniques. Thus, the *FeatureTypeStyle* element within the SE document can be used to apply the same symbology style to all features within one feature type or layer. This approach corresponds to *single point symbol* techniques in thematic cartography. For example, all linear and polygonal features of hydrography (rivers, streams, lakes, ponds, oceans, etc.) layer can be visualized in a light blue for polygon fills, and in darker blue for all polygon outlines and all lines.

More comprehensive qualitative or quantitative cartographic symbolization techniques use values of feature attributes for symbolization. For example, different classes of roads can be shown in different symbology based on attribute values from the field Class. A SLD profile offers an optional operation *DescribeLayer* that returns the feature types of the layer or layers specified in the request, and the attributes can be discovered with the *DescribeFeatureType* operation of a WFS interface or the *DescribeCoverageType* of a WCS interface. This is a mechanism by which a client can obtain feature/coverage-type information for a WMS named layer.

The last figure in the WMServices section shows an example of style parameters for three layers `LAYERS=Rivers,Roads,Houses` by using HTTP GET request. The SLD profile for WMS defines additional parameters allowing clients to GET request layers to be portrayed according to some specified style. SLD introduces four additional parameters that can be used in a GetMap GET request.

The same styling can be described by using a XML Styled-Layer Descriptor (SLD) document as shown in the following figure. A ***Styled-Layer Descriptor*** document describes a map's appearance using a user-defined XML encoding. A SLD document includes a *StyledLayerDescriptor* XML element that contains a sequence of styled-layer definitions. These styled-layer definitions may use WMS named or user-defined layers and styling. An SLD document is defined as a sequence of styled layers.

```
<StyledLayerDescriptor version="1.0.0">
      <NamedLayer>
            <Name>Rivers</Name>
            <NamedStyle>
                  <Name>CenterLine</Name>
            </NamedStyle>
      </NamedLayer>
      <NamedLayer>
            <Name>Roads</Name>
            <NamedStyle>
                  <Name>CenterLine</Name>
            </NamedStyle>
      </NamedLayer>
            <NamedLayer>
            <Name>Houses</Name>
            <NamedStyle>
                  <Name>Outline</Name>
            </NamedStyle>
      </NamedLayer>
</StyledLayerDescriptor>
```

Figure: Sample of Styled-Layer Descriptor

An SLD XML document can become complex with user-defined styling.

The OGC offers three approaches to use SLD code from WMS documents:

- WMS HTTP GET request can include reference to a remote SLD XML file
- The SLD XML document can be included within HTTP GET request
- The SLD XML document can be embedded in the WMS HTTP POST GetMap request

The following fragment defines the sample of the first approach:

```
http://yourfavoritesite.com/WMS?
      VERSION=1.0.5&
      REQUEST=GetMap&
      SRS=EPSG%3A4326&
      BBOX=0.0,0.0,1.0,1.0&
      SLD=http%3A%2F%2Fmyclientsite.com%2FmySLD.xml&
      WIDTH=400&
      HEIGHT=400&
      FORMAT=PNG
```

Figure: The SLD is identified using a URL. The URL for the prepared SLD document is http://myclientsite.com/mySLD.xml

Symbology Encoding (SE) language defines some basic elements used both by SE itself, but also within SLD. SE language defines elements used for rendering features and coverages. The root element of a Symbology Encoding is therefore a *FeatureTypeStyle* or *CoverageStyle*. The *FeatureTypeStyle* defines the styling that is to be applied to a single feature type. The *CoverageStyle* defines the styling that is to be applied to a subset of Coverage data.

Read more about the SLD and WMS-SLD operations, WMS SLD named and user-defined layers' and styles' definitions, format of a SE map-styling language for producing georeferenced maps with user-defined styling, SLD symbolizers types (that describes how a feature is to appear on a map),

map legend and scale design with *GetLegendGraphic* operation at the http://www.opengeospatial.org/standards/sld and http://www.opengeospatial.org/standards/symbol.

3.9. Conclusion

Web map services are implemented on a distributed computing platform (DCP). There many technologies, such as J2EE, .NET, Common Gateway Interface (CGI), Active Server Pages (ASP), JavaServer Pages (JSP), AJAX etc., are used to develop web map applications. These technologies are implemented within multi-tiered architecture. Map engines can be implemented on different tiers of this architecture.

Four processes of the geo-spatial data presentation (Cuthbert, 1997) can be implemented within different layers of multi-tiered architecture. Thus, for Web Mapping Server, the client is a standard web browser that can send GetMap requests and display responses in the form of map images. Map engines reside on the server side of multi-tiered architecture.

A Web Feature Server works with the vector data in the format described by Geographic Markup Language (GML). WFS servers only select and send discrete feature data to a client in response to a client's query. The client has to be able to display and process the spatial data that currently standard web browsers cannot do. A thick client has to be implemented that can, for example, display, create, modify, and delete features in the feature server through the HTTP protocol. A Web Coverage Server, similar to WFS, can serve grid or raster data through the web.

To solve geo-service interpretability problems among different web mapping systems, the OpenGIS Consortium (OGC) proposed standard specifications, including geo-data presentation and geo-service interfaces. To standardize the geo-data format, Simple Feature (SF) (OGC-SF, 1998) and Grid/Coverage (GC) (OGC-GC, 2001) implementation specifications are proposed for vector data and raster data that were updated to new respective specifications. To standardize the geo-services interaction, Web Mapping Server (WMS) (OGC-WMS, 2006 current version) and Web Feature Server (WFS) (OGC-WFS, 2006 current version) implementation specifications were proposed for map (map's image) presentation and features (vector data) manipulation via the Web.

The OGS also offers related specification for filtering (selection) and portrayal features within WMS, WFS, and WCS. There are two ways to symbolize features through OGS web services: by using LAYERS and STYLES parameters with arguments or by a Styled-Layer Descriptor (SLD) document. Symbology Encoding (SE) language is proposed by OGC to define a SLD document for WMS predefined styling or user-defined style.

Module self-study questions:

1. What are the main outputs for WMS and WFS?
2. What is the basic difference between WMS and WFS?
3. What is a fundamental difference between WMS and WCS?
4. What is a main difference between WFS and WCS?
5. What is a vector feature and what is coverage? What is the main difference between feature and coverage?
6. What is a filter within a web services context? What three types of filters are supported by OGC?
7. What is the difference between SLD and SE?

Recommended Readings:

[1] The OpenGIS Web Map Server Cookbook, http://www.opengeospatial.org/resource/cookbooks
[2] OpenGIS Web Map Service (WMS) Implementation Specification, http://www.opengeospatial.org/standards/wms
[3] OpenGIS Web Feature Service (WFS) Implementation Specification, http://www.opengeospatial.org/standards/wfs
[4] Section 3, System Design Strategies An ESRI ® Technical Reference Document, 2007, http://www.esri.com/library/whitepapers/pdfs/sysdesig.pdf

References

[1] Zhang, S., and Goddard, S., OpenGIS Conforming Map-Feature Server Implementation Specifications in Component-Based Distributed Systems, *Proceedings of the 2003 International Society for Environmental Information Sciences Conference*, Regina, Canada, July 2003, http://www.cse.unl.edu/~goddard/Papers/Conference/ISEIS03ZhangGoddard.pdf

[2] OpenGIS Web Map Service (WMS) Implementation Specification, http://www.opengeospatial.org/standards/wms

[3] OpenGIS Web Feature Service (WFS) Implementation Specification, http://www.opengeospatial.org/standards/wfs

[4] OpenGIS® Web Coverage Service (WCS) Implementation Specification, http://www.opengeospatial.org/standards/wcs

[5] OpenGIS® Filter Encoding Implementation Specification, http://www.opengeospatial.org/standards/filter

[6] OpenGIS® Styled Layer Descriptor Profile of the Web Map Service Implementation Specification, http://www.opengeospatial.org/standards/sld

[7] OpenGIS Symbology Encoding Implementation Specification, http://www.opengeospatial.org/standards/symbol

Terms used

- Client-server architecture
- Multi-tiered architecture
- Peer-to-peer
- HTTP/HTTPS protocol
- AJAX
- HTTP GET and keyword-value pairs
- HTTP POST
- WMServices
- WFServices
- Web Coverage Service
- Filter Encoding
- Symbology Encoding
- Styled Layer Descriptor

4. XML-based Languages for Mapping and SOA

The Extensible Markup Language (XML) is a general-purpose markup language. By adding semantic constraints, application languages can be implemented in XML. These include GML (Geographic Markup Language), ArcXML (ESRI Arc Extensible Markup Language), KLM (Google Keyhole Markup Language), SVG (Scalable Vector Graphics), GeoRSS (Geographic Really Simple Syndication) and many others. Moreover, XML is sometimes used as the specification language for such application languages.

This module discusses a few XML application grammas that are used for spatial data inter-exchange encoding; web mapping service communication and map definition; consuming and mashuping web services within Service Oriented Architecture; and, vector graphic representation on the client. These XML application schemas could be used for data modeling and inter-exchange within different public and commercial applications such as cadastral and land use, traffic and transportation, telecommunication, environment and many others. In addition, topics related to Service Oriented Architecture, web map services, and related technologies are discussed.

Module Outline

Topic 1: XML-based Languages
Topic 2: Geography Markup Language (GML)
Topic 3: ESRI Arc Extensible Markup Language (ArcXML)
Topic 4: Google Keyhole Markup Language (KML)
Topic 5: Scalable Vector Graphics (SVG)
Topic 6: Web Map Services and Service Oriented Architecture (SOA)

4.1. XML-based Languages

As discussed in a previous module, the Extensible Markup Language (XML) allows its users to define their own tags and store data in a human-readable format. Therefore, XML is used as the specification language for many application languages. By adding semantic constraints, application languages can be implemented in XML. These include Chemical Markup Language (CML), Systems Biology Markup Language (SBML), Really Simple Syndication (RSS), GML (Geographic Markup Language), ArcXML (ESRI Arc Extensible Markup Language), KLM (Google Keyhole Markup Language), and many others. One can also note the XML Revolution in the field of data and data schemas encoding.

XML is also a language that is recommended by the International Organization for Standardization (ISO, http://www.isotc211.org/) and the Open Geospatial Consortium (OGC, http://www.opengeospatial.org) for encoding geographic information schemas, data, and web services. OGC XML request and response documents, specified by OGC XML Schema, are currently being used for geospatial web services (see Module 3 for details).

Thus, *ISO 19118:2005 Geographic information - Encoding* standard specifies the requirements for defining encoding rules that are used for the interchange of geographic data within the ISO 19100 series of standards. An encoding rule allows geographic information defined by application schemas and standardized schemas to be coded into a system independent data structure suitable for transport and storage. The XML-based encoding rule is intended to be used for neutral data interchange and relies on the XML and the ISO/IEC 10646 character set standards (see Module 4 of GII-03 for more details).

However, the OGC "GML" community has not accepted ISO 19118 XML encoding for geo-spatial data. A new ISO/TC211 work item proposal has been accepted to create a new harmonized version of encoding, namely, Geography Markup Language (GML) and ISO 19136.

In this module, the Geography Markup Language, recommended as a standard by ISO and OGC for spatial data encoding and exchange, along with two other industrially used XML-based languages, ArcXML and KML, will be discussed.

4.2. Geography Markup Language (GML)

The Geography Markup Language ***encoding*** can be used for the ***transport*** and ***storage*** of geographic information, including geometry (features in vector format) and properties that are spatial and non-spatial properties of objects. The GML can also be used as a ***modeling*** language for geographic information and design for feature web-based services.

Note *that GML encodes geographic* ***content*** *– it does not style or data display*. The XML Transformation Language (XSLT) handles XML styling or symbolization.

Initially GML was developed as an OpenGIS Implementation Specification. Later GML became a work item of ISO/TC 211, first published as ISO 19136 - Geographic information - Geography Markup Language (GML) standard in 2007. GML is based on XML technologies (W3C) and supports XML, XML Namespaces, XML Schema, and Xlinks. GML is extensible and supports the definition of profiles (proper subsets) of the full GML capabilities. GML's Version 3.2.1 is now available (as of January 2008).

Main GML designations are:

- Supports the description of geospatial *application schemas* for *information communities*
- Enables the creation and maintenance of *linked* geographic application schemas and datasets
- Supports the *transport* and *storage* of application schemas and data sets
- Increases the ability of organizations to *share* geographic application schemas and the information they describe
- Leaves it to implementers to decide whether application schemas and datasets are stored in native GML or whether GML is used only for schema and data transport
- Provides GML Schemas that can serve as common constructs and concepts which may be used by all the different application domains
- Defines the base for other standard technologies and protocols that are fostered by OGC. Thus, one important technology is WFS (Web Feature Service), which provides a mechanism for interaction with a geospatial database using GML
- Provides a "design pattern" that is one of the most significant additions to design process on XML schema design.

In accordance with ISO 19136:2007 - Geography Markup Language standard - GML is an XML grammar written in XML Schema for the transport and storage of geographic information and the modeling of user-defined application schemas. GML is an XML encoding language that complies with ISO 19118 for the transport and storage of geographic information modeled according to the conceptual modeling framework, also used in the ISO 19100 series (geomatics standards) and the OpenGIS Abstract Specification. GML can handle spatial and non-spatial properties of geographic features.

The key concept of the ISO 19100 series, and consequently used by GML, is to model the world by using features that are an abstraction of real world phenomena (e.g., a road, a river, a person, a vehicle, an administrative area, an event, etc.). These features can be geographic or associated with locations on the Earth. The state of a feature is defined by a set of properties, where each property may be thought of as a triple {name, type, value}. A feature may have a number of properties that includes geometry. Features can be grouped into collections that can also have feature types.

The key idea of ISO 19100 series is to model a spatial world from conceptual logic into a particular conceptual application schema, where the structure and data content are defined in UML (Unified Modeling Language). Further, this UML application model can be encoded into GML for inter-exchangeable data transfer and storage. The ISO 19109 - Rules for Application Schema standard defines the rules on how to create an application schema for particular features, or set of features, in compliance with other ISO geomatics standards. XML Schema, offered by ISO 19136 in compliance with ISO 19118, must be used to map the feature application model from UML to GML. Alternatively, an application schema can be constructed directly by adhering to the rules for GML in XML Schema.

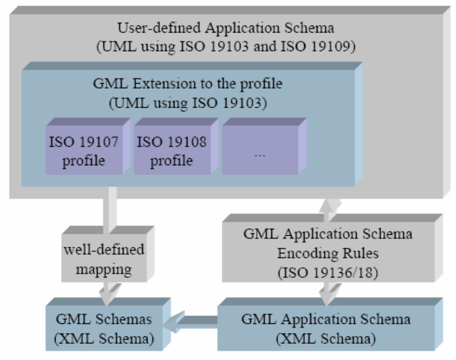

User-defined Application Schema
(UML using ISO 19103 and ISO 19109)
GML Extension to the profile
(UML using ISO 19103)
ISO 19107 profile
ISO 19108 profile
...
well-defined mapping
GML Application Schema Encoding Rules (ISO 19136/18)
GML Schemas (XML Schema)
GML Application Schema (XML Schema)

Figure: Mapping from UML to XML Schema and relationship between the ISO 19100 series of International Standards (ISO 19136)

The mapping from an ISO 19109 conformant UML application schema to the corresponding GML application schema is based on a set of encoding rules. The schema encoding rules are based on the general idea that the class definitions in the application schema are mapped to type and element declarations in XML Schema, so that the objects in the instance model can be mapped to corresponding element structures in the XML document.

The following figure describes the generation of application schemas in a pure GML environment without using UML:

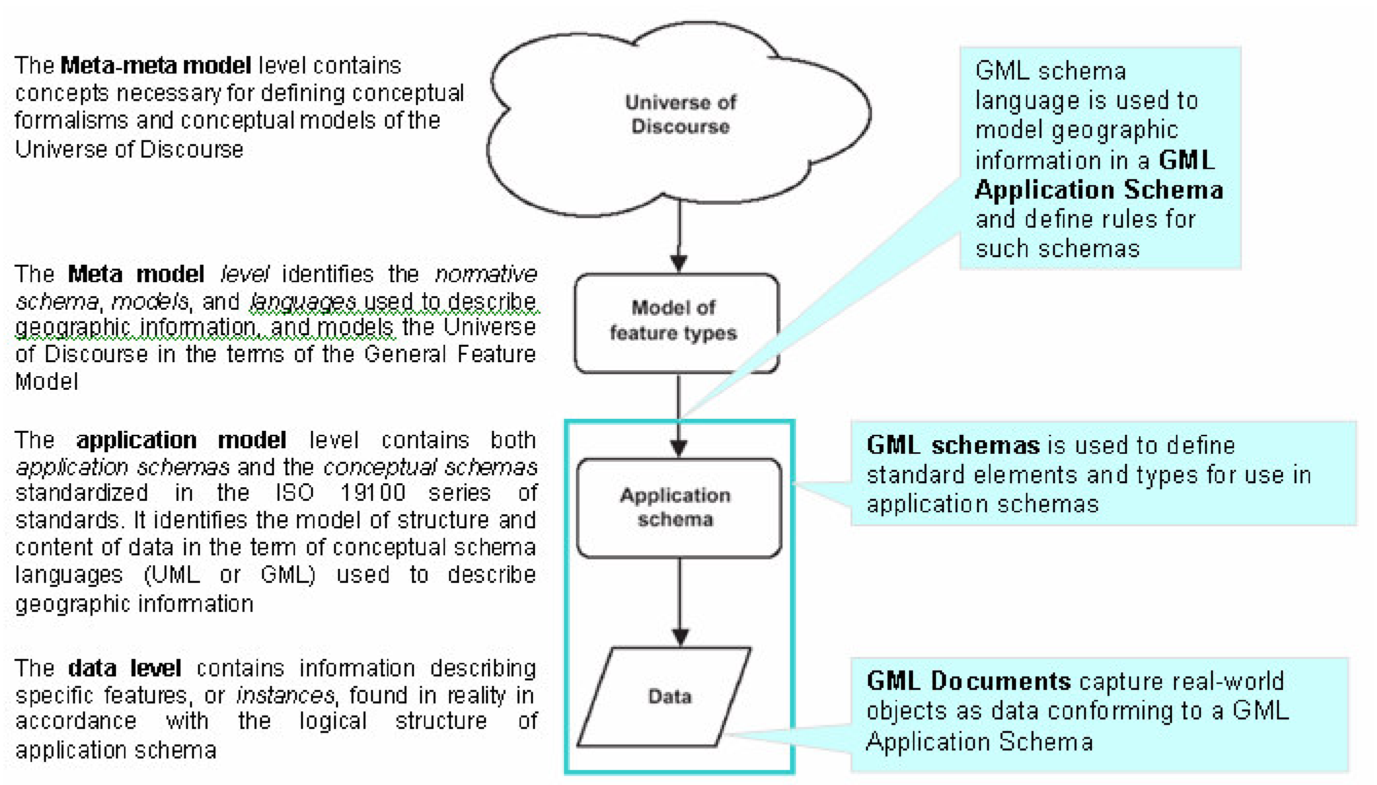

The **Meta-meta model** level contains concepts necessary for defining conceptual formalisms and conceptual models of the Universe of Discourse
The **Meta model** *level* identifies the *normative schema*, *models*, and *languages* used to describe geographic information, and models the Universe of Discourse in the terms of the General Feature Model
The **application model** level contains both *application schemas* and the *conceptual schemas* standardized in the ISO 19100 series of standards. It identifies the model of structure and content of data in the term of conceptual schema languages (UML or GML) used to describe geographic information
The **data level** contains information describing specific features, or *instances*, found in reality in accordance with the logical structure of application schema
Universe of Discourse
Model of feature types
Application schema
Data
GML schema language is used to model geographic information in a **GML Application Schema** and define rules for such schemas
GML schemas is used to define standard elements and types for use in application schemas
GML Documents capture real-world objects as data conforming to a GML Application Schema

Figure: Alternatively, an application schema can be constructed directly by adhering to the rules for GML in XML Schema (see ISO 19101 - Reference Model for details)

The GML model has a straightforward representation using the UML profile. Thus, features are represented in:

- UML by objects, where the name of the feature type is used as the name of the object class
- GML instances by XML elements, where the name of the feature type is used as the name of the element.

Feature properties are represented in:

- UML by association roles with feature type classes and attributes of feature type classes, where the property semantics are given by the association role name or attribute name
- GML instances by sub-elements (known as property elements) of feature elements, where the property semantics are given by the property element name.

The property value has a type indicated in:

- UML by the class of the association target, or by the data type of the attribute
- GML, in the case of properties with complex values, by the name of the object element contained within the property element and in case of a property with simple value by the type of the literal value containing no embedded XML markup.

GML defines the various entities such as features, geometries, topologies, etc. through a hierarchy of GML object types. The normative GML schema is organized around these object types. A GML application must reuse core GML features defined in the conceptual model. A feature is encoded as an XML element with the name of the feature type. Other identifiable objects are encoded as XML elements with the name of the object type. Each feature attribute and feature association role is a property of a feature. Feature properties are encoded in an XML element.

A property element may contain its value as content encoded inline, or reference its value with a simple XLink. The value of a property may be simple, or it may be a feature or other complex object.

Thus, ISO 19136 describes the normative XSD Schemas, explains their contents, structure and dependencies, and defines how to encode geographic information for:

- Basic data types that are used in the GML schema
- Basic geometry (0d, 1d, 2d)
- Additional geometric primitives (0d, 1d, 2d, 3d)
- Geometric composites
- Geometric aggregates
- Coordinate reference systems
- Topology
- Temporal information and dynamic features
- Definitions and dictionaries
- Units, measures, and values
- Directions
- Observations
- Grids and coverages
- Default styling

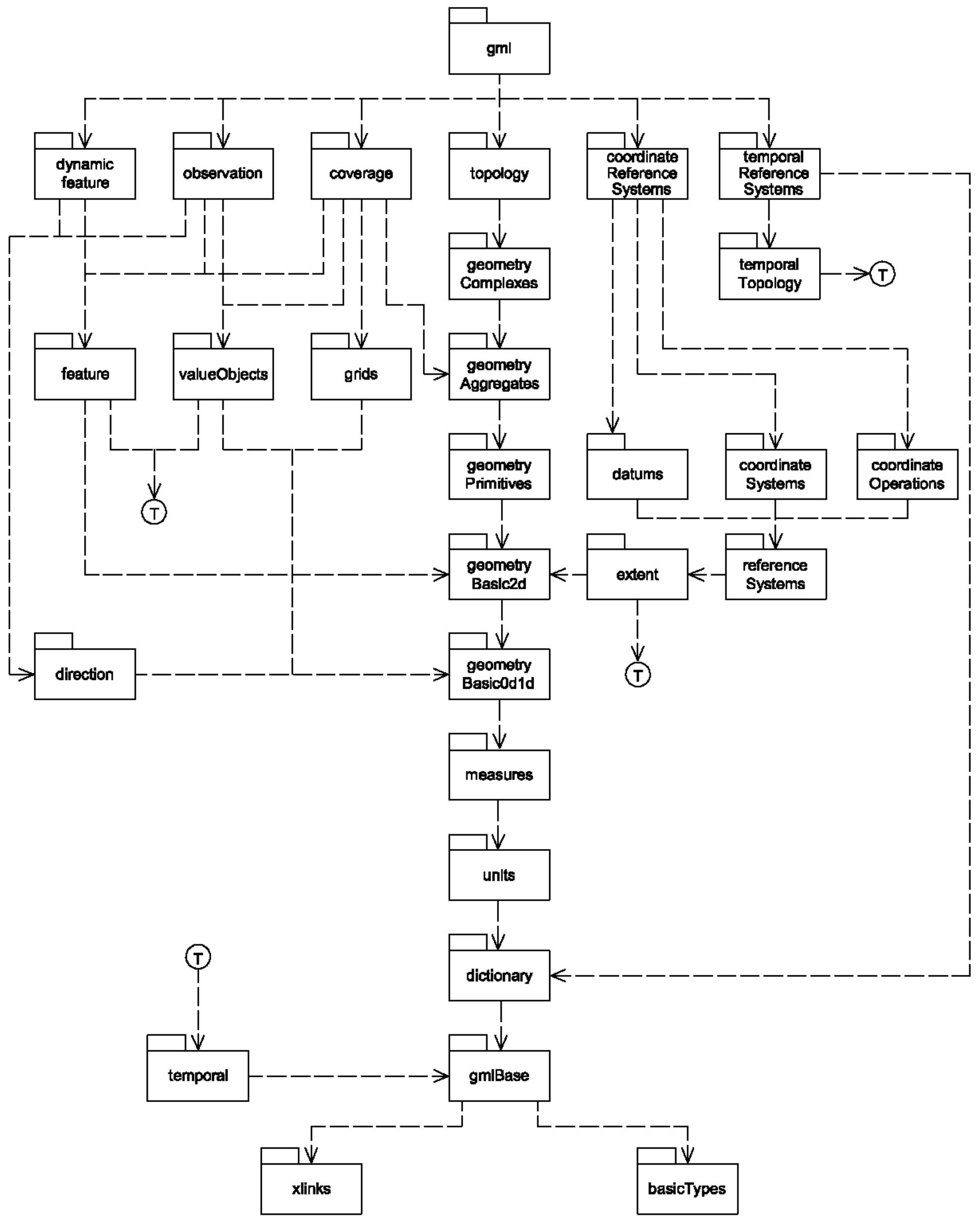


Figure: GML Schemas with dependencies (ISO 19136)

The GML schema comprises the components (XML elements, attributes, simple types, complex types, attribute groups, groups, etc.) that are described in the ISO 19136.

Designers of particular GML application schemas may extend or restrict the types defined in the GML schema to define appropriate types for their application domain. Non-abstract elements, attributes, and types from the GML schema may be used directly in an application schema, if no changes are required.

If a developer is modeling a particular class of object, one will need a specific XML Schema Document (XSD). For example, if geographic features are modeled, one will need the feature.xsd. If features have properties which make use of units of measure, one of the basicTypes.xsd, measure.xsd, and valueObjects.xsd will need to be imported. If a spatial reference system is used, coordinateReferenceSystems.xsd will be required. If topology is applied to model spatial primitives, topology.xsd has to be imported. If coverages (e.g. remotely sensed images, aerial photographs, soil distribution, and digital elevation models) are constructed, coverage schemas will be needed, etc. Each model complies with respective ISO standards (e.g. the geometry model of GML complies with ISO 19107 - Spatial schema). Several schemas can be brought together in the same document. The GML schema documents are available online.

Let us look on some examples of GML XSD schema elements and respective results of XML data encoding.

The element `gml:AbstractFeature` is declared as follows:

```
<element    name="AbstractFeature"    type="gml:AbstractFeatureType"    abstract="true"
substitutionGroup="gml:AbstractGML"/>
```

`gml:AbstractFeature` may be thought of as anything that is a GML feature and may be used to define variables or templates in which the value of a GML property is any feature.

The basic feature model is given by the `gml:AbstractFeatureType`, defined in the schema as follows:

```
<complexType name="AbstractFeatureType" abstract="true">
<complexContent>
 <extension base="gml:AbstractGMLType">
   <sequence>
      <element ref="gml:boundedBy" minOccurs="0"/>
      <element ref="gml:location" minOccurs="0"/>
   </sequence>
 </extension>
</complexContent>
</complexType>
```

The content model for `gml:AbstractFeatureType` adds two specific properties suitable for geographic features to the content model defined in `gml:AbstractGMLType`. The value of the `gml:boundedBy` property describes an envelope that encloses the entire feature instance and is primarily useful for supporting rapid searching for features that occur in a particular location. The value of the `gml:location` property describes the extent, position, or relative location of the feature.

The `gml:locationName` property element is a convenience property where the text value describes the location of the feature. It is defined as follows:

```
<element name="locationName" type="gml:CodeType"/>
```

An illustration of how a data instance may appear for a location given using a text string:

```
<Feature>
        <gml:locationName>Klaipeda town of residence</gml:locationName>
</Feature>
```

The `gml:locationReference` property element is a convenience property where the text value referenced by the `xlink:href` attribute describes the location of the feature. It is defined as follows:

```
<element name="locationReference" type="gml:ReferenceType"/>
```

The following is an example that illustrates how a data instance may appear for a location given by another service:

```
<Feature>
        <gml:locationReference
xlink:href="http://www.somesite.lt/bin/gazm01?placename=kliapeda&placetype=R&province=P
A+"/>
</Feature>
```

In accordance with GML 3.2 specifications, application-specific names must be chosen for geometry properties in GML application schemas. The names of the properties should be chosen to express the semantics of the respective values. Using application specific names is the preferred method for names of properties, including geometry properties.

For example, an `<ext:Tower>` feature type for external application could have a location that returns point geometry to identify its location at the point. The `Tower` feature type could be defined by the user according to the following schema:

```
<element name="Tower" type="ext:TowerType" substitutionGroup="gml:_Feature"/>
<complexType name="TowerType">
   <complexContent>
      <extension base="gml:AbstractFeatureType">
        <sequence>
          <element ref="gml:pointProperty"/>
        </sequence>
      </extension>
   </complexContent>
</complexType>
```

The GML schema includes predefined property types (e.g., `PointProperty` type) that may be used as types of geometry property element.

An example that illustrates how a data instance may appear for a `Tower` instance:

```
<ext:Tower>
<gml:pointProperty>
   <gml:Point srsName="urn:x-ogc:def:crs:EPSG::4326">
     <gml:pos>2478112.85645389 6695151.20645945</gml:pos>
   </gml:Point>
 </gml:pointProperty>
</ext:Tower>
```

This example uses available `gml:Point` object from `PointProperty` type as defined in the GML geometry schemas. The same property element may be used to indicate a location by a reference such as:

```
<ext:Tower xlink:href="http://my.somesite.org/locations/point96"/>
```

Where `http://my.somesite.org/locations/point96` a point (a `gml:Point` element) supplied by the service indicated. The object is either a child element of the property or referenced by an xlink:href attribute in the property element. The `xlink:href` attribute is interpreted in the way that the value of the property is the object referenced in the link. The object can be part of the same GML document or anywhere in the internet/intranet.

Any geometry element that inherits the semantics of `gml:AbstractGeometryType` may be viewed as a set of direct positions `<gml:pos>`. Direct position instances hold the X and Y-coordinates (e.g. `<gml:pos>2478112.85645389 6695151.20645945</gml:pos>`) for a position within some coordinate reference system (CRS) (e.g. `4326` CRS that defines from `urn:x-ogc:def:crs:EPSG::4326`). All of the classes derived from `gml:AbstractGeometryType` inherit an optional association to a coordinate reference system from `srsName` attribute of `gml:SRSReferenceGroup` attribute group. All direct positions must directly or indirectly be associated with a coordinate reference system from the `srsName` attribute.

Let's look at an example for a road feature with attributes whose structure is defined in the following figure:

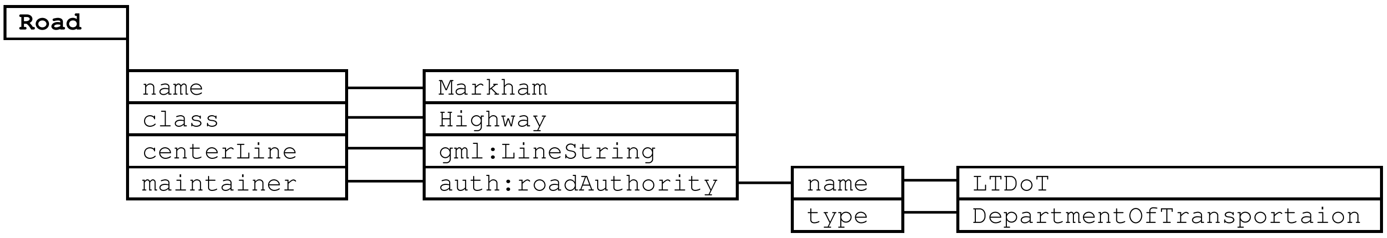


Figure: An application schema of `Road` feature

For example, modeling the `Road` linear feature type for an application external to GML, with attributes `road_name`, `class`, and `maintainer`, can be defined by users according to the following schema:

```
<element name="Road" type="ext:RoadType" substitutionGroup="gml:_Feature"/>
 <complexType name="RoadType">
  <complexContent>
  <extension base="gml:AbstractFeatureType">
  <sequence>
    <element name="road_name" minOccurs="0">
     <simpleType>
     <restriction base="string">
     <maxLength value="50"/>
     </restriction>
</simpleType>
    </element>
    <element name="class" minOccurs="0" type="string"/>
    <element ref="auth:roadAuthority"/>
    <element ref="gml:centerLine" minOccurs="0"/>
    <element ref="gml:multiCurveProperty" minOccurs="0"/>
  </sequence>
  </extension>
  </complexContent>
 </complexType>
```

Example that illustrates how a data instance may appear for a `Road` instance:

```
<ext:Road gml:id="id6df7af28-9cf1-4f0c-8858-7eaed738b544">
   <ext:name>Markham</ext:name>
   <ext:class>Highway</ext:class>
   <auth:maintainer gml:id=„o.1f32a3">
     <name>LTDoT</name>
     <type>DepartmentOfTransportaion</type>
   </auth:maintainer>
   <gml:centerLine>
     <gml:LineString srsName="urn:x-ogc:def:crs:EPSG::4326">
        <gml:posList>2233796.19000493         6508921.45806473         2379332.41231563
6536523.15539952  2477192.97559352  6591726.5500691  2554979.57717337  6619328.24740389
2670404.85693703  6654457.68037544  2815941.07924774  6659476.17079994  2956458.81113393
6659476.17079994</gml:posList>
     </gml:LineString>
   </gml:centerLine>
</ext:Road>
```

The `gml:AbstractCurve` element is the abstract head of the substitution group for all continuous curve elements. A property `curveProperty` is the predefined property which may be used by GML application schemas whenever a GML feature has a property with a value that is substitutable for `gml:AbstractCurve`. A `gml:LineString` is a special curve that consists of a single segment with linear interpolation. It is defined by two or more coordinate tuples, with linear interpolation between them.

The next example illustrates how a data instance may appear for an external `ext:School` type represented by polygonal element in GML, as shown below:

```
<ext:School gml:id="idc8feb418-6934-42d6-ab01-436b49e8e111">
 <gml:surfaceProperty>
   <gml:Surface srsName="urn:x-ogc:def:crs:EPSG::4326">
    <gml:patches>
     <gml:PolygonPatch>
      <gml:exterior>
       <gml:LinearRing>
         <gml:posList>2522359.38941408         6516449.19370149         2522359.38941408
6544050.89103628  2562507.31281013  6544050.89103628  2562507.31281013  6516449.19370149
2522359.38941408 6516449.19370149</gml:posList>
       </gml:LinearRing>
      </gml:exterior>
     </gml:PolygonPatch>
    </gml:patches>
   </gml:Surface>
 </gml:surfaceProperty>
</ext:School>
```

Figure: Polygonal feature in GML document

A `gml:Polygon` is a special surface that is defined by a single surface patch `<gml:PolygonPatch>`. The boundary of this patch is coplanar and the polygon uses planar interpolation in its interior. The elements `gml:exterior` and `gml:interior` describe the surface boundary of the polygon. A boundary of a surface consists of a number of rings. In the normal 2D case, one of these rings is distinguished as being the exterior boundary. A `gml:LinearRing` is defined by four or more coordinate tuples, with linear interpolation between them; the first and last coordinates must be coincident. The `gml:posList` element allows for a compact way to specify the coordinates of the positions, if all positions are represented in the same coordinate reference system.

GML modeling and encoding provides support for conceptual design of application schema and for software developers. Support for application schema designers includes the rules for application schemas that are used as guidelines for modeling of data structures. Support for software developers includes predefined GML specification that is used as guidelines on how to work with and process XML documents.

ESRI ArcGIS Data Interoperability extension supports export/import of ESRI formats into GML for Simple Feature format.

4.3. ESRI Arc Extensible Markup Language (ArcXML)

Unlike GML, which *focuses on content*, ***Arc Extensible Markup Language*** (ArcXML) is used to define ArcIMS ***map configuration files***, contents of *request* and *respond* messages, and ESRI ArcIMS service ***communication***. ArcXML is also a derivative of XML that is a subset of SGML.

ArcXML uses a hierarchical system of tags and sub-tags. There are five root tags in ArcXML:

- `<ARCXML>` is used in every ArcXML document to declare what it is
- `<CONFIG>` is used to configure a map service by defining the data to be included and how it will appear
- `<REQUEST>` is used to make a request from the client to the server
- `<RESPONSE>` is used to make a response from the Spatial Server to the client
- `<MARKUP>` is used to describe changes made on the client.

All ArcXML statements are constructed using elements and attributes organized in a hierarchical structure. As part of this structure, elements are categorized as parent and/or child elements. Child elements are embedded inside parent elements. Most elements have one or more attributes. Attributes are a series of name-value pairs that describe an element. Some ArcXML elements require attributes. If an attribute is not specified for an element in the map configuration file, a default value will be used if one is available for the attribute. ArcXML file organization agrees with W3C XML recommendations.

When writing elements and attributes, a strict convention must be followed or ArcIMS may not be able to create the map service. Elements, which identify the nature of the content being described, must be written in uppercase characters and properly opened and closed with less than (`<`) and greater than (`>`) characters. Attribute names must be in lowercase characters and the associated values enclosed in double quotes. Only one attribute value can be defined at a time.

The ***map configuration files*** are used as input to ArcIMS services. This file provides information about map properties and instructions on how to render maps as ESRI Image (WMS) and Feature (WFS) services. This file contains elements that define the local environment (the language, etc) and map elements (map properties, workspaces, and layers).

A map configuration file is a text file with an *.axl* extension and is distinguished by the `<CONFIG>` root tag within it. The structure of a map configuration file is:

```
<?xml version="1.0" encoding="UTF-8"?><ARCXML version="1.1">
  <CONFIG>
    <ENVIRONMENT>...</ENVIRONMENT>
    <MAP>
      <PROPERTIES>...</PROPERTIES>
      <WORKSPACES>...</WORKSPACES>
      <LAYER>...</LAYER>
    </MAP>
  </CONFIG>
  </ARCXML>
```

- The `PROPERTIES` section includes the initial map extent, map units, and current projection, as well as additional instructions used for Image Services
- The `WORKSPACES` section includes the location of all the data used to create map layers
- The `LAYER` element is used for each layer in a map and contains the information about how the data should be symbolized.

So from this, it can be seen, that ESRI *.axl* file *does not contain a content of actual data*, but only paths to used data (e.g., paths to shapefile, images, ArcSDE databases). Therefore, ArcXML services completely different purposes compared to GML.

The following sample map configuration file uses the elements discussed above and some additional child elements that are described in the ArcXML Programmer's Reference Guide (http://edndoc.esri.com/arcims/9.2/). The purpose of `</MAP>` child elements is to identify the data source and identify how to render the data.

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

   <ARCXML version="1.1">
     <CONFIG>
       <ENVIRONMENT>
         <LOCALE country="US" language="en" variant="" />
         <UIFONT color="0,0,0" name="SansSerif" size="12" style="regular" />
         <SCREEN dpi="96" />
       </ENVIRONMENT>
       <MAP>
         <PROPERTIES>
        <ENVELOPE minx="306455.4687500001" miny="5973368.499999999"
maxx="680107.6249999998" maxy="6257754.0" name="Initial_Extent" />
        <MAPUNITS units="meters" />
         </PROPERTIES>
         <WORKSPACES>
        <SHAPEWORKSPACE name="shp_ws-0" directory="E:\ArcIMS\Data\LT" />
         </WORKSPACES>
         <LAYER type="featureclass" name="Municipalities" visible="true" id="0">
        <DATASET name="savivaldybes" type="polygon" workspace="shp_ws-0" />
        <SIMPLERENDERER>
          <SIMPLEPOLYGONSYMBOL boundarytransparency="1.0" filltransparency="1.0"
fillcolor="153,255,102" boundarycaptype="round" boundarycolor="255,200,0" />
        </SIMPLERENDERER>
         </LAYER>
       </MAP>
     </CONFIG>
   </ARCXML>
```

This .axl file provides for an ESRI web mapping service use of `savivaldybes` polygonal shape file from `E:\ArcIMS\Data\LT` folder to render one `Municipalities` layer with `boundarytransparency="1.0"`, `filltransparency="1.0"`, `fillcolor="153,255,102"` `boundarycaptype="round"` and `boundarycolor="255,200,0"` `SIMPLEPOLYGONSYMBOL` styles how it looks like at the following figure.

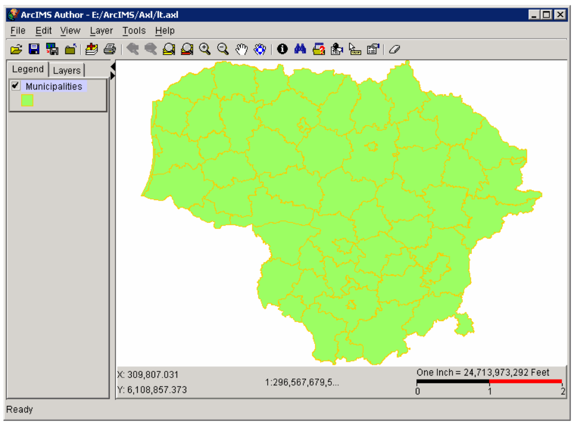

ArcIMS Author - E:/ArcIMS/Axl/lt.axl
File Edit View Layer Tools Help
Legend
Layers
Municipalities
X: 309,807.031
Y: 6,108,857.373
1:296,567,679,5...
One Inch = 24,713,973,292 Feet
0
1
2
Ready

Figure: ArcIMS Author can be used as an .axl file editor

The *.axl* file can also be used to define projections for particular layers (e.g., `<COORDSYS id="54030"`), to join DBF files with `<SPATIALQUERY joinexpression …>` tags, to assign additional rendering and labeling options, to subsets data to label features, etc.

ArcXML is also used for ***communication*** between the client and ESRI web map servers, and between the business logic tier and servers of ArcIMS. ArcXML is also used as the protocol for communicating with the ArcIMS Spatial Server. Module 5 shows the communication flow between ArcIMS client and server components in more detail.

ArcXML defines content for the following *requests* from clients:

- `GET_IMAGE` - a request to generate a map image from ArcIMS or ArcGIS Server
- `GET_FEATURES` - requests features as a compressed binary stream in ESRI format from the ArcIMS Feature Server or requests attribute data in ArcXML format from the ArcIMS Query or ArcMap Servers
- `GET_GEOCODE` - a request containing address information to the Geocode Server
- `GET_EXTRACT` - a request to the Extract Server to extract content layers into shapefile format, placed in a zip file and shipped to a client
- `GET_SERVICE_INFO` - requests information about each layer in an ArcIMS service. With Image and Feature services, the request has options for returning information on the fields, envelopes, extensions, and renderers. With ArcMap Image services, the request has options for returning information on fields, envelopes, dataframes, and the table of contents
- `GET_RASTER_INFO` - a request to return the pixel value of an image at a given x,y coordinate location for the specified layer
- `GET_LAYOUT` – a request to get an ArcMap layout
- `GET_PROJECT` - a request to re-project geometry from one coordinate system to another coordinate system
- `GET_METADATA` - a request to send requests to the ESRI Metadata Service
- `PUBLISH_METADATA` – a request for administering and publishing metadata

Each ArcXML type of *request* has its respective ArcIMS Spatial Server type of *response:*

- `GET_EXTRACT` response is `EXTRACT`
- `GET_FEATURES` response is `FEATURES`
- `GET_GEOCODE` response is `GEOCODE`

- `GET_IMAGE` response is `IMAGE`
- `GET_LAYOUT` response is `LAYOUT`
- `GET_METADATA` response is `METADATA`
- `GET_PROJECT` response is `PROJECT`
- `GET_RASTER_INFO` response is `RASTER_INFO`
- `GET_SERVICE_INFO` response is `SERVICEINFO`
- `PUBLISH_METADATA` response is `METADATA_ACTION`

Sample of `<GET_IMAGE>` request:

```
  <ARCXML version="1.1">
  <REQUEST>
  <GET_IMAGE>
  <PROPERTIES>
  <ENVELOPE minx="429196.520177882" miny="5445070.86354896" maxx="429996.733754915"
maxy="5445803.33475179" />
  <IMAGESIZE width="500" height="350" />
  </PROPERTIES>
  </GET_IMAGE>
  </REQUEST>
  </ARCXML>
```

The response by ArcIMS that returns part of location to the map image (`http://tree.mala.bc.ca/output/Municipalities.jpg`) and visualizes the `Municipalities.jpg` image within a client Internet browser could be:

```
  <?xml version="1.0" encoding="UTF-8"?>
  <ARCXML version="1.1">
  <RESPONSE>
  <IMAGE>
  <ENVELOPE minx="429073.433250091" miny="5445070.86354896" maxx="430119.820682706"
maxy="5445803.33475179" />
  <OUTPUT url="http://tree.mala.bc.ca/output/Municipalities.jpg" />
  </IMAGE>
  </RESPONSE>
  </ARCXML>
```

In addition, ArcIMS and ArcGIS Server support OGC WMS and WFS servers via ESRI WMS and WFS Connecters. In such scenarios, the communication between client and server supports the following OGC XML POST requests-responses:

- `GetCapabilities` – a request provides information about a WMS or WFS service
- `GetMap` – a request retrieves a map from a WMS site
- `GetFeatureInfo` - a request provides feature information by identifying a point on a map based on its pixel location
- `DescribeFeatureType` - a request generates a schema description for feature types serviced by a WFS implementation

- `GetFeature` – a request retrieves features from a Web Feature Service. The `GetFeature` response will return geometry data in the GML encoding.

The ArcXML Data Type Definition (DTD) is available for use with ArcXML 1.1. This DTD defines the structure rules for the elements and attributes in ArcXML and checks for document validity. If the developer wants to validate ArcXML, one can use the DTD that comes with ArcIMS (http://support.esri.com/index.cfm?fa=downloads.samplesUtilities.viewSample&PID=16&MetaID=1242).

The DTD can be used with many XML editors, such as XML Spy (http://www.xmlspy.com) or Xeena (http://alphaworks.ibm.com/tech/xeena). When using the DTD with ArcXML, element names, attributes, and enumerated attribute values are case sensitive as defined in the DTD. ArcXML element names are always uppercase. Attributes are always lowercase. Enumerated attribute values are generally all lowercase.

When using the `DOCTYPE` statement in a map configuration file, request, or response, the statement must come second after the XML declaration line. The `DOCTYPE` statement contains information on the location of the DTD. For example:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE ARCXML SYSTEM "<dtd_location>/arcxml.dtd">
<ARCXML version="1.1">
...
</ARCXML>
```

We will learn more about *.axl* and ArcXML communication functions in the next Module.

4.4. Google Keyhole Markup Language (KML)

The ***Google Keyhole Markup Language*** (KML) is an XML-based markup language designed to consume Google Maps web mapping service and display additional user-specified features within Google Earth, Google Maps, and Google Maps for mobile, and other programs, including ESRI ArcGIS Explorer, NASA WorldWind, Adobe PhotoShop, AutoCAD, Flickr, and Yahoo Pipes.

KML was initially developed for use with Google Earth. Because that project was originally named Keyhole, as was the company that undertook this work, the related markup language followed suit. When Google acquired Keyhole in 2004, that project came with it and eventually became Google Earth.

Google Maps is a map service that fuses several different mapping technologies with other Google services to create interactive online maps. In the web domain, Google Maps service can be served as Service provider for Service requestor within ***Service Oriented Architecture** (**SOA**).* Web services are the set of protocols by which services can be published, discovered, and used in a technology neutral, standard form. SOA is a method of building business applications that utilize common services to support business functions, as discussed in the last section of this module. In SOA, some web service once published can be consumed through a variety of client applications and processes by the client.

Thus, for example, service requestor uses KLM language to invoke Google Maps web mapping service and ***mashups*** additional user-defined features on the top of generated map. The term “mashup” has been used to describe web applications that combine data and content from more than one source into a single integrated display. An example is the use of data mashup to add location information to real-estate properties from a web-based real-estate list on the top of cartographic or remote sensing data from Google Maps. New web services, not originally provided by either source, is now created.

Similar to GML and ArcXML, KML uses a tag-based structure with nested elements and attributes and is based on the XML standard. All tags are case-sensitive and must be appear exactly as they are listed in the KML 2.2 Reference. The Reference indicates which tags are optional. Within a given element, tags must appear in the order shown in the Reference (http://code.google.com/apis/kml/documentation/kml_tags_beta1.html).

KLM provides the opportunity to add annotations and overlay features on two-dimensional online Google Maps or in three-dimensional Google Earth applications. A KML file includes specifications for various features for display. The KML feature set includes placemarks, 3D models, text descriptions, images, ground overlays, paths, polygons, and so forth (see for more at the http://code.google.com/apis/kml/documentation/kml_tags_beta1.html). Each location has an associated longitude and latitude (KLM supports only geographic coordinates in WGS84 datum, and altitude as a distance above the Earth's surface, in meters) and view-specific data such as heading, altitude, and tilt may be provided to define a so-called "camera view" for geospatial data. KML shares some of its grammar for geometry with the GML.

Sample of a Placemark for the Point feature with associated geometry is:

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://earth.google.com/kml/2.0">
<Placemark>
   <description> Vilnius City</description>
   <name>Vilnius</name>
   <Point>
     <coordinates>25.258197,54.680445,0</coordinates>
   </Point>
</Placemark>
</kml>
```

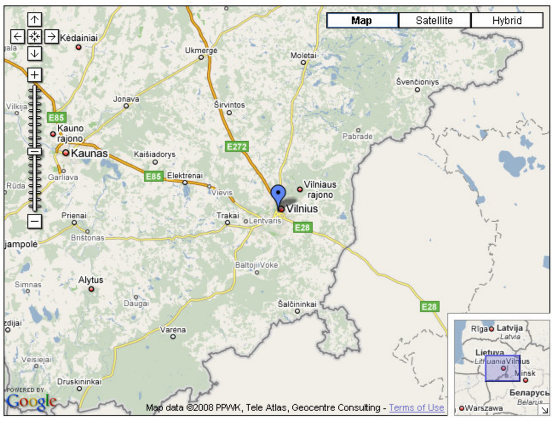


Figure: A Placemark with a Point has an icon associated with it that marks a point on the Earth. To pre-view this code on a Google Map go to http://www.thechrisoshow.com/display_kml/

KML documents are saved as text files with .kml or .kmz extensions. KML documents can be distributed in the form of .kmz files, which are zipped KML documents. A .kmz file usually contains a single KML document (named “doc.kml”), along with images for overlays and icons it may reference internally. KML files can be created with the Google Earth user interface or with an XML or text editor.

The current specification for KML is 2.2, which has been submitted to the Open Geospatial Consortium (OGC) for standardization as an open standard that any geo-browser may use. Because the OGC already has custody over GML, this makes good sense.

KML is being widely adopted by many applications and it is important to make sure that KML files are compliant with KML standards. There are two ways to validate KML. O,nline validation tools are available, such as the Feedvalidator (feedvalidator.org). In addition, it is possible to do offline validation with XML KML Schema (http://code.google.com/apis/kml/schema/kml21.xsd) by using XML parser editors that allow schema validation (e.g., Xerces).

4.5. Scalable Vector Graphics (SVG)

SVG stands for Scalable Vector Graphics, an XML grammar language used to display two-dimensional stylable vector graphics with usable XML namespaces. SVG 1.1 is a W3C Recommendation that is compatible with the "Extensible Markup Language (XML) 1.0" Recommendation and forms the core of current SVG developments. SVG formats can describe both static and animated vector graphics. SVG can be purely declarative or may include scripting. Images can contain hyperlinks using outbound simple XLinks.

To be ***scalable*** means to increase or decrease uniformly. In terms of graphics, scalable means not being limited to a single, fixed, pixel size. SVG graphics are scalable to different display resolutions. The same SVG graphic can be placed at different sizes on the same web page and re-used at different sizes on different pages. SVG graphics can be magnified to see fine detail. Unlike bitmap graphics (JPEG, GIF, PNG), SVG graphics can be zoomed in and out as necessary.

Vector graphics contain geometric objects such as lines and curves. This gives greater flexibility compared to raster-only formats (such as PNG and JPEG) that have to store information for every pixel of the graphic. Typically, vector formats can also integrate raster images and can combine them with vector information. SVG is no exception.

SVG is also intended to be used as one component in a multi-namespace XML application. This multiplies the power of each of the namespaces used to allow innovative new content to be created. For example, SVG graphics may be included in a document that uses any text-oriented XML namespace - including XHTML. SVG is a good, general-purpose component for any multi-namespace grammar that needs to use graphics.

SVG allows three types of graphic objects:

- Vector graphics
- Raster graphics
- Text

As an application of XML, a SVG file is, therefore, a simple text file, which can be viewed and edited as with any other markup. For example, the following SVG markup appears in a capable viewer as a star :

```
  <?xml version="1.0" standalone="no"?>
  <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
    "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
  <svg width="12cm" height="4cm" viewBox="0 0 1200 400" version="1.1"
xmlns="http://www.w3.org/2000/svg" preserveAspectRatio="xMidYMid meet"
zoomAndPan="magnify">
    <desc>Example - star </desc>
    <!-- Show outline of canvas using 'rect' element -->
    <rect x="1" y="1" width="1198" height="398" fill="none" stroke="blue" stroke-
width="2"/>
    <polygon fill="red" stroke="blue" stroke-width="10" points="350,75 379,161 469,161
397,215 423,301 350,250 277,301 303,215 231,161 321,161"/>
  </svg>
```

In the example, the `<polygon>` tag is used to create a graphic that contains at least three sides. SVG files must be saved with a *.svg* extension. However, SVG files could be embedded into HTML pages.

ESRI ArcGIS supports export of their vector formats into SVG graphics.

The use of SVG on the web requires a SVG enabled browser. At this time, all major browsers have committed to some level of SVG support, except for Internet Explorer. As of 2007, Internet Explorer requires a plugin to view SVG content. The most widely available SVG plugin on the desktop is from Adobe Systems and supports most of SVG 1.0/1.1.

The formal definition of each of the SVG abstract modules, as a DTD module, can be found at http://www.w3.org/TR/SVG11/svgdtd.html. The DTD contains all allowable SVG elements. Read more about SVG at http://www.w3.org/TR/SVG11/ or http://www.w3schools.com/svg/.

4.6. Web Map Services and Service Oriented Architecture (SOA)

A Service-Oriented Architecture (SOA) is a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity.

A service is a function that is well-defined, self-contained, and does not depend on the context or state of other services. A service is the endpoint of a connection. In the service-oriented architecture, a service consumer is sending a service request message to a service provider. The service provider returns a response message to the service consumer. The request and subsequent response connections are defined in some way that is understandable to both the service consumer and service provider. A service provider can also be a service consumer.

Web services can be used to implement a service-oriented architecture. A major focus of web services is to make functional provider's blocks accessible over standard Internet protocols that are independent from platforms and programming languages. These services can be new applications or just wrapped around existing legacy systems to make them network-enabled.

Each SOA building block can play one or more of three roles:

- *Service provider* creates a web service and possibly publishes its interface and access information to the service registry.
- The *service broker* or service registry is responsible for making the web service interface and implementation access information available to any potential service requestor. Public brokers are available through the Internet. There are also brokers that catalog other brokers. The ***Universal Description Discovery and Integration*** (UDDI) specification defines a way to publish and discover information about Web services.
- The *service requestor* (or web service client, consumer) locates entries in the broker registry using various find operations and then binds to the service provider in order to invoke one of its web services.

Some means of connecting services to each other is needed. The technology of connection is the technology of service-oriented architectures. Web services use XML to create a robust connection. XML has been used extensively in SOA to create data that is wrapped in a nearly exhaustive description container. Analogously, the services themselves are typically described by ***Web Services Description Language*** (WSDL), and communications protocols by ***Simple Object Access Protocol*** (SOAP). Therefore, ***Web service*** *is a software component, described via WSDL, that is accessible via SOAP over HTTP*.

Web Services Description Language (WSDL) is a format for describing a Web Service interface or to describe services and how they can be bound to specific network addresses. WSDL has three parts:

- *Definitions* are expressed usually in XML and include both data type definitions and message definitions. These definitions are usually based upon some agreed upon XML vocabulary, such as OGC Web Services (OWS) specifications.

- *Operations* describe actions for the messages supported by a Web service. There are four types of operations:
    - One-way messages sent without a reply required
    - Request/response messages requires reply from the receiver
    - Solicit response that is a request for a response
    - Notification messages sent to multiple receivers

  Operations are grouped into port types. Port types define a set of operations supported by the Web service.

- Service bindings connect port types to a port. A port is defined by associating a network address with a port type. A collection of ports defines a service. This binding is commonly created using SOAP, but other forms may be used.

The WSDL forms the basis for Web Services. The steps involved in providing and consuming a service are:

1. A service provider describes its service using WSDL. This definition is published to a directory of services; the directory could use UDDI.
2. A service requestor issues a request to the service broker directory to locate a service and determine how to communicate with that service.
3. Part of the WSDL provided by the service provider is passed along to the service requestor. This tells the service consumer what the requests and responses are for the service provider.
4. The service requestor uses the WSDL to send a request to the service provider.
5. The service provider provides the expected response to the service requestor.

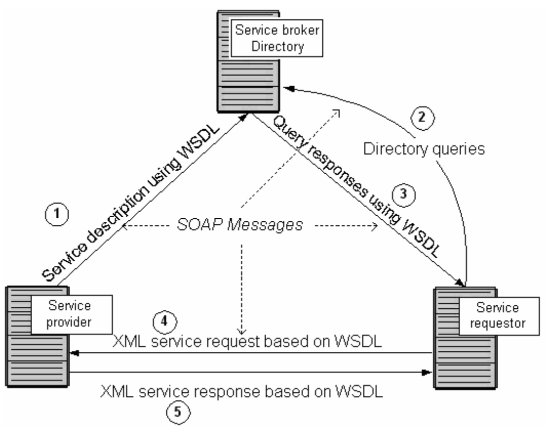

Service broker Directory
2
Directory queries
1
Service description using WSDL
Query responses using WSDL
3
SOAP Messages
Service provider
4
XML service request based on WSDL
Service requestor
XML service response based on WSDL
5

Figure: The step to use WSDL

The service broker directory could be a UDDI registry. Universal Description, Discovery, and Integration provides the definition of a set of services supporting the description and discovery of businesses, organizations, and other Web Services providers, the Web Services they make available, and the technical interfaces that may be used to access those services. UDDI is based on a common set of industry standards, including HTTP, XML, XML Schema, and SOAP.

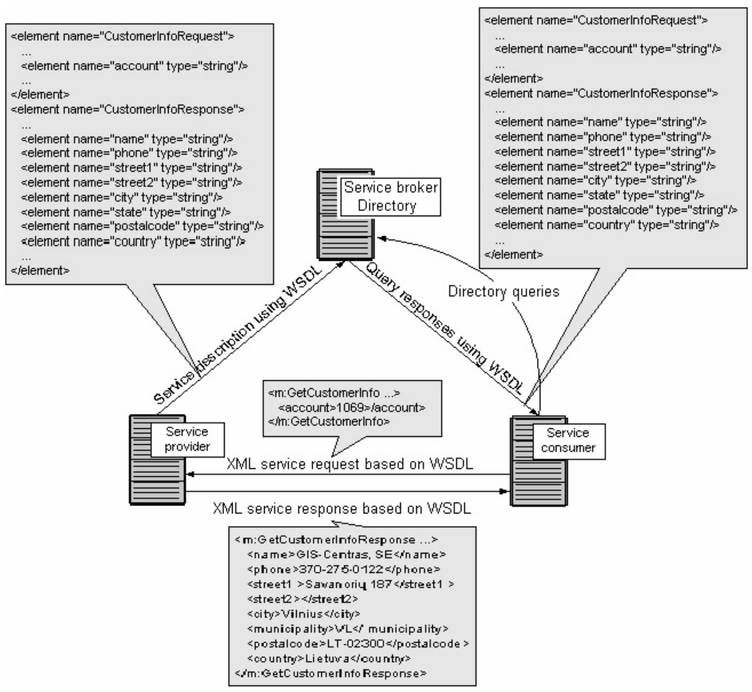
The idea is to "discover" organizations and the services that organizations offer, much like using a phone book or dialing information. The UDDI registry can be used to search in various ways to obtain contact information and the Web Services available for various organizations. Registry system consists of three directories: white pages, yellow pages, and green pages.

All the messages within SOA are sent using Simple Object Access Protocol (http://www.w3.org/TR/2007/REC-soap12-part0-20070427/) that is part of the set of standards specified by the W3C. SOAP allows computers to communicate independently of an operating system or platform by using HTTP and XML. SOAP is a method for using Extensible Markup Language (XML) in message and remote procedure call (RPC) based protocols. SOAP is fundamentally a stateless, one-way message exchange paradigm. SOAP Version 1.2 provides the definition of the XML-based information that can be used for exchanging structured and typed information between peers in a decentralized, distributed environment via messages. A SOAP message is formally specified in an XML grammar with SOAP elements (e.g., <env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">), which provides an abstract description of its contents.

SOAP provides the <env> envelope for sending the Web Services messages. SOAP is generally used to send messages over HTTP, but other means of connection may be used. SOAP can be used to exchange complete documents or to call a remote procedure.

The SOAP envelope contains two parts (see more at the http://www.w3.org/TR/2007/REC-soap12-part0-20070427/#intro):

1. An optional header providing information on authentication, encoding of data, or how a recipient of a SOAP message should process the message.
2. The body that contains the message. These messages can be defined using the WSDL specification.



```
<element name="CustomerInfoRequest">
  ...
  <element name="account" type="string"/>
  ...
</element>
<element name="CustomerInfoResponse">
  ...
  <element name="name" type="string"/>
  <element name="phone" type="string"/>
  <element name="street1" type="string"/>
  <element name="street2" type="string"/>
  <element name="city" type="string"/>
  <element name="state" type="string"/>
  <element name="postalcode" type="string"/>
  <element name="country" type="string"/>
  ...
</element>
```

```
<element name="CustomerInfoRequest">
  ...
  <element name="account" type="string"/>
  ...
</element>
<element name="CustomerInfoResponse">
  ...
  <element name="name" type="string"/>
  <element name="phone" type="string"/>
  <element name="street1" type="string"/>
  <element name="street2" type="string"/>
  <element name="city" type="string"/>
  <element name="state" type="string"/>
  <element name="postalcode" type="string"/>
  <element name="country" type="string"/>
  ...
</element>
```

```
<m:GetCustomerInfo ...>
  <account>1069>/account>
</m:GetCustomerInfo>
```

```
<m:GetCustomerInfoResponse ...>
  <name>GIS-Centras, SE</name>
  <phone>370-275-0122</phone>
  <street1 >Savanorių 187</street1 >
  <street2></street2>
  <city>Vilnius</city>
  <municipality>VL</ municipality>
  <postalcode>LT-02300</postalcode>
  <country>Lietuva</country>
</m:GetCustomerInfoResponse>
```

Service broker
Directory
Service description using WSDL
Query responses using WSDL
Directory queries
Service provider
Service consumer
XML service request based on WSDL
XML service response based on WSDL

Figure: The messaging using Web Services within SOA (http://www.service-architecture.com/)

Similar technologies (WSDL, SOAP over HTTP, UDDI) can be used to implement and maintain Web map services. These services can give access to a wide range of mapping and GIS capabilities. For example, ArcWeb Services, hosted by ESRI, allows one to add mapping and spatial functionality to client applications in a service-oriented architecture. These services may include data visualization, accessed data including routing, place finding, address finding, spatial query searches, and business intelligence performance (http://www.arcwebservices.com/v2006/help/index.htm).

ESRI's ArcGIS Server also provides the foundation for geospatially enabling a service-oriented architecture. ArcGIS Server offers spatial data visualization, spatial data analysis, and data management services. These services can also be called upon and integrated with other Web services using standard Web services protocols, such as SOAP and XML (http://webhelp.esri.com/arcgisserver/9.2/).

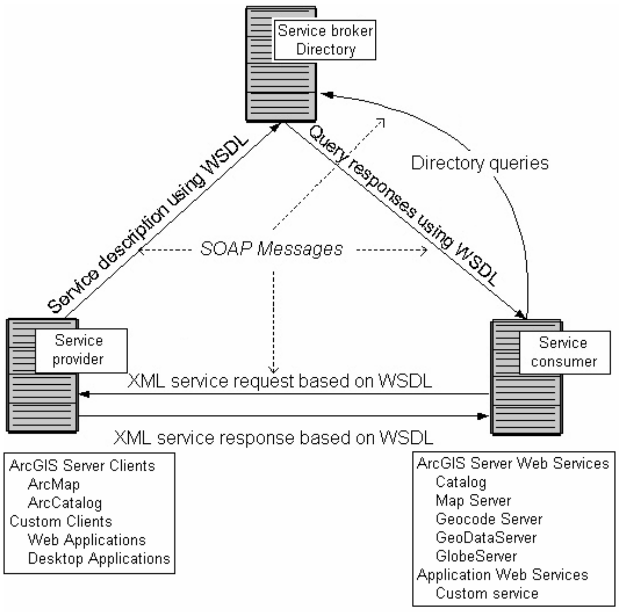


Figure: ESRI ArcWeb Services

New improvement in web technologies, for example the so-called ***Web 2.0***, is also applied for geospatial data and services development. Web 2.0 provides enhancements over read-only websites and includes a social element where users generate and distribute content.

Web 2.0 refers to a "second generation" of web sites, primarily distinguished by the ability of visitors to contribute information for collaboration and sharing. Web 2.0 applications use Web services and may include Ajax, Flash (http://labs.adobe.com/), or JavaFX (http://www.sun.com/software/javafx/index.jsp) user interfaces, web syndication, RSS feeds (see Conclusion section), blogs, and wikis. Web syndication refers to making web feeds available from a site in order to provide other people with a summary of the website's recently added content. A web feed is often a XML-based document that contains content items with web links to longer versions. A blog or web log is a website where entries can be made in order to maintain or add content, and are commonly displayed in reverse chronological order. A wiki is software that allows users to create, edit, and link web pages easily. Wikis are often used to create collaborative websites and to power community websites.

While there are no set standards for Web 2.0, it is characterized by building on the existing web server architecture and using services. Web 2.0 can, therefore, be regarded as displaying some SOA characteristics.

Mashups are also regarded by some as Web 2.0 applications. Mashups currently come in three general types: consumer mashups, data mashups, and business mashups. The best-known sample of consumer mashup is Google Maps applications. Consumer mashups combine data elements from multiple sources, hiding this behind a simple unified graphical interface. Other common types are “data mashups” and “enterprise mashups”. A data mashup mixes data of similar types from different sources (e.g., Yahoo Pipes), as for example, combining the data from multiple RSS feeds into a single feed with a graphical front end. An enterprise mashup usually integrates data from internal and external sources. For example, it could create a real estate report by combining an external list of all houses sold on the market with internal data about which houses one agency sold. A business mashup is a combination of all the above, focusing on both data aggregation and presentation, and additionally adding collaborative functionality, making the end result suitable for use as a business application.

4.7. Conclusion

The Extensible Markup Language (XML) is used to implement application languages that can serve different purposes. Thus, geospatial applications can use the following XML grammar languages:

- Geographic Markup Language (GML) is used for:
  - Encoding geographic information
  - Transporting and storage geographic information
  - Modeling application schemas of spatial data

- ESRI Arc Extensible Markup Language (ArcXML) is used for:
  - Defining rendering of map content, location of spatial data, and local environment for ESRI mapping services
  - Encoding request and respond messages within ESRI ArcIMS architecture

- Google Keyhole Markup Language (KLM) is used for:
  - Invoking Google Maps web mapping service
  - Mashuping user-defined features on the top of Google map

- Scalable Vector Graphics (SVG) is used for:
  - Displaying two-dimensional vector graphics including in a web browser

- Simple Object Access Protocol (SOAP) messages are used for:
  - Exchanging complete documents or to call a remote procedure between web applications (e.g. within SOA)

- Web Services Description Language (WSDL) is used for:
  - Defining web services including both data type definitions and message definitions, web service operations, and bindings

- The programming interface for Universal Description Discovery and Integration (UDDI) specification is based on XML

- Geographic Really Simple Syndication (GeoRSS) is used for:
  - RSS or "Web feed" is a family of XML formats for exchanging news, especially news about Web pages or other Web content (http://www.rssboard.org/rss-specification). Many dynamic web sites, especially "*blogs*", provide RSS "*feeds*" of their new or changed content. The basic idea is simple structured XML formats that includes only key descriptive elements like author, date, title, narrative description, and hypertext link - elements that help a reader or an RSS “aggregator” service decide what source materials are worth examining in more detail.
  - GeoRSS (http://www.georss.org/) is a simple proposal for RSS feeds described by location or geo-tagged. GeoRSS feeds can be used for a geographic search and aggregation. Within GeoRSS, a location is described in an interoperable manner so that applications can request, aggregate, share, and map geographically tagged feeds. More than just getting feeds for a particular city or zip code, by using GeoRSS it is possible to search all sorts of geographic criteria.

- There are currently two encodings of GeoRSS: GeoRSS-Simple, and GeoRSS-GML. GeoRSS-Simple supports basic geometries (point, line, box, polygon) and covers the typical use cases when encoding locations. GeoRSS GML is a formal GML Application Profile and supports a greater range of features and coordinate reference systems other than WGS-84 latitude/longitude. Both formats are designed for use with Atom 1.0 (http://en.wikipedia.org/wiki/Atom_%28standard%29), RSS 2.0, and RSS 1.0, although it can be used just as easily in non-RSS XML encodings.

Encode a point in the GeoRSS-Simple looks like:

```
<georss:point>25.258197 54.680445</georss:point>
```

In the GeoRSS GML version:

```
<georss:where>
   <gml:Point>
      <gml:pos>25.258197 54.680445</gml:pos>
   </gml:Point>
</georss:where>
```

Module self-study questions:

[1] What is the focus of GML encoding?
[2] What are the differences between GML and ArcXML?
[3] What communication language is used for OGC WMS?
[4] How are spatial data encoded for OGC WFS?
[5] How are spatial data encoded for ESRI ArcIMS Feature Server?
[6] What are the three building blocks of SOA? What are the means of connecting services to each other within SOA?
[7] What is “mashup” and how is it used in Web 2.0 mapping technology?

Recommended Readings:

[1] OpenGIS Geography Markup Language (GML) Encoding Standard, http://portal.opengeospatial.org/files/?artifact_id=20509
[2] Ho, M., Extending ArcIMS MapServices with ArcXML, ESRI Educational Services, http://www.esri.com/news/arcuser/0102/extend1of3.html
[3] SVG Tutorial, W3 School, http://www.w3schools.com/svg/
[4] SOAP Tutorial, W3 School, http://www.w3schools.com/soap/
[5] WSDL Tutorial, W3 School, http://www.w3schools.com/wsdl/
[6] Web Services Tutorial, W3 School, http://www.w3schools.com/webservices/
[7] Web Services and Service-Oriented Architectures, http://www.service-architecture.com/

References

[1] OpenGIS Geography Markup Language (GML) Encoding Standard, http://portal.opengeospatial.org/files/?artifact_id=20509
[2] ArcXML Programmer's Reference Guide for ArcIMS 9.2, http://edndoc.esri.com/arcims/9.2/, http://webhelp.esri.com/arcims/9.2/general/ or http://support.esri.com/index.cfm?fa=knowledgebase.webHelp.arcIMSGateway
[3] KML in Google Maps, http://code.google.com/apis/kml/documentation/mapsSupport.html
[4] Scalable Vector Graphics (SVG), http://www.w3.org/Graphics/SVG/
[5] Web Services Description Language (WSDL), http://www.w3.org/2002/ws/desc/
[6] SOAP, http://www.w3.org/TR/2007/REC-soap12-part0-20070427/
[7] OASIS UDDI Specification TC, http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec

Terms used

- XML grammar application
- Geographic Markup Language
- Encoding
- Arc Extensible Markup Language
- AXL Map Configuration File
- Google Keyhole Markup Language
- Service-Oriented Architecture
- Mashup
- Scalable Vector Graphics
- Scalable
- Service
- Universal Description Discovery and Integration
- Web Services Description Language
- Simple Object Access Protocol
- Web syndication
- Blogs
- Wikis
- Web 2.0
- Geographic Really Simple Syndication
- Feed

5. ESRI Web Application Architectures

In previous modules, web mapping architecture, its components, and web languages for user interface, communication, data, and schema descriptions were discussed. In this module, previously discussed topics are presented by using implemented industrial examples – ESRI ArcIMS and ArcGIS Server. As mentioned in Module 3, ESRI was a pioneer in the development of web mapping software. ESRI products were chosen for analytical purposes because they are supplemented by documentation that is easy to use.

ESRI ArcIMS uses Internet Map Server (IMS) software for authoring, designing, publishing, and administering Internet mapping applications. It provides a solution for delivering dynamic maps and GIS data and services via the Web. ArcIMS allows web clients, map servers, data servers, and the web server to communicate with one another.

This new ESRI product, ArcGIS Server, also provides functionality for web mapping applications. In addition, it supports rich geoprocessing functionality over the web interface. In addition, ArcGIS Server provides the foundation for geospatially enabling a service-oriented architecture (SOA).

Module Outline

5.1. What is ESRI ArcIMS and ArcGIS Server?

As mentioned in Module 3, ESRI was a pioneer in the development of web mapping software. Currently, ESRI web mapping and GIS services are supported by ArcIMS and ArcGIS Server software. As noted in Module 3, web services are published on a web server and client applications can request capabilities of, and consume, published services when accessing the web map servers. Client and web servers are loosely connected whereby each client communication represents a complete transaction. Transactions are processed in the form of a request to the appropriate web-based map or GIS server and a response is returned to the client.

ArcIMS is an Internet mapping solution and a framework for distributing map (and basic GIS) capabilities via the Internet. As a serving technology, ArcIMS includes several of ESRI *server* side technologies (map engines) that support deployment of map and metadata services on the web. As a publishing technology, ArcIMS features capabilities for supporting a wide variety of *clients*. ArcIMS can also host OGC WMS and WFS services.

ArcGIS Server can host not only web mapping and metadata services (ESRI and WMS), but also rich geoprocessing and other services, that bring ArcGIS Server into the category of a GIS web server, not only a web mapping and geocoding server. ArcGIS Server can be deployed in a web architecture or as a LAN/WAN network service for desktop clients. ArcGIS Server is also used to deploy "smart clients" that is a loosely connected, lightweight, handheld, or desktop computer that supports persistent data cache and disconnected GIS client operations. Client application deployment and data synchronization are managed by ArcGIS Server parent services.

Components for ArcIMS and ArcGIS Server can be implemented within the multi-tiered software architecture that was discussed in Module 3. These components have different names and functionality, but logically they perform similar functional roles and are supported by common platform configuration strategies. In general, these applications consist of three tiers, described as the client or presentation tier, the business logic tier, and the data storage tier. These tiers describe logical groupings of the functionality of the various application components and do not necessarily correspond to their physical location.

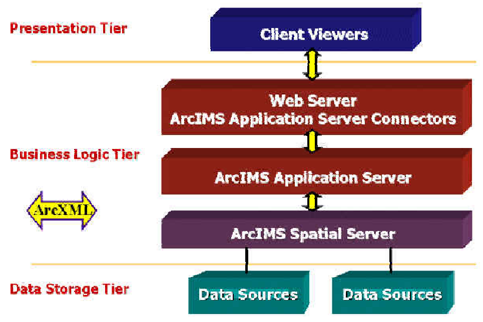
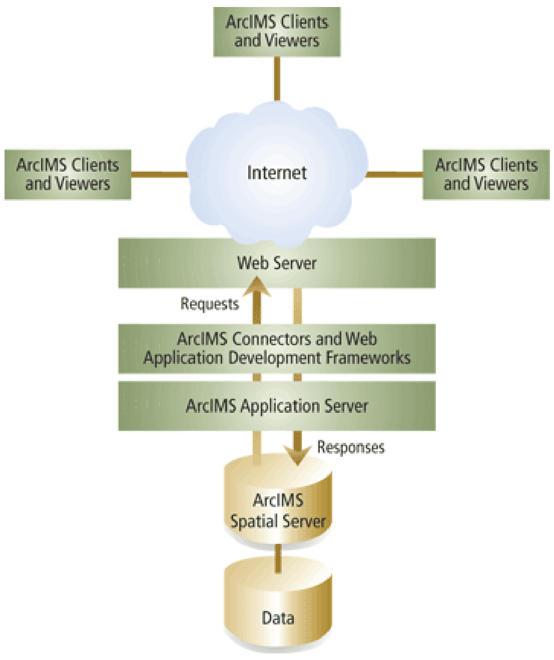


Figure: ArcIMS consists of three tiers (http://www.esri.com/software/arcgis/arcims/)

The *presentation tier* consists of client side components that are used to send requests to the server and to view the maps and data.

The *business logic tier* consists of the server side components. Both software solutions include web application (WA), service manager (SM), and spatial server (SS) layers that can be deployed on different platform combinations. This tier is where all of the request processing and server administration takes place. Location of the various software components and the selected software configuration can impact system performance.

The *data storage tier* (DS) consists of file servers that hold the shape files and image files, as well as the server running Oracle (or other RDBMS) and SDE. The spatial server can also be considered part of the data storage tier.


ArcIMS Clients and Viewers
ArcIMS Clients and Viewers
Internet
ArcIMS Clients and Viewers
Web Server
Requests
ArcIMS Connectors and Web Application Development Frameworks
ArcIMS Application Server
Responses
ArcIMS Spatial Server
Data

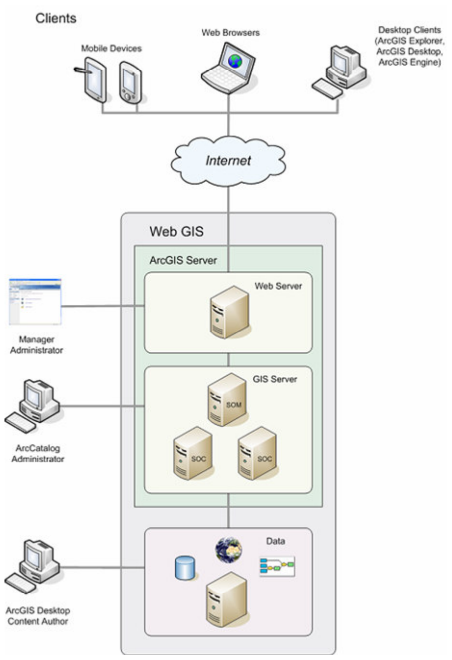
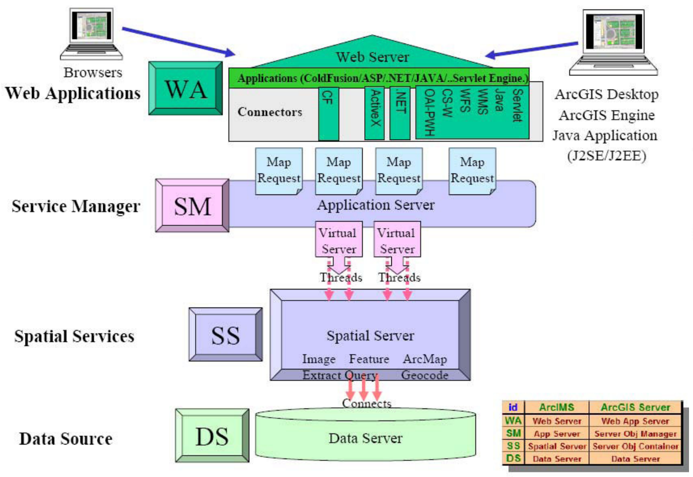


Figure: The components of ArcIMS (http://webhelp.esri.com/arcims/9.2/general/)

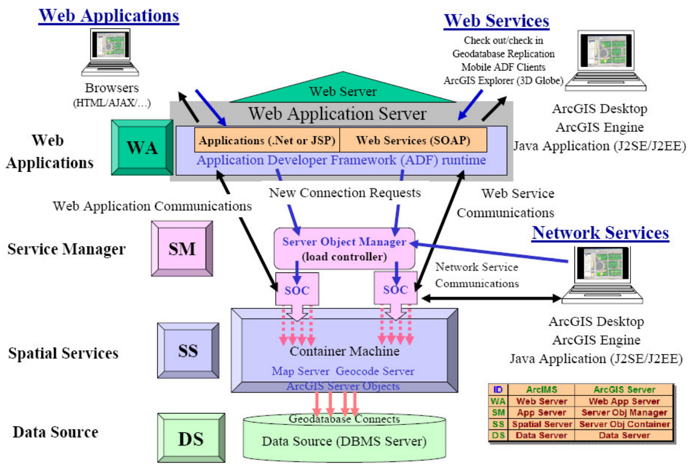
Figure: The components of ArcGIS Server (http://webhelp.esri.com/arcims/9.2/general/)

5.2. ESRI Web Components: Server Side

In this section, the component of ESRI web mapping application, such as ArcIMS, will be discussed. As mentioned above, in a logical scene, the web mapping application of ArcGIS Server performs similar functions as ArcIMS.



id	ArcIMS	ArcGIS Server
WA	Web Server	Web App Server
SM	App Server	Server Obj Manager
SS	Spatial Server	Server Obj Container
DS	Data Server	Data Server



ID	ArcIMS	ArcGIS Server
WA	Web Server	Web App Server
SM	App Server	Server Obj Manager
SS	Spatial Server	Server Obj Container
DS	Data Server	Data Server

Figure: The components of ArcIMS (http://www.esri.com/library/whitepapers/pdfs/sysdesig.pdf)

Figure: The components of ArcGIS Server (http://www.esri.com/library/whitepapers/pdfs/sysdesig.pdf)

How does the viewer and server communicate? The ArcIMS client makes a request, which is sent to the *Web Server*, passed through the *Connectors* to the *Application Server* and then to the appropriate *Virtual Server*, which hands it to one of the *Spatial Servers*. All requests and responses are written or translated in ArcXML. According to this flow, the ArcIMS components include the following:

- ArcIMS *Web Applications* layer includes the *web* HTTP *server* and the web applications server *connector*.
  - ***Web Server*** is a component that communicates directly with the client, and is required in order for ArcIMS to work. It is the computer program that stores and serves requested HTML pages or files. A web server has to be capable of servicing Hypertext Transfer Protocol (HTTP) requests. For example, the web browser is a client that requests HTML files from web servers.

    Web servers are external software components to ArcIMS and not included with ArcIMS installation packages. Different web servers can be used, including Microsoft Internet Information Server (IIS), Netscape Enterprise Server, iPlanet, and Apache.

  - ArcIMS Application Server ***Connectors*** allows the web server to communicate with the ArcIMS Application Server, and also provides a communication pipeline between a web server or third party application server (e.g. ColdFusion) and the ArcIMS Application Server. Their job is to translate a request from another language into ArcXML before it is sent to the Application Server. It can be used to manage and enhance (via customization) user workflow and client display presentation.

As ArcIMS uses ArcXML for communication between the ArcIMS Application Server and Spatial Servers, a mechanism is needed to translate requests from HTTP/XML standards-based protocols or other development environments (e.g. cold fusion pages, asp) into ArcXML. This translation is done through Application Server *connectors* and Web Application Development Frameworks (Web ADF). The connectors and Web ADFs reside between the web server with Servlet Engine and ArcIMS Application Server, or between a third party application server (e.g. Cold fusion) and the ArcIMS Application Server.

ArcIMS supports several development environments and can process requests from some common standards-based protocols. ArcIMS includes few connectors (see figure below). The ArcIMS Servlet Connector is the default connector for ArcIMS. All Server Connectors must be installed on the same machine as the Web Server.

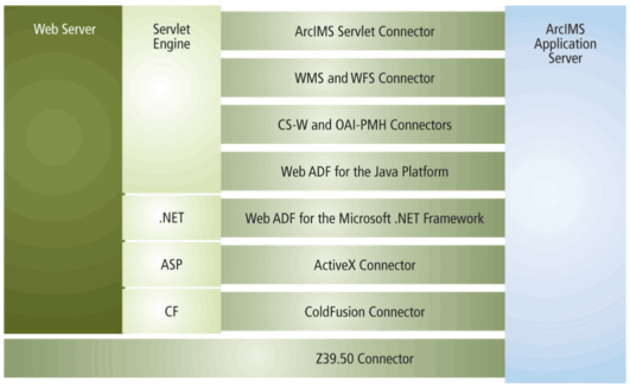


Figure: ArcIMS Application Server *connectors* (http://webhelp.esri.com/arcims/9.2/)

ArcIMS also includes connectors that adhere to Open Geospatial Consortium, Inc. specifications. The WMS Connector, WFS Connector, and CS-W Connector (OGC Standard Catalog Service 2.0 to ArcIMS Metadata Services) are available. A translation engine receives OGC-standard WMS requests and converts them into ArcXML requests that are forwarded to the published ArcIMS services. The ArcXML responses are translated back into OGC standard responses and are returned to the requesting clients. The WFS Connector allows using GML to stream feature and attribute information to clients.

- ArcIMS ***Spatial Server*** has to communicate with the Application Server, parse ArcXML requests into their components, read in shapefile, image and SDE data from other servers, and process the actual request. When a request is received, the Spatial Server performs one or more functions:

  - *Image* function generates map image file from map created using ArcIMS Author (logically correspond to WMS)
  - *ArcMap Image* function generates map image file from map created using ArcMap (logically correspond to WMS)
  - *Feature* function streams map features (logically correspond to WFS)

- *Query* function searches for features matching search criteria
- *Geocode* function performs address-matching operations
- *Extract* function creates shapefiles from selected map features
- *Metadata* function publishes and searches metadata
- *Route* function calculates routes between a set of two or more stops when using the optional ESRI RouteServer extension
- *SDCGeocode* function performs address-matching operations on RouteServer data.

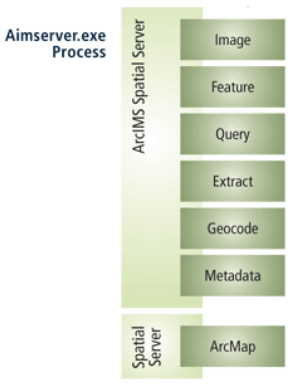


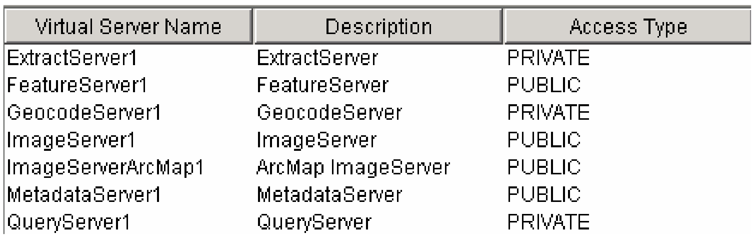
Figure: ArcIMS Spatial Server (aimsserver.exe) and its components – functions (http://webhelp.esri.com/arcims/9.2/)

- The map's contents are published over the Internet or Intranet via ArcIMS through one or more ArcIMS services. ArcIMS ***Services*** are processes that continuously run on the ArcIMS Spatial Server and perform a specified operation at predefined times or in response to certain events. An ArcIMS service presents spatial data and/or metadata to users via a web server.

  There are four ArcIMS service types: Image Service and ArcMap Image Service (logically corresponds to OGC WMS for a lightweight thin client), Feature Service (logically corresponds to OGC WFS - features are streaming in a temporary ESRI compressed format or in GML for thick client) and Metadata Service (lets to share metadata).

- It is possible to have multiple Spatial Servers on multiple machines, and a mechanism is needed to manage these Spatial Servers and the services running on them. ArcIMS ***Virtual Server*** is a mechanism that groups one or more Spatial Servers, and its purpose is to administer and manage the load distribution among the Spatial Servers.

  Virtual Server is not an actual server. An ArcIMS Service is assigned to a Virtual Server: when a service is started, it must be assigned to an Image, ArcMap, Feature, Metadata Virtual, or other Virtual Server rather than directly to an individual ArcIMS Spatial Server. The service starts on all instances within the Virtual Server group. ArcIMS uses a Virtual Server concept to manage a site. Thus, ArcIMS services on one or more Spatial Servers can be linked together for one delivery to a client.



Virtual Server Name	Description	Access Type
ExtractServer1	ExtractServer	PRIVATE
FeatureServer1	FeatureServer	PUBLIC
GeocodeServer1	GeocodeServer	PRIVATE
ImageServer1	ImageServer	PUBLIC
ImageServerArcMap1	ArcMap ImageServer	PUBLIC
MetadataServer1	MetadataServer	PUBLIC
QueryServer1	QueryServer	PRIVATE

Figure: Virtual Server types

Grouping ArcIMS Spatial Servers is important not only for administration but also for reliability. If an ArcIMS Spatial Server goes down, incoming requests can still be handled by other Spatial Servers assigned to the same Virtual Server.

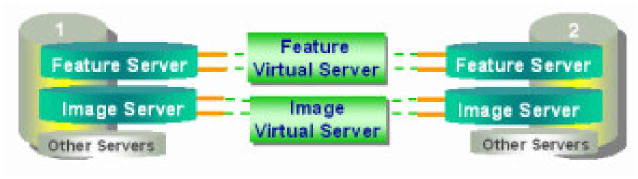
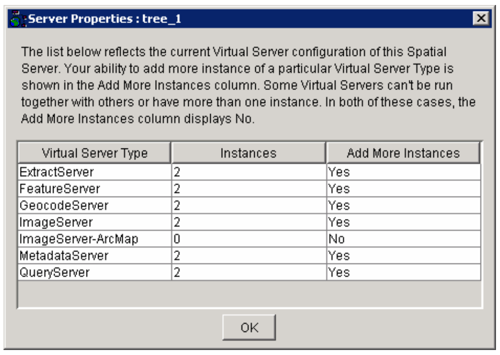


Figure: A Feature Virtual Server groups the instances from the Feature Server of two Spatial Servers. Likewise, an Image Virtual Server groups the instances of the two Image Servers.

- A Spatial Server is a process that initializes and manages one or more server ***instances*** (threads). ArcIMS, by default, assigns two instances per Virtual Server process.



Server Properties : tree_1

The list below reflects the current Virtual Server configuration of this Spatial Server. Your ability to add more instance of a particular Virtual Server Type is shown in the Add More Instances column. Some Virtual Servers can't be run together with others or have more than one instance. In both of these cases, the Add More Instances column displays No.

Virtual Server Type	Instances	Add More Instances
ExtractServer	2	Yes
FeatureServer	2	Yes
GeocodeServer	2	Yes
ImageServer	2	Yes
ImageServer-ArcMap	0	No
MetadataServer	2	Yes
QueryServer	2	Yes

OK

Figure: By default, ArcIMS assigns two instances per Virtual Server process. The exception is for Spatial Servers hosting ArcMap Image Services. These Spatial Servers are assigned one instance each.

The number of instances equates to the number of requests that can be handled simultaneously. For example, if a Virtual Server consists of two Spatial Servers each with two instances, then that Virtual Server can handle four simultaneous requests.

- The ArcIMS ***Service Manager*** components provide a service management role that supports the inbound map service request queues (Virtual Servers) and configured connects to the ArcIMS public service engines (Image, ArcMap, Feature, Extract). Inbound requests are routed to available service instances for processing. A relatively small amount of processing is required to support the application server functions.

  - ArcIMS ***Application Server*** is used as a background process that keeps track of which services are running on which Virtual Server and handles the load distribution of incoming requests. Within each Virtual Server, the Application Server keeps track of which Spatial Server instances are associated with the Virtual Server. Using this information, the Application Server dispatches an incoming request to the appropriate Spatial Server instance. When the number of requests exceeds the number of available instances, they are stored in the Application Server's queue. The Application Server sends the request to the appropriate instance as the instances become available.

The Application Server does not have to be on the same machine as the Web Server.

- ArcIMS ***Monitor*** is a background process that tracks the state of the Spatial Servers and starts or restores ArcIMS Services. The Application Server and Spatial Server processes communicate directly with each other. When a Spatial Server process stops communicating with the Application Server, the Application Server instructs the Monitor to restart the Spatial Server process. In a standard configuration, each machine containing one or more Spatial Server processes has one Monitor.

- ArcIMS ***Tasker*** is a background process that watches output files generated by the Spatial Server and periodically deletes the output image files on a user-specified interval.

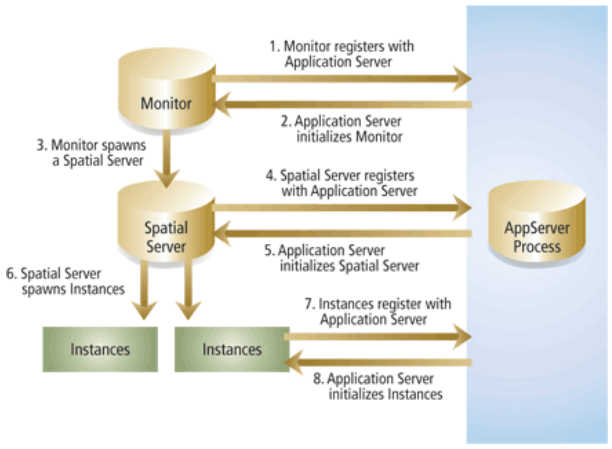

1. Monitor registers with Application Server
Monitor
2. Application Server initializes Monitor
3. Monitor spawns a Spatial Server
4. Spatial Server registers with Application Server
Spatial Server
AppServer Process
5. Application Server initializes Spatial Server
6. Spatial Server spawns Instances
7. Instances register with Application Server
Instances
Instances
8. Application Server initializes Instances

Figure: The Application Server, Monitor, and Spatial Server processes work together to communicate, start services, and process requests (http://webhelp.esri.com/arcims/9.2/)

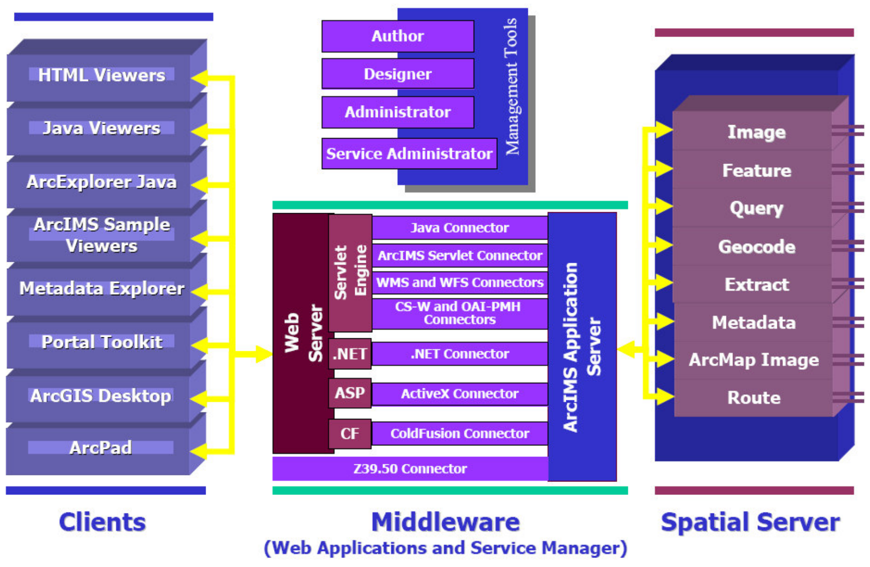


Figure: The overview of the ArcIMS 9.2 component architecture (http://www.esri.com/library/whitepapers/pdfs/sysdesig.pdf)

- ***Data Source*** is the data of file server and is where the GIS data is stored. An ArcSDE data source supports the query processing functions. Standard GIS image or file data sources are also represented at this level.

ArcSDE Connects
Data Source (ArcSDE Server)

Figure: The sample of GIS data source tier

5.3. ESRI ArcIMS User Interface Components: Server Side

One advantage of ArcIMS is that it provides resources and tools for creating and administering web applications. The following ArcIMS tools are available:

- ArcIMS ***Author*** standalone application is used to create a map .axl configuration file that represents a map. This AXL file is used as input to an Image Service or Feature Service. Author allows one to add data layers, set layer symbology, create stored queries, and set layers for geocoding within an .axl configuration file. A configuration file contains instructions about what data will be made available, the actual paths to that data and log in information, and the exact appearance of that data.

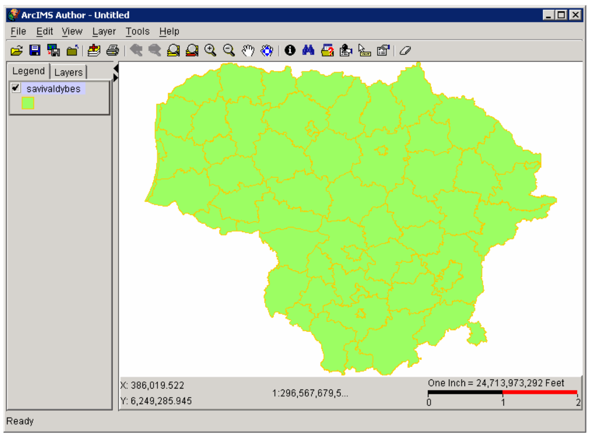


Figure: ArcIMS Author

Map configuration files are written in ArcXML - an XML extension to ArcIMS (see Module 4 for details). Author functionality does not permit one to add all the ArcXML tags available for use in a map configuration file. In this regard, .axl files can be edited as any XML documents and additional functionality (e.g. map projection information that can be added manually).

An ArcMap document (.mxd) file, created in ArcMap, is used as input to an ArcMap Image Service.

Table: Authoring map for ArcIMS processes

Process	**Service type**			
	Image	**Feature**	**ArcMap Image**	**Metadata**
Map creation product	Author	Author	ArcMap	XML Editor
Configuration file format	ArcXML	ArcXML	Binary ArcMap document in .mxd or.pmf	ArcXML
Clients	HTML Viewer, Java Viewers ArcMap, ArcPad, Custom HTML application	Java Viewers, ArcExplorer, ArcMap	HTML Viewer, Java Viewers ArcMap, ArcPad	Metadata Viewer, ArcCatalog

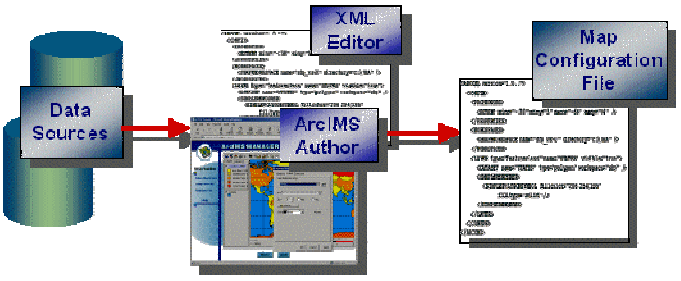


Figure: Process for Generating a Map File for Image and Feature Services

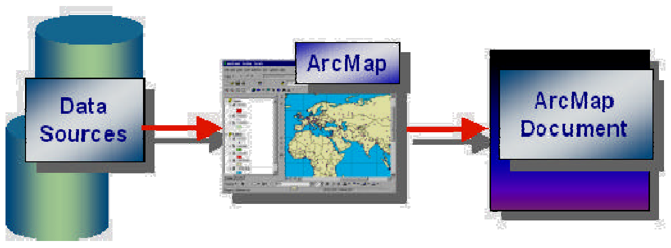


Figure: Process for Generating a Configuration File for ArcMap Image Services

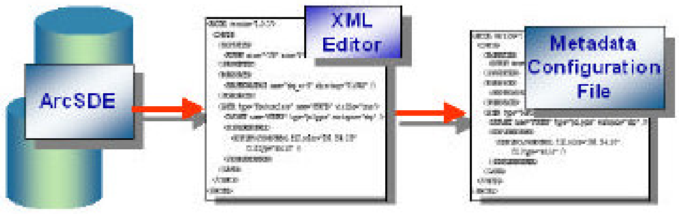
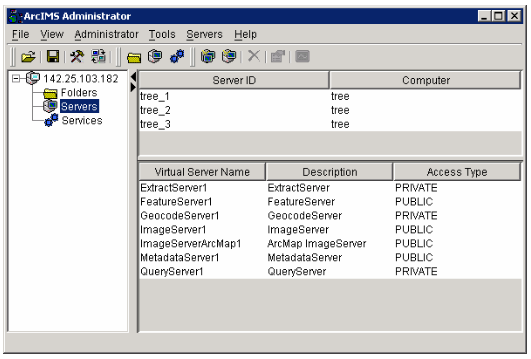


Figure: Authoring metadata configuration files for Metadata Services.

- ArcIMS ***Administrator,*** as a standalone application, or ***Service Administrator*** as a web application or command line administration, can be used to create and manage ArcIMS services, servers, and folders.



Server ID	Computer
tree_1	tree
tree_2	tree
tree_3	tree

Virtual Server Name	Description	Access Type
ExtractServer1	ExtractServer	PRIVATE
FeatureServer1	FeatureServer	PUBLIC
GeocodeServer1	GeocodeServer	PRIVATE
ImageServer1	ImageServer	PUBLIC
ImageServerArcMap1	ArcMap ImageServer	PUBLIC
MetadataServer1	MetadataServer	PUBLIC
QueryServer1	QueryServer	PRIVATE

Figure: ArcIMS Administrator

The Administrator or Service Administrator creates, starts, stops, saves, and removes services. It can also set and change service properties, such as the Virtual Server, Spatial Servers, or Folders assignment Virtual Servers. For example, the Spatial Server processes are handled either by starting and stopping Monitor or by using Administrator or Service Administrator.

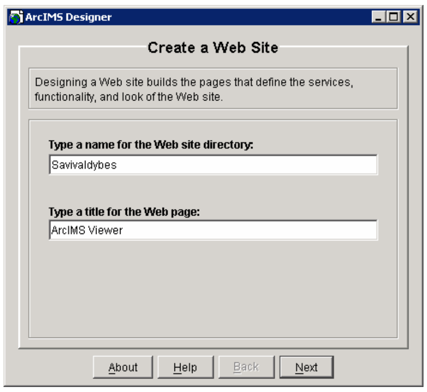
An .axl file is used to create an ArcIMS map service. Although the ArcIMS Service and the .axl file are closely related, they are completely independent. One .axl can define multiple ArcIMS Services, and once the ArcIMS Service is created, the .axl file can be deleted or modified without affecting the ArcIMS Service.

- ArcIMS ***Web Manager*** and ***Designer*** can be used to create an actual web site that is a folder with the collection of HTML, DHTML, and JavaScript files. An ArcIMS web site can include of one or more services, which were created with Administrator, and it may also contain a legend, tools for navigating the map, a scale bar, and an overview map.

ArcIMS *Designer* is a wizard-driven application that guides through the web site design process. Designer outputs three viewers: HTML Viewer for standard browser and corresponds to WMS, Java Custom Viewer, and Java Standard Viewer for standard browser with ESRI add-on and corresponds to WFS.

The new ArcIMS 9.2 *Web Manager* also provides step-by-step instructions that guides one through the web site building process. It has more functionality compared to *Designer*. Not only Image and Feature Services can be selected during the design, but ArcGIS Server services, ArcWeb services, and WMS services are also available. The Web Manager can merge the few user-selected services. Also, Web Manager allows the possibility to select more functionalities, for example, to add the finding task such as addresses application or searching for features; to add map elements such as a table of contents or overview map; to configure a header page elements for a web site; etc.

The web mapping application, which was built with Web Manager, uses .Net, Java and AJAX technology that include seamless panning, scroll-wheel zoom, keyboard navigation, and a fully functional table of contents with swatches, among other new features.


ArcIMS Designer

Create a Web Site

Designing a Web site builds the pages that define the services, functionality, and look of the Web site.

Type a name for the Web site directory: Savivaldybes

Type a title for the Web page: ArcIMS Viewer

Figure: ArcIMS Designer

In addition, ArcIMS provides a number of resources to get started with web site templates. Thus, the HTML viewer templates Web Sites are installed automatically with ESRI ArcIMS, and defines the graphical look and functionality of each user's ArcIMS Web site. The viewers provide a framework for the map, toolbar, legend, overview map, and other graphic elements for the site. If a web site is developed with the Web Manager, the Web Mapping Application template is ready for deployment with the Web Manager.

The following overview table summarizes how a web mapping application site can be created with ArcIMS manager applications

Table: The steps for creating a ArcIMS site (http://webhelp.esri.com/arcims/9.2/)

1	Map	A map and respective map configuration file for a web application are created by using, Author, ArcMap or manually. Map configuration file can be optimized for well performance.
2	Service	A created map has to must publish as a service with Administrator or Service Administrator.
3	Web App	A web application site is created with Designer or Web Manager by using available services. Customization can be applied.
4	Tune	Created web site can be tuned and monitored in order to meet for performance and security requirements.

ArcIMS manager applications such as Author, Designer, and Administrator can be installed on a machine other than the ArcIMS host for a remote web site creation and administration. In addition, Service Administrator can be used for multi-hosts' administration from a remote machine. The Service Administrator will be used to administer the few ArcIMS Application Servers.

5.4. ESRI ArcIMS Components: Client Side

The ArcIMS **Client Viewer** is a component of ArcIMS that can make specific requests to the server in order to get a desired response, for example, such as a map image or the tabular results of a query. The client also has to know how to handle the ArcXML response from the server (e.g., formatting the ArcXML stream into an HTML attribute table).

ArcIMS supports the following main types of Viewers:

- Thin lightweight clients such as HTML/DHTML web browsers (e.g., Internet Explorer or Netscape)
- Thick clients can be deployed either as a stand-alone application (e.g., ESRI ArcExplorer), or as an applet in a web browser (e.g., ESRI Java Viewer applet that can be only sent to the client's machine once, after the first download from a web mapping site. Also, the Java Runtime Environment (JRE) will be required to be installed on the client machine)
- Custom thick clients written in a COM compliant language (such as VB, .NET, J2EE, etc.)
- Thick clients such as ArcGIS applications (e.g., ArcMap and ArcPad)

5.5. ESRI ArcIMS Communication Flow

Communication between client and ArcIMS Server is loose in that each client communication represents a complete transaction. Transactions are processed in the form of a request to the appropriate ArcIMS Server and a response is returned to the client.

The sequence of communication between an ArcIMS client and sever is presented in the following table and figure:

Table: Steps of an ArcIMS client and server communication cycle

Steps	Actions
2.	ArcIMS client generates a request to an ArcIMS site. The request queries the ArcIMS Service. Example queries include request for a new map at a different scale, returning the attribute information for a feature, changing the rendering of a layer, or turning layers on and off, etc. `POST` and `GET` HHTP request methods can be used to send request to an ArcIMS server.
3.	The Web Server receives the request and passes it to the Server Connector or third party application server, which turn hand the request to a connector.
4.	The connecter opens a path for the ArcIMS Application Server respond and the request is handed from the connecter to the Application Server.

5.	The Application Server sends the request to an available Spatial Saver within a Virtual Server group. The ArcIMS Spatial Server receives a ArcXML `<REQUEST>` document that can include the following types of ArcIMS or/and OGC requests (see ArcXML Programmer's Reference Guide for ArcIMS 9.2 at the http://webhelp.esri.com/arcims/9.2/general/ for details):
	
	▪ `GET_IMAGE`
	▪ `GET_FEATURES`
	▪ `GET_GEOCODE`
	▪ `GET_EXTRACT`
	▪ `GET_SERVICE_INFO`
	▪ `GET_RASTER_INFO`
	▪ `GET_LAYOUT`
	▪ `GET_Project`
	▪ `GET_METADATA`
	▪ `PUBLISH_METADATA`
	▪ `GetCapabilities`
	▪ `GetMap`
	▪ `GetFeatureInfo`
	▪ `GetCapabilities`
	▪ `DescribeFeatureType`
	▪ `GetFeature`
	
	A sample of request can be:
	
	`<?xml version="1.0" encoding="UTF-8" ?>`
	`<ARCXML version="1.1">`
	`<REQUEST>`
	`          <GET_IMAGE>`
	`                    <PROPERTIES>`

```
                                      <ENVELOPE   minx="-125"   miny="25"   maxx="-67"
maxy="50" />
           </PROPERTIES>
           </GET_IMAGE>
</REQUEST>
</ARCXML>
```

6. The Spatial Saver generates the response as either a XML string (such as query results or an image location) for image or WMS request, or stream of data for a Feature or WFS request.

Each ArcIMS type of *request* has its respective the Spatial Server type of *response*, for example:

- `GET_EXTRACT` response is `EXTRACT`
- `GET_FEATURES` response is `FEATURES`
- `GET_GEOCODE` response is `GEOCODE`
- `GET_IMAGE` response is `IMAGE`
- Etc

In the case of `GET_IMAGE` request, the Spatial Server uses ArcXML `<CONFIG>` .axl document of the respective image server to create the map image (e.g. in gif format) in its output directory. It then sends an ArcXML `<RESPONSE>` document to the client with the name of that image and a URL that the client can use to access the image. A sample of response could be:

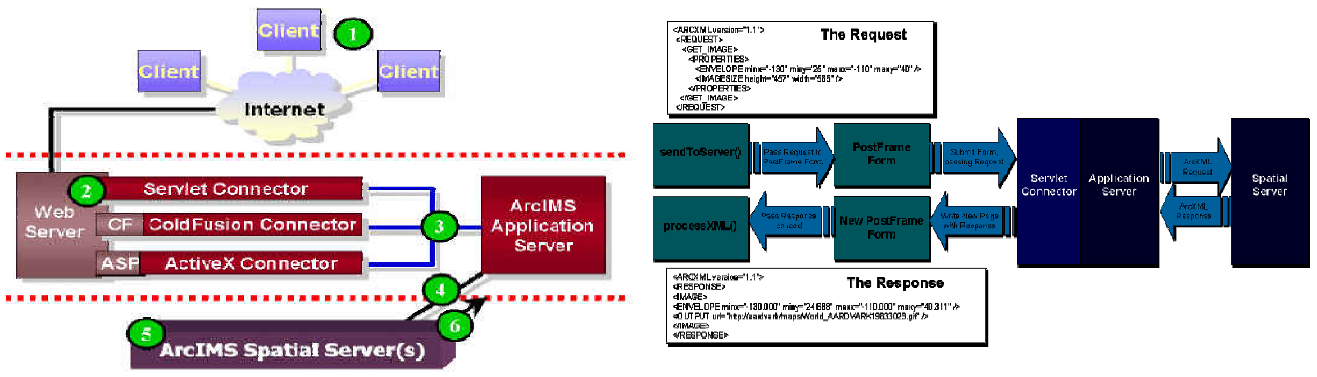
```
<?xml version="1.0" encoding="UTF-8" ?>
<ARCXML version="1.1">
           <RESPONSE>
            <IMAGE>      <ENVELOPE    minx="-87.5"    miny="30.0"    maxx="-59.5"
            maxy="50.0" />
            <OUTPUT                          url="http://tree.mala.bc.ca/output/
            savivaldybes5681824311.jpg" />
                      </IMAGE>
           </RESPONSE>
</ARCXML>
```

An example of a map configuration file is shown below. The initial extent of the map's view, the map units of used data, the path to a shapefile, and then the name of the shapefile and its symbology for representation are defined within this file.

```
<ARCXML version="1.0.1">
 <CONFIG>
   <MAP>
     <PROPERTIES>
       <ENVELOPE minx="1205720.8" miny="21567.0" maxx="1585746.9"
maxy="307341.899475" name="Initial_Extent"/>
       <MAPUNITS units="METERS"/>
       <BACKGROUND color="0,0,210" transcolor="255,255,255"/>
     </PROPERTIES>
  <WORKSPACES>
     <SHAPEWORKSPACE name="Transport" directory="\\website\data\shapes "/>
  </WORKSPACES>
   <LAYER type="featureclass" name="Roads" visible="true" id="8">
     <DATASET name="roads" type="line" workspace="Transport"/>
```

	`        <SIMPLERENDERER>`
	`        <SIMPLELINESYMBOL width="2" color="156,50,235" type="solid" captype="round"`
	`        </SIMPLERENDERER>`
	`    </LAYER>`
	`   </MAP>`
	`  </CONFIG>`
	`</ARCXML>`

7.	Response returns through the reserve order of the initial request with the requested information.



```
<ARCXML version="1.1">
 <REQUEST>
  <GET_IMAGE>
   <PROPERTIES>
    <ENVELOPE minx="-130" miny="25" maxx="-110" maxy="40" />
    <IMAGESIZE height="457" width="565" />
   </PROPERTIES>
  </GET_IMAGE>
 </REQUEST>
```

```
<ARCXML version="1.1">
<RESPONSE>
<IMAGE>
<ENVELOPE minx="-130.000" miny="24.688" maxx="-110.000" maxy="40.311" />
<OUTPUT url="http://aardvark/mapsWorld_AARDVARK19633023.gif" />
</IMAGE>
</RESPONSE>
```

Figure: The ArcXML request and response cycle (http://webhelp.esri.com/arcims/9.2/general/)

5.6. ESRI ArcIMS Installation

Sometimes it is not an easy task to install a web-mapping server. It requires installation of several components, as noted in the previous sections. Therefore, ESRI provides a very detailed guide on how to install not only ESRI components but also third party components (e.g. web server with servlet engine). In a typical installation, the Web site resides on one computer. However, users still have to define the best solution for site configuration and perhaps need to install ArcIMS on a distributed system.

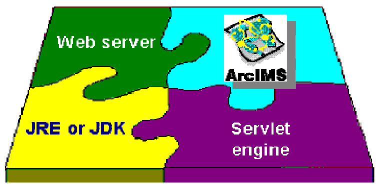
ESRI ArcIMS installation guide assumes one has a Web server and servlet engine installed and knows how to administer the Web server, stop and start services/daemons, and create virtual directories. It also assumes knowledge of the Internet and related terminology.

The five installation steps are required to get ArcIMS up and running. These steps are:

1. Verification that a site meets ArcIMS system requirements

ArcIMS can be run on a few operation system platforms, such as Windows, Sun Solaris, etc.

The following system software is required to run ArcIMS**:**



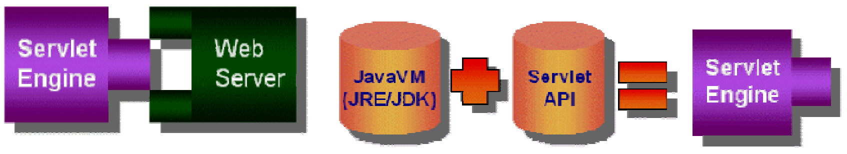


Figure: User has to choose and install Java 2 Platform Standard Edition Java Runtime Environment (J2SE JRE), Web server and Servlet engine (see more at the ESRI ArcIMS installation guide)

However, ESRI provides information on setting up a Web server and Servlet Engine on their website (e.g. at the http://support.esri.com/index.cfm?fa=knowledgebase.techarticles.articleShow&d=31218).

ArcIMS Spatial server and ArcIMS ArcMap Server requires an ESRI authorization file.

2. Planning of an ArcIMS site configuration

This part is most the challenging because an ArcIMS site resides on one or more computers, with the ArcIMS host (ArcIMS User Interface Components - Author, Designer, Administrator, and Service Administrator, Service Administrator, Connectors, and ArcIMS Application Server), ArcIMS Spatial Servers, and databases.

There are many possible ArcIMS site configurations. Some sites run all ArcIMS components on a single computer while other sites are comprised of multiple computers. Configurations will vary depending on the available computer resources and the required efficiency of mapping services provided. Questions have to be answered: How many services will be created at one time? How do heavy processing functions occur? How many simultaneous users will be able to visit the site each day?

ESRI provides planning recommendations for ArcIMS site configuration, such as the following:

- The Connectors must be installed on the Web server machine
- The ArcIMS Servlet Connector should only be installed on the Web server machine
- The Author, Designer, and Administrator can be installed on any machine
- Spatial Server can be installed on one machine or many machines
- Application Server can be installed on one machine or many machines

The following common configuration scenarios of distributed various installation components across various computer platforms are possible. The ArcIMS features in the installation configuration diagram, illustrated in the same color, should be installed on the same machine (see the ESRI ArcIMS installation guide).

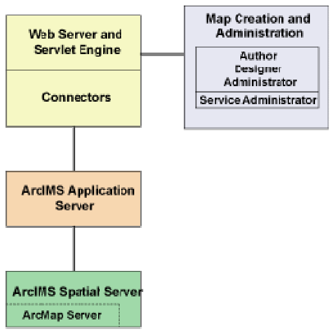


Figure: ArcIMS site on single machine

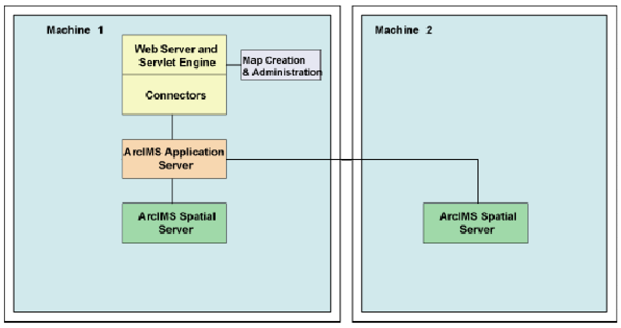

Machine 1
Web Server and Servlet Engine
Map Creation & Administration
Connectors
ArcIMS Application Server
ArcIMS Spatial Server
Machine 2
ArcIMS Spatial Server

Figure: Two ArcIMS Spatial Servers on multiple machines

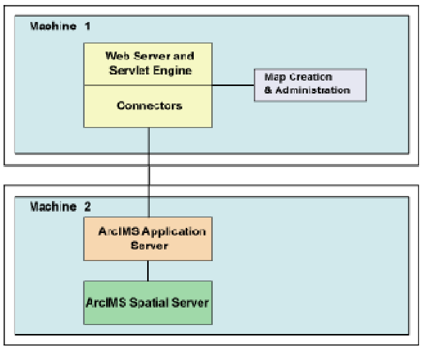

Machine 1
Web Server and Servlet Engine
Map Creation & Administration
Connectors
Machine 2
ArcIMS Application Server
ArcIMS Spatial Server

Figure: Dedicated Web server machine

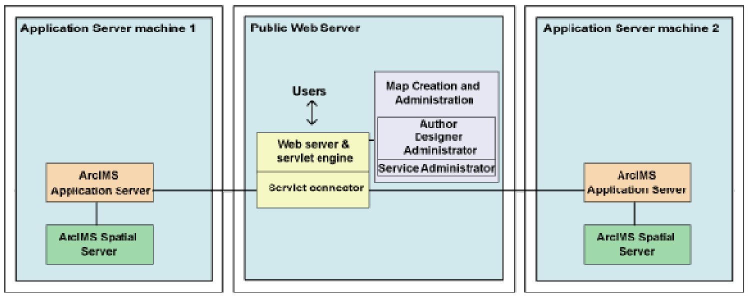

Application Server machine 1
ArcIMS Application server
ArcIMS Spatial Server
Public Web Server
Users
Web server & servlet engine
Servlet connector
Map Creation and Administration
Author
Designer
Administrator
Service Administrator
Application Server machine 2
ArcIMS Application Server
ArcIMS Spatial Server

Figure: Multiple ArcIMS Application Servers

3. ArcIMS Installation

The ArcIMS 9.2 install includes the installation and post installation setup.

- The installation process will install selected features. This process is well documented in the ESRI ArcIMS installation guide.
- The post installation is the setup process that completes ArcIMS installation. In the post installation setup, depending on what features were installed, it can authorize ArcIMS for use, configure ArcIMS, configure the ArcSDE Services Monitor, configure Web server-servlet engine, and configure system JRE.

On the post installation step, each Spatial Server installation has to be prompted to the machine where the Application Server is installed. One Spatial Server cannot support multiple Application Servers, but one Application Server can support multiple Spatial Servers.

Each Application Server installation needs to know the name of the Web server. On the post installation step, the name of the Web Server has to be prompted for the Application Server and the site preferences and diagnostics property files can be edited.

4. Web server configuration

At this step, Web server and Servlet Engine must be verified so they are communicating with each other properly. They can be configured automatically or manually.

5. ArcIMS configuration

At this step, ArcIMS environmental parameters can be set up.

5.7. ESRI ArcIMS and ArcGIS Server Customization Options

One of the advantages of ArcIMS and ArcServer is that there are few customization options for different software platforms, including non-programming options. Customization can be done on:

- Server sites where the following components could be customized:
    - Web sites: website designed by Designer or Web Manager can be customized by making HTML, CCS, JavaScript, or VBScript modifications. This level of customization may require no programming.

      ArcIMS Designer uses the viewer templates in order to define how end-users will see web sites and the functionality that is available to them for exploring maps. ArcIMS provides two types of viewers: the ***HTML viewer*** is used for ESRI image services and requires only a web browser support and the ***Java viewer*** allows one to stream feature geometry and corresponds to ESRI feature services and requires a thick client. The HTML Viewer uses HTML, JavaScript, and some Dynamic HTML (DHTML). It provides a framework for the map, toolbar, legend, overview map, and other graphic elements for the site. Customization can be done on the server side by editing HTML and JavaScript codes (see more at http://webhelp.esri.com/arcims/9.2/general/mergedProjects/Books/Customizing_the_HTML_Viewer.pdf and http://webhelp.esri.com/arcims/9.2/general/mergedProjects/Books/Customizing_the_Java_Viewer.pdf).

      For a web site created with the Web Manager with .asp technology, layers and tasks can be added or removed, and mapping and page elements can be changed by using the Web Manager (http://edndoc.esri.com/arcobjects/9.2/NET_Server_Doc/manager/applications/intro_web_apps_mgr.htm). In addition, a web mapping application built with Web Manager can be open in an Integrated Development Environment (e.g. MS Visual Studio .NET) and modified. Thus, the Web Manager application is built on top of the ArcIMS Web ADF (Application Development Framework) and all the Web controls and API's associated with the ADF are available (e.g. ASP.NET controls). Developer can change properties on the controls and for more advanced customization, Developer can write new code (http://edndoc.esri.com/arcobjects/9.2/NET_Server_Doc/developer/ADF/extending_manager_web_apps.htm).

      ArcGIS Server uses ArcGIS Server Manager to create web applications. A web-mapping site, created by Manager, can be customized by modifying properties of web controls and making basic HTML modifications. Programming against the J2EE and .NET application programming interfaces (API's) (see http://webhelp.esri.com/arcgisserver/9.2/) can also be used. ArcGIS Server comes with an ADF that contains tools for building GIS web applications. Developer can drag and drop the web ADF's controls onto a web form to quickly create an application that does mapping, editing, geocoding, geoprocessing, and more. The web ADF's Developer libraries can be used. The web ADF also contains the ability to integrate multiple service types from ArcIMS and ArcWeb Services into one map, and provides entry points for programming with these service types using .NET or Java. During the installation, the web ADF is automatically integrated into an installed development environment, such as MS Visual Studio, Eclipse, or Creator.

  - Connecters: ArcIMS can be customized and extended using industry-standard Web development environments such as JSP, ASP, .NET, and others (see e.g. http://webhelp.esri.com/arcims/9.2/general/mergedProjects/Books/Using_ActiveX_Connector.pdf, http://webhelp.esri.com/arcims/9.2/general/mergedProjects/Books/Using_ColdFusion_Connector.pdf)

  - ArcXML configuration file: ArcIMS clients and servers communicate using ArcXML, which is an ESRI extension to standard XML. Changes of ArcXML is a way to customize ArcIMS applications.

  - External Application Server (e.g. ColdFusion)

  - ArcGIS Server provides an ArcObjects software-based server development environment for deploying GIS server-based ArcGIS applications and services. ArcGIS Server provides a rich application developer framework for both the J2EE and .NET environments.

- Client side where the following components could be customized:
  - Custom application (e.g. ArcMap)
  - Web browser' applet, plug-in, etc. (e.g. the ESRI Java Viewer applet for a client can be customized by using Java programming).

- DBMS level.

5.8. *ESRI ArcIMS Ancillary Features*

5.8.1 ESRI optional extensions

The Web publishing capabilities in a GIS web site can be enhanced through a series of optional ArcIMS product extensions. ESRI also offers the following extensions for ArcIMS. ArcIMS Data Delivery (http://www.esri.com/software/arcgis/arcims-datadelivery/index.html) extension can be used for spatial data publishing in the different spatial formats used within the industry. ArcIMS Route Server (http://www.esri.com/software/arcgis/arcims-routeserver/index.html) extension adds routing functionality to ArcIMS Web site.

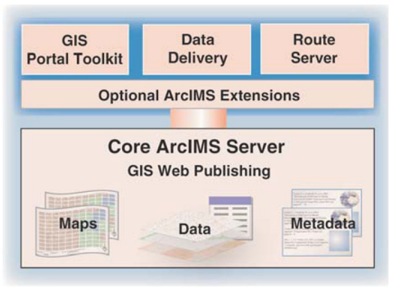


Figure: ESRI optional extensions

5.8.2 Security Issues

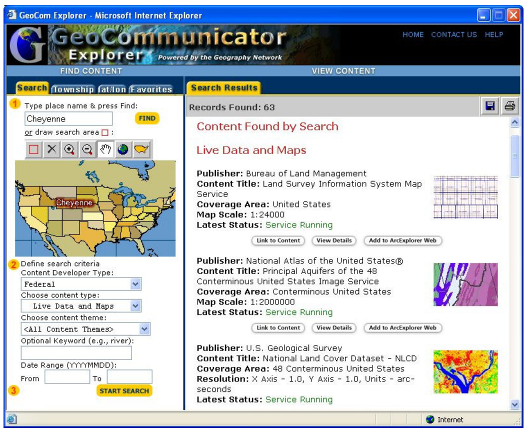
ArcIMS has standard and additional mechanisms to protect data and sites such as:

- User authentication for map services, allowing one to define which users or client machines (by IP address) have access to GIS data. Additional restrictions can be set up such as an expiration date and time for a user, use of specific ArcXML elements, etc.
- Secure hypertext transfer protocol and secure sockets layer protocol for managing the security of a Web site
- Several built-in ways to set limits to the number of requested features when making GET_FEATURES requests
- Restricting geometry being returned from ArcIMS image services to ArcMap client for query requests
- Restricting requests to a Spatial Server through the parameters in property file of the Servlet Connector

In addition, standard security mechanisms of operation system, web server, third party application server, and DBMS can be used to put further restrictions on user authentication, data encryptions, folder access control, etc. (See ArcGIS Enterprise Security: Delivering Secure Solutions, An ESRI White Paper, 2005, http://www.esri.com/library/whitepapers/pdfs/arcgis-security.pdf for more details). Addition security mechanisms can be developed by customization of application server connectors.

5.8.3 ArcIMS Metadata Service

ArcIMS provides the mechanism for hosting a Metadata Service, allowing clients to publish to the service as well as search its contents. With the 9.2 release, ArcIMS offers Metadata Explorer web application that is included with the ArcIMS Web ADF for the Java Platform and that gives one the ability to browse the contents of a Metadata Service from a web browser. Metadata Explorer looks similar to the interface for accessing the Geography Network.


GeoCom Explorer - Microsoft Internet Explorer
GeoCommunicator Explorer
Powered by the Geography Network
HOME CONTACT US HELP
FIND CONTENT
VIEW CONTENT
Search Township lat/lon Favorites
Search Results
FIND
START SEARCH
Link to Content View Details Add to ArcExplorer Web
Internet

1 Type place name & press Find: Cheyenne
or draw search area :

2 Define search criteria
Content Developer Type: Federal
Choose content type: Live Data and Maps
Choose content theme: <All Content Themes>
Optional Keyword (e.g., river):
Date Range (YYYYMMDD):
From To
3

Records Found: 63

Content Found by Search

Live Data and Maps

Publisher: Bureau of Land Management
Content Title: Land Survey Information System Map Service
Coverage Area: United States
Map Scale: 1:24000
Latest Status: Service Running

Publisher: National Atlas of the United States®
Content Title: Principal Aquifers of the 48 Conterminous United States Image Service
Coverage Area: Conterminous United States
Map Scale: 1:2000000
Latest Status: Service Running

Publisher: U.S. Geological Survey
Content Title: National Land Cover Dataset - NLCD
Coverage Area: 48 Conterminous United States
Resolution: X Axis - 1.0, Y Axis - 1.0, Units - arc-seconds
Latest Status: Service Running

Figure: The Geography Network interface for search spatial data and services via metadata

5.8.4 ArcIMS Tuning Tips

ESRI also provides several instructions on how to help keep the web site running efficiently. These concern such things as how to:

- Set up scale-dependent rendering within .axl file
- Use simple symbology and labels
- Generalize spatial data
- Use specific image types and size
- Work with map projections;
- Create spatial indexes

These, and other instructions, can be seen at the following site: http://webhelp.esri.com/arcims/9.2/general/ and/or by watching the free ESRI online virtual campus course Introduction to ArcIMS at: http://campus.esri.com/acb2000/showdetl.cfm?&DID=6&Product_ID=833&CATID=84&CFID=4506767&CFTOKEN=37230417.

5.9. Conclusion

ArcIMS provides functionality to deliver dynamic maps and data via the web; easy ways to create applications that use geographic content; to develop custom applications using industry-standard web development environments; and, to share data and implement GIS portals. ArcIMS provides a highly scalable framework for web map publishing within the corporate intranets and worldwide Internet. A wide range of clients, including custom web applications, the ArcGIS Desktop, and mobile and wireless devices, can use ArcIMS services. ArcIMS is also open for customization. ArcIMS has the Application Server connecters in order to comply with OGC WMS and WFS specifications.

ArcGIS Server hosts more diverse GIS and mapping services compared to ArcIMS. These services include: ArcGIS Mapping, OGC WMS, Mobile Data Access, KLM, Geodata Access extraction, Access feature, Geoprocessing, Network Analysis, Globe, and Geocoding (see http://webhelp.esri.com/arcgisserver/9.2/dotNet/). These services can be housed within a web browser or by a custom application. ArcGIS applications, such as ArcMap and ArcGlobe, can also act as clients to GIS services.

However, ArcIMS and ArcGIS servers are not free software and if a user or organization does not have budget for ArcIMS purchase, they are free to use open source or code software. In such scenarios, the following software could be used:

- GeoServer, http://docs.codehaus.org/display/GEOS/Home (open source)
- MapServer, http://mapserver.gis.umn.edu/ (open source)
- MapGuide from Autodesk, http://usa.autodesk.com/adsk/servlet/index?id=7171990&siteID=123112 (open code)
- OpenLayers, API: Javascript, http://openlayers.org/ (open source)
- etc.

ESRI documentation describes the ArcIMS architecture and their components in detail and the principles of ESRI web mapping software operation can be used to understand functionalities of other web mapping software, such as open source, which usually has a less detailed help system.

Module self-study questions:

1. What are the main responsibilities of ESRI's Application Server?
2. What ArcIMS component corresponds to map engine?
3. What is the ESRI Virtual Server and why it was it introduced?
4. What multiplicity relationships are found between Spatial Servers and Application Servers?
5. What is the difference between the ESRI Image and Feature Service? What will the output image format of the Image Service be?
6. Examine the .axl `<CONFIG>` file from the section "ArcIMS Communication Flow" and try to answer the following questions:
   a. What version of ArcXML is being used?
   b. Where is this project's data stored?
   c. What map units are being used?
7. Why does ArcIMS have a scalable architecture?
8. What is the principle difference between ESRI ArcIMS and ArcGIS Server?

Recommended Readings:

[1] Section 4, 5 and 6, System Design Strategies An ESRI Technical Reference Document, 2007, http://www.esri.com/library/whitepapers/pdfs/sysdesig.pdf

[2] Introduction to ArcIMS at the http://campus.esri.com/acb2000/showdetl.cfm ?&DID=6&Product_ID=833&CATID=84&CFID=4506767&CFTOKEN=37230417, free online ESRI virtual campus course

References

[1] ArcIMS Online Main Help http://webhelp.esri.com/arcims/9.2/general/ or http://support.esri.com/index.cfm?fa=knowledgebase.webHelp.arcIMSGateway

[2] The ArcGIS Server Web Help, http://support.esri.com/index.cfm?fa=knowledgebase.webHelp.agServer

[3] Getting Started with ArcIMS, http://downloads.esri.com/support/documentation/ims_/1010Getting_Started_with_ArcIMS.pdf

[4] ArcXML Programmer's Reference Guide for ArcIMS 9.2, http://webhelp.esri.com/arcims/9.2/general/

[5] System Design Strategies An ESRI ® Technical Reference Document, 2007, http://www.esri.com/library/whitepapers/pdfs/sysdesig.pdf

[6] ArcGIS Enterprise Security: Delivering Secure Solutions, An ESRI White Paper, 2005, http://www.esri.com/library/whitepapers/pdfs/arcgis-security.pdf

Terms used

- ArcIMS
- ArcGIS Server
- Web Server
- Application Server Connector
- Spatial Server
- Services
- Virtual Server
- Instance
- Thread
- Application Server
- Monitor
- Tasker
- Author
- Administrator
- Service Administrator
- Web Manager
- Designer
- Client Viewer
- AXL configuration file
- Java 2 Platform Standard Edition Java Runtime Environment (J2SE JRE)
- Servlet engine
- Application Developer Framework (ADF)
- Java Viewer
- HTML Viewer