# Mt Kenya University

P.O. Box 342-01000 Thika

Email: info@mku.ac.ke

Web: www.mku.ac.ke

# DEPARTMENT OF INFORMATION TECHNOLOGY

# COURSE CODE: BIT 3207

COURSE TITLE: ADVANCED WEB DESIGN AND DEVELOPMENT FOR BUSINESS WORLD

Instructional manual for BBIT – Distance Learning

# TABLE OF CONTENT

# COURSE OUTLINE

**BIT 3207:** ADVANCED WEB DESIGN AND DEVELOPMENT FOR BUSINESS WORLD

**Contact Hours:** 42

**Pre-requisite: BIT 2105 -** INTRODUCTION TO WEB DESIGN AND DEVELOPMENT

**Lecturer:** Mr. Kibaara

**Purpose:** To acquire advanced web technologies for advanced web systems

**Objectives:** By the end of the course unit the learner should:

- Be introduced to advanced technologies used to build the world wide web (WWW) commonly applied in business
- Develop an advanced business website with transactional capacity

**Course Assessments:** Continuous Assessment Tests 30%

End of semester examination 70%

**Teaching/Learning Methodology:** Lectures, Tutorials, Computer laboratory

**Instructional Materials/Equipment:** Audio visual aids, Computer laboratory, Internet Access

**Required Textbooks:**

Morrison M.c.Morrison J. (2000), Databas driven website, Thomas learning

**Textbooks for Further reading:**

Wilide E wilde's W.W.W.(1999), Technical Foundation of the W.W.W., Springer

**ADVANCED WEB DESIGN AND DEVELOPMENT FOR BUSINESS WORLD - TOPICS – Details**

**Week 1: Introduction**

- Networking fundamentals
- Client-side components
- Server-side component
- Database components

**Week 2: Architecture and design**

- I.N-tier designs
- user interface database servers, directory servers, transition servers,
- interfacing to legacy systems

- security issues
- Object-oriented modeling for web applications using UML and WAC

**Week 3-4: Active Server Pages (ASP)**

- ASP Introduction
- ASP Install
- ASP Syntax

**Week 5: CAT 1**

**Week 6-9: Active Server Pages (ASP)**

- ASP Variables
- ASP loops
- ASP Conditional statements
- ASP Procedures
- ASP Forms
- ASP Cookies
- ASP Session
- ASP Application
- ASP #include
- ASP Global.asa

**Week 10-12: Object Oriented Development using ASP**

- Introduction
- Classes, objects, inheritance

**Module Compiler: David Kibaara**

# CHAPTER ONE: INTRODUCTION TO ADVANCED WEB DESIGN AND DEVELOPMENT CONCEPTS

***Learning Objectives:***

*By the end of this chapter the learner shall be able to;*

i. Explain the networking fundamentals and TCP/IP protocol suite

ii. Explain the components in the client side and server side

## 1.1 Networking fundamentals

A computer network is a collection of hardware components and computers interconnected by communications channels that allow sharing of resources and information. The rules and data formats for exchanging information in a computer network are defined by communications protocols.

### *1.1.1 Properties of a computer network*

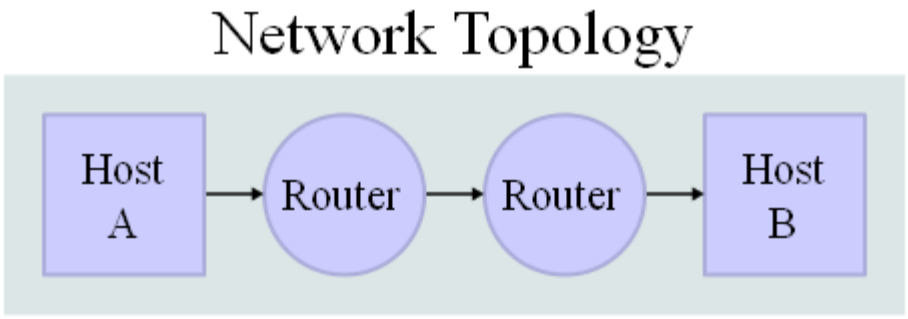
- Facilitate communications - Using a network, people can communicate efficiently and easily via email, instant messaging, chat rooms, telephone, video telephone calls, and video conferencing.
- Permit sharing of files, data, and other types of information - In a network environment, authorized users may access data and information stored on other computers on the network. The capability of providing access to data and information on shared storage devices is an important feature of many networks.
- Share network and computing resources - In a networked environment, each computer on a network may access and use resources provided by devices on the network, such as printing a document on a shared network printer. Distributed computing uses computing resources across a network to accomplish tasks.
- May be insecure - A computer network may be used by computer hackers to deploy computer viruses or computer worms on devices connected to the network, or to prevent these devices from normally accessing the network (denial of service).

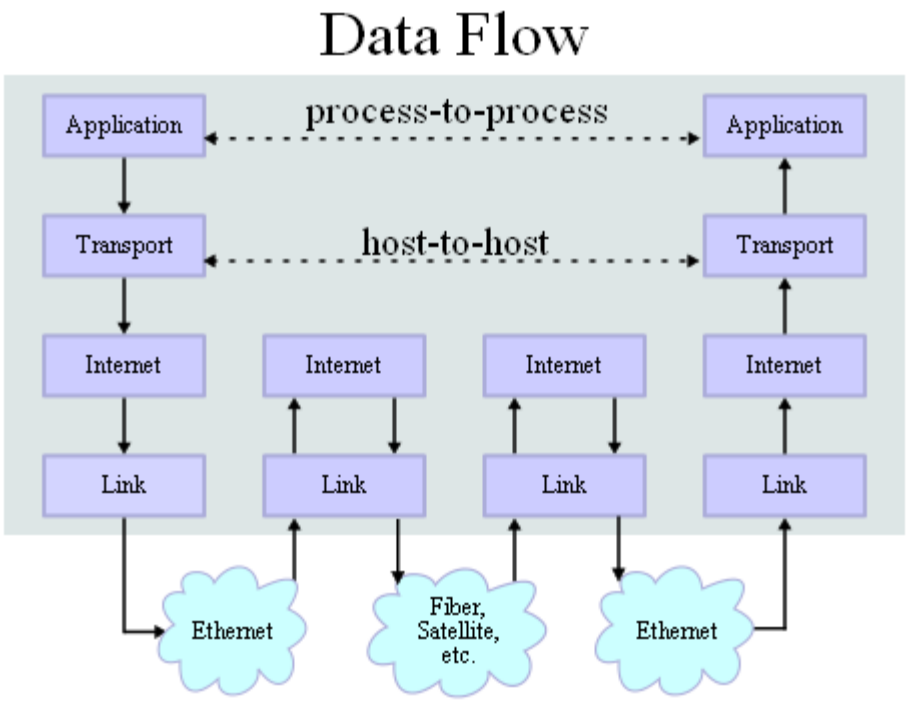
### *1.1.2 TCP/IP model*

The TCP/IP model is a description framework for computer network protocols created in the 1970s by DARPA, an agency of the United States Department of Defense. It evolved from ARPANET, which were the world's first wide area network and a predecessor of the Internet.

The TCP/IP model describes a set of general design guidelines and implementations of specific networking protocols to enable computers to communicate over a network. TCP/IP provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination. Protocols exist for a variety of different types of communication services between computers. TCP/IP has the following layers:

- Application Layer (process-to-process): This is the scope within which applications create user data and communicate this data to other processes or applications on another or the same host. The communications partners are often called peers. This is where the "higher level" protocols such as SMTP, FTP, SSH, HTTP, etc. operate.
- Transport Layer (host-to-host): The Transport Layer constitutes the networking regime between two network hosts, either on the local network or on remote networks separated by routers. The Transport Layer provides a uniform networking interface that hides the actual topology (layout) of the underlying network connections. This is where flow-control, error-correction, and connection protocols exist, such as TCP. This layer deals with opening and maintaining connections between Internet hosts.
- Internet Layer (internetworking): The Internet Layer has the task of exchanging datagram's across network boundaries. It is therefore also referred to as the layer that establishes internetworking; indeed, it defines and establishes the Internet. This layer defines the addressing and routing structures used for the TCP/IP protocol suite. The primary protocol in this scope is the Internet Protocol, which defines IP addresses. Its function in routing is to transport datagram's to the next IP router that has the connectivity to a network closer to the final data destination.
- Link Layer: This layer defines the networking methods within the scope of the local network link on which hosts communicate without intervening routers. This layer describes the protocols used to describe the local network topology and the interfaces needed to affect transmission of Internet Layer datagram's to next-neighbor hosts.




### *1.1.3 Internet socket*

Internet socket or network socket is an endpoint of a bidirectional inter-process communication flow across an Internet Protocol-based computer network, such as the Internet. The term Internet sockets is also used as a name for an application programming interface (API) for the TCP/IP protocol stack, usually provided by the operating system. Internet sockets constitute a mechanism for delivering incoming data packets to the appropriate application process or thread, based on a combination of local and remote IP addresses and port numbers. Each socket is mapped by the operating system to a communicating application process or thread.

A socket address is the combination of an IP address (the location of the computer) and a port (which is mapped to the application program process) into a single identity, much like one end of a telephone connection is the combination of a phone number and a particular extension.

### *1.1.4 Hypertext Transfer Protocol (HTTP)*

The Hypertext Transfer Protocol (HTTP) is a networking protocol for distributed, collaborative, hypermedia information systems. HTTP is the foundation of data communication for the World Wide Web. The standards development of HTTP has been coordinated by the Internet Engineering Task Force (IETF) and the World Wide Web Consortium (W3C).

HTTP functions as a request-response protocol in the client-server computing model. In HTTP, a web browser, for example, acts as a client, while an application running on a computer hosting a web site functions as a server. The client submits an HTTP request message to the server. The server, which stores content, or provides resources, such as HTML files, or performs other functions on behalf of the client, returns a response message to the client. A response contains completion status information about the request and may contain any content requested by the client in its message body.

A web browser (or client) is often referred to as a user agent (UA). Other user agents can include the indexing software used by search providers, known as web crawlers, or variations of the web browser such as voice browsers, which present an interactive voice user interface.

The HTTP protocol is designed to permit intermediate network elements to improve or enable communications between clients and servers. High-traffic websites often benefit from web cache servers that deliver content on behalf of the original, so-called origin server, to improve response time. HTTP proxy servers at network boundaries facilitate communication when clients without a globally routable address are located in private networks by relaying the requests and responses between clients and servers.

HTTP is an Application Layer protocol designed within the framework of the Internet Protocol Suite. The protocol definitions presume a reliable Transport Layer protocol for host-to-host data transfer. The Transmission Control Protocol (TCP) is the dominant protocol in use for this purpose. However, HTTP has found application even with unreliable protocols, such as the User Datagram Protocol (UDP) in methods such as the Simple Service Discovery Protocol (SSDP).

HTTP Resources are identified and located on the network by Uniform Resource Identifiers (URIs)—or, more specifically, Uniform Resource Locators (URLs)—using the http or https URI schemes. URIs and the Hypertext Markup Language (HTML), form a system of inter-linked resources, called hypertext documents, on the Internet, that led to the establishment of the World Wide Web in 1990 by English physicist Tim Berners-Lee.

## 1.2 Client-side components

Client-side refers to operations that are performed by the client in a client–server relationship in a computer network.

Typically, a client is a computer application, such as a web browser, that runs on a user's local computer or workstation and connects to a server as necessary. Operations may be performed client-side because they require access to 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 in a timely manner for all of the clients it serves. Additionally, if operations can be performed by the client, without sending data over the network, they may take less time, use less bandwidth, and incur a lesser security risk.

When the server serves data in a commonly used manner, for example according to the HTTP or FTP protocols, users may have their choice of a number of client programs. In the case of more specialized applications, programmers may write their own server, client, and communications protocol that can only be used with one another.

Programs that run on a user's local computer without ever sending or receiving data over a network are not considered clients, and so the operations of such programs would not be considered client-side operations.

### *1.2.1 Hypertext Markup Language (HTML)*

Hypertext Markup Language (HTML) is the predominant markup language for web pages. HTML elements are the basic building-blocks of webpage's. HTML is written in the form of HTML elements consisting of tags, enclosed in angle brackets (like <html>), within the web page content. HTML tags normally come in pairs like <h1> and </h1>. The first tag in a pair is the start tag, the second tag is the end tag (they are also called opening tags and closing tags). In between these tags web designers can add text, tables, images, etc.

The purpose of a web browser is to read HTML documents and compose them into visible or audible web pages. The browser does not display the HTML tags, but uses the tags to interpret the content of the page. HTML elements form the building blocks of all websites. HTML allows images and objects to be embedded and can be used to create interactive forms. It provides a means to create structured documents by denoting structural semantics for text such as headings, paragraphs, lists, links, quotes and other items. It can embed scripts in languages such as JavaScript which affect the behavior of HTML webpage's.

Web browsers can also refer to Cascading Style Sheets (CSS) to define the appearance and layout of text and other material. The W3C, maintainer of both the HTML and the CSS standards, encourages the use of CSS over explicitly presentational HTML markup.

### *1.2.2 Extensible Markup Language (XML)*

Extensible Markup Language (XML) is a set of rules for encoding documents in machine-readable form. It is defined in the XML 1.0 Specification produced by the W3C, and several other related specifications,

all gratis open standards. The design goals of XML emphasize simplicity, generality, and usability over the Internet. It is a textual data format with strong support via Unicode for the languages of the world. Although the design of XML focuses on documents, it is widely used for the representation of arbitrary data structures, for example in web services.



### *1.2.3 Web browser*

A web browser is a software application for retrieving, presenting, and traversing information resources on the World Wide Web. An information resource is identified by a Uniform Resource Identifier (URI) and may be a web page, image, video, or other piece of content. Hyperlinks present in resources enable users easily to navigate their browsers to related resources. A web browser can also be defined as an application software or program designed to enable users to access, retrieve and view documents and other resources on the Internet. Although browsers are primarily intended to access the World Wide Web, they can also be used to access information provided by web servers in private networks or files in file systems. The major web browsers are Internet Explorer, Firefox, Google Chrome, Safari, and Opera.

### *1.2.4 Applet*

An applet is any small application that performs one specific task that runs within the scope of a larger program, often as a plug-in. An applet typically also refers to Java applets, i.e., programs written in the Java programming language that are included in a web page. Applets are used to provide interactive features to web applications that cannot be provided by HTML alone. They can capture mouse input and also have controls like buttons or check boxes. In response to the user action an applet can change the provided graphic content. This makes applets well suitable for demonstration, visualization, and teaching. Applets are also used to create online game collections that allow players to compete against live opponents in real-time.

HTML pages may embed parameters that are passed to the applet. Hence the same applet may appear differently depending on the parameters that were passed.

Examples of Web-based Applets include:

- QuickTime movies
- Flash movies
- Windows Media Player applets, used to display embedded video files in Internet Explorer (and other browsers that support the plugin)
- 3D modeling display applets, used to rotate and zoom a model
- Browser games can be applet-based, though some may develop into fully functional applications that require installation.

## 1.3 Server-side component

Server-side refers to operations that are performed by the server in a client–server relationship in computer networking. Typically, a server is a software program, such as a web server, that runs on a remote server, reachable from a user's local computer or workstation. Operations may be performed server-side because they require access to information or functionality that is not available on the client, or require typical behavior that is unreliable when it is done client-side.

Server-side operations also include processing and storage of data from a client to a server, which can be viewed by a group of clients. Advantage: This lightens the work of your client.

### *1.3.1 Web server*

Web server can refer to either the hardware (the computer) or the software (the computer application) that helps to deliver content that can be accessed through the Internet. The most common use of web servers is to host web sites but there are other uses like data storage or for running enterprise applications.

### *1.3.2 Servlet*

A servlet is a Java programming language class used to extend the capabilities of servers that host applications accessed via a request-response programming model. Although servlets can respond to any type of request, they are commonly used to extend the applications hosted by Web servers. Thus, it can be thought of as a Java Applet that runs on a server instead of a browser.

### *1.3.3 JavaServer Pages (JSP)*

JavaServer Pages (JSP) is a Java technology that helps software developers serve dynamically generated web pages based on HTML, XML, or other document types. Released in 1999 as Sun's answer to ASP and PHP, JSP was designed to address the perception that the Java programming environment didn't provide developers with enough support for the Web. To deploy and run, a compatible web server with servlet container is required. The Java Servlet and the JavaServer Pages (JSP) specifications from Sun Microsystems and the JCP (Java Community Process) must both be met by the container.

## 1.4 Database components

A database is an organized collection of data for one or more purposes, usually in digital form. The data is typically organized to model relevant aspects of reality (for example, the availability of rooms in hotels), in a way that supports processes requiring this information (for example, finding a hotel with vacancies). The term "database" refers both to the way its users view it, and to the logical and physical materialization of its data, content, in files, computer memory, and computer data storage. This definition is very general, and is independent of the technology used. However, not every collection of data is a database; the term database implies that the data is managed to some level of quality (measured in terms

of accuracy, availability, usability, and resilience) and this in turn often implies the use of a general-purpose **Database management system (DBMS)**. A general-purpose DBMS is typically a complex software system that meets many usage requirements, and the databases that it maintains are often large and complex. The utilization of databases is widely spread to a degree that virtually any technology and product nowadays relies on databases and DBMSs for its development and commercialization, or even may have such embedded in it. Also organizations and companies, from small to very large, heavily depend on databases for their operations.



### *1.4.1 Database Management System (DBMS)*

A database management system (DBMS) is a software package with computer programs that control the creation, maintenance, and the use of a database. It allows organizations to conveniently develop databases for various applications by database administrators (DBAs) and other specialists. A DBMS allows different user application programs to concurrently access the same database. DBMSs may use a variety of database models, such as the relational model or object model, to conveniently describe and support applications. It typically supports query languages, which are in fact high-level programming languages, dedicated database languages that considerably simplify writing database application programs. Database languages also simplify the database organization as well as retrieving and presenting information from it. A DBMS provides facilities for controlling data access, enforcing data integrity, managing concurrency control, recovering the database after failures and restoring it from backup files, as well as maintaining database security.

### *1.4.2 Structured Query Language (SQL)*

SQL is a programming language designed for managing data in relational database management systems (RDBMS). Originally based upon relational algebra and tuple relational calculus, its scope includes data insert, query, update and delete, schema creation and modification, and data access control.

### *1.4.3 Database server*

A database server is a computer program that provides database services to other computer programs or computers, as defined by the client–server model. The term may also refer to a computer dedicated to running such a program. Database management systems frequently provide database server functionality, and some DBMSs (e.g., MySQL) rely exclusively on the client–server model for database access. Such a server is accessed either through a "front end" running on the user's computer which displays requested data or the "back end" which runs on the server and handles tasks such as data analysis and storage.

In a master-slave model, database master servers are central and primary locations of data while database slave servers are synchronized backups of the master acting as proxies. Some examples of Database servers are Oracle, DB2, Informix, Ingres, SQL Server.

### *1.4.4 Java Database Connectivity*

Java DataBase Connectivity (JDBC) is an API for the Java programming language that defines how a client may access a database. It provides methods for querying and updating data in a database. JDBC is oriented towards relational databases. A JDBC-to-ODBC bridge enables connections to any ODBC-accessible data source in the JVM host environment.



## Chapter Review Questions

1. Discuss the layers in the TCP/IP Model
2. Why is TCP/IP model important in computer networking?
3. Discuss the following terms
    - Web server
    - Applet
    - XML
    - Servlet
    - DBMS
    - SQL

## Suggested Further Reading

1. Mark S et al (1996), Special Edition using internet HTML Que Publishing Co ltd
2. Alex H et all(2000),, Professional active server Wrox publishers programmer to programmer series
3. http://www.w3schools.com

# CHAPTER TWO: ARCHITECTURE AND DESIGN

***Learning Objectives:***

*By the end of this chapter the learner shall be able to;*

i. Explain the multitier internet architecture

ii. Understand legacy systems and how to integrate them with current systems

iii. Understand object oriented modeling for the internet

## 2.1 Multitier architecture

In software engineering, multi-tier architecture (often referred to as n-tier architecture) is a client–server architecture in which the presentation, the application processing, and the data management are logically separate processes. For example, an application that uses middleware to service data requests between a user and a database employs multi-tier architecture. The most widespread use of multi-tier architecture is the three-tier architecture.

N-tier application architecture provides a model for developers to create a flexible and reusable application. By breaking up an application into tiers, developers only have to modify or add a specific layer, rather than have to rewrite the entire application over. There should be a presentation tier, a business or data access tier, and a data tier.

The concepts of layer and tier are often used interchangeably. However, one fairly common point of view is that there is indeed a difference, and that a layer is a logical structuring mechanism for the elements that make up the software solution, while a tier is a physical structuring mechanism for the system infrastructure.

### 2.1.1 Three-tier architecture

Three-tier is a client–server architecture in which the user interface, functional process logic ("business rules"), computer data storage and data access are developed and maintained as independent modules, most often on separate platforms. The three-tier model is software architecture and a software design pattern.

Apart from the usual advantages of modular software with well-defined interfaces, the three-tier architecture is intended to allow any of the three tiers to be upgraded or replaced independently as requirements or technology change. For example, a change of operating system in the presentation tier would only affect the user interface code.



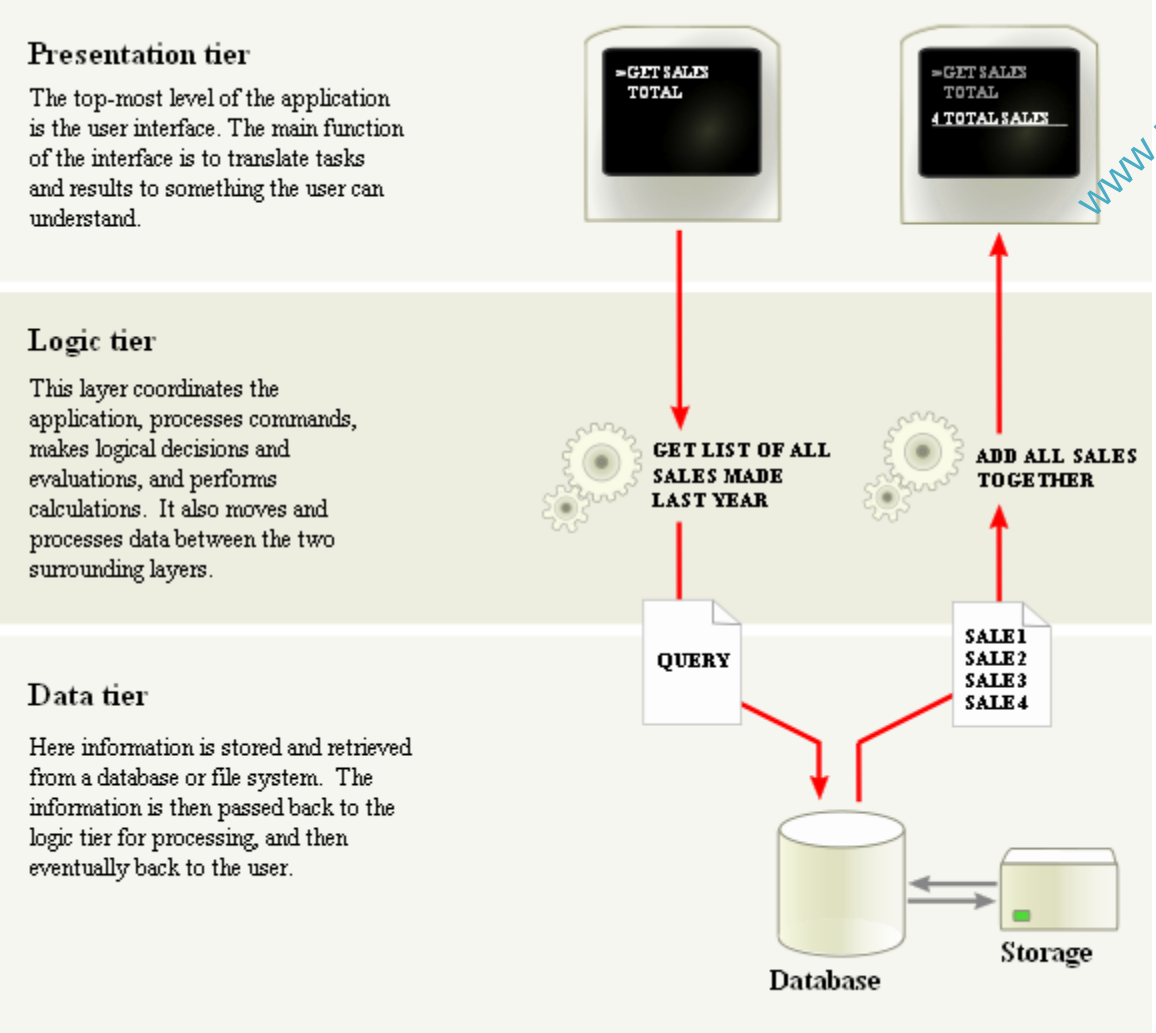
Typically, the user interface runs on a desktop PC or workstation and uses a standard graphical user interface, functional process logic may consist of one or more separate modules running on a workstation or application server, and an RDBMS on a database server or mainframe contains the computer data storage logic. The middle tier may be multi-tiered itself (in which case the overall architecture is called an "n-tier architecture").

Three-tier architecture has the following three tiers:

- Presentation tier - This is the topmost level of the application. The presentation tier displays information related to such services as browsing merchandise, purchasing, and shopping cart contents. It communicates with other tiers by outputting results to the browser/client tier and all other tiers in the network.

- Application tier (business logic, logic tier, data access tier, or middle tier) - The logic tier is pulled out from the presentation tier and, as its own layer, it controls an application’s functionality by performing detailed processing.

- Data tier - This tier consists of database servers. Here information is stored and retrieved. This tier keeps data neutral and independent from application servers or business logic. Giving data its own tier also improves scalability and performance.




### *2.1.2Web development usage*

In the web development field, three-tier is often used to refer to websites, commonly electronic commerce websites, which are built using three tiers:

1. A front-end web server serving static content, and potentially some cached dynamic content. In web based application, Front End is the content rendered by the browser. The content may be static or generated dynamically.

2. A middle dynamic content processing and generation level application server, for example Java EE, ASP.NET, PHP, ColdFusion platform.

3. A back-end database, comprising both data sets and the database management system or RDBMS software that manages and provides access to the data.

## 2.2 Interfacing to legacy systems

### *2.2.1 Legacy system*

A legacy system is an old method, technology, computer system, or application program that continues to be used, typically because it still functions for the users' needs, even though newer technology or more efficient methods of performing a task are now available. A legacy system may include procedures or terminology which are no longer relevant in the current context, and may hinder or confuse understanding of the methods or technologies used.

### *2.2.2 Organizations can have compelling reasons for keeping a legacy system, such as:*

- The system works satisfactorily, and the owner sees no reason for changing it.
- The costs of redesigning or replacing the system are prohibitive because it is large, monolithic, and/or complex.
- Retraining on a new system would be costly in lost time and money, compared to the anticipated appreciable benefits of replacing it (which may be zero).
- The system requires near-constant availability, so it cannot be taken out of service, and the cost of designing a new system with a similar availability level is high. Examples include systems to handle customers' accounts in banks, computer reservation systems, air traffic control, energy distribution (power grids), nuclear power plants, military defense installations, and systems such as the TOPS database.
- The way that the system works is not well understood. Such a situation can occur when the designers of the system have left the organization, and the system has either not been fully documented or documentation has been lost.
- The user expects that the system can easily be replaced when this becomes necessary.

### *2.2.3 Reasons why Legacy systems are considered problematic by software engineers:*

- Legacy systems often run on obsolete (and usually slow) hardware, and spare parts for such computers may become increasingly difficult to obtain.
- If legacy software runs on only antiquated hardware, the cost of maintaining the system may eventually outweigh the cost of replacing both the software and hardware unless some form of emulation or backward compatibility allows the software to run on new hardware.
- These systems can be hard to maintain, improve, and expand because there is a general lack of understanding of the system; the staff who were experts on it have retired or forgotten what they

knew about it, and staff who entered the field after it became "legacy" never learned about it in the first place. This can be worsened by lack or loss of documentation.

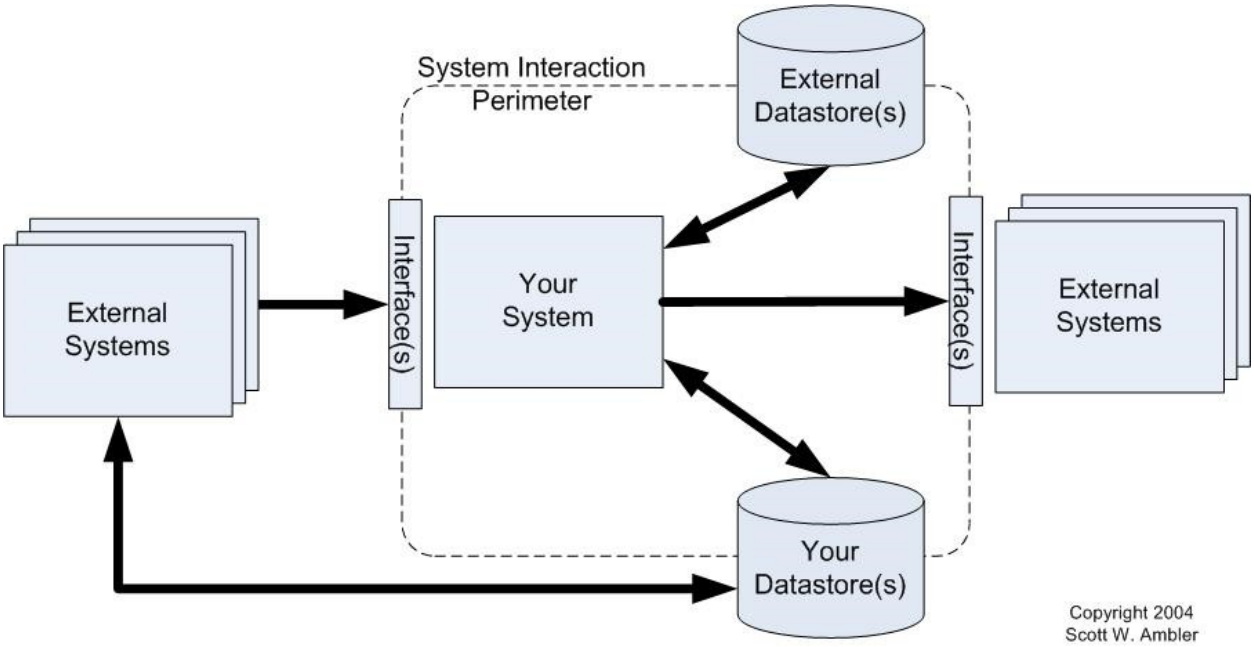
- Legacy systems may have vulnerabilities in older operating systems or applications due to lack of security patches being available or applied. There can also be production configurations that cause security problems. These issues can put the legacy system at risk of being compromised by attackers or knowledgeable insiders.

- Integration with newer systems may also be difficult because new software may use completely different technologies. The kind of bridge hardware and software that becomes available for different technologies that are popular at the same time are often not developed for differing technologies in different times, because of the lack of a large demand for it and the lack of associated reward of a large market economies of scale, though some of this "glue" does get developed by vendors and enthusiasts of particular legacy technologies.

### *2.3.4 Process of Interfacing new to legacy systems*

1. *Identifying Interactions*

To identify the interactions which your system has with others you need to identify the "interaction perimeter" of your system, depicted in Figure below.




As you can see there are five interaction possibilities:

a. Reads from external data source(s). Your system may read from external data sources such as files or databases. You will need to understand both the structure and semantics of the data.

b. Updates to external data source(s). There are several implications. First, other systems may depend on the updates that your system makes, coupling them to yours. Second, your system may increase the traffic to the legacy data source and thereby effect the performance of other systems.

c. External system interface(s). Your system may interact with other systems through provided systems interfaces such as web services or an application program interface (API). Your system could even invoke behavior which other systems depend on, such as a batch job which updates an external database.

d. Your data source(s). Other systems may read or write to data sources "owned" by your system. These systems depend on the data, or at least portions of the data.

e. Your system interface(s). Other systems may interact with yours by accessing your data sources or via your own system interface(s).

There are several techniques ways to identify your system interaction perimeter:

a. Read your system documentation. Your existing documentation, if any, should indicate how your system interfaces to other legacy assets and how they interact with it. Don't assume that this documentation is complete and correct. Even though your organization may be meticulous in maintaining documentation it is possible that an interaction was introduced by another team that hasn't been documented.

b. Seek help from others. Enterprise architects, administrators, and operations staff are good people to involve because they have to work with multiple systems on a daily basis. Your enterprise administrators, particularly information and network administrators, may be your best bet as they're responsible for managing the assets in production.

c. Analyze the code. Without accurate documentation, or access to knowledgeable people, your last resort may be to analyze the source code for the legacy system, including code which invokes your system such as Job Control Language (JCL). This effort is often referred to as software archaeology.

2. *Identifying Legacy Assets*

Once you've identified a potential interface which your system is involved with you need to identify the other systems/assets involved with that interface. Knowing the interface exists doesn't automatically mean that you know which systems are using/supplying that interface. Ideally you shouldn't need to know this information, but realistically you sometimes do. For example, your organization may have a collection of web services provided by a variety of systems which yours may reuse. You will need to

know what services are available and what their signatures are, but you likely don't need to know the underlying system(s) offering each service. However, you may be accessing a legacy database which is owned by another team. Minimally you'll need to know the structure and semantics of the data which you're accessing. However, you may also need to know which systems provide the data you're using and the way in which they provide it (e.g. in batch refreshed daily at 2 am Greenwich mean time) to determine if you really want to use that data source. Ideally the contract model describing the database contains this sort of information, if not you may need to do the legacy analysis to obtain it.



The thing to realize is that your system is part of the interaction perimeters of each of the external legacy assets, so you must look for the sort of interactions listed earlier from the point of view of those assets. It isn't hard but it is usually tedious and time consuming.

3. *Analyzing the Interaction(s)*

Not only do you need to know which systems are coupled to yours, you need to understand how they're coupled to your system. Your goal is to identify the way that these systems are coupled to your systems interaction perimeter. Issues to look for during your analysis include:

a. Direction of the interaction. As noted previously, the direction of the interaction has different potential impacts.

b. Information being exchanged. You need to identify which data, often down to the element level, is being accessed and how it is being used. This will help you to identify replacement data sources, if any, and how effective they are at covering the original need. For example an external system may access your database to obtain a complete list of products offered within the city of Atlanta. If all other data sources only relate products to the states in which they're sold then the external system will no longer have the preciseness which it had before.

c. Functionality being invoked. You need to identify the exact services (operations, procedures, functions, and so on) being invoked at the interaction perimeter.

d. Frequency and volume. The frequency and the volume of the interaction are important pieces will indicate the load which you will put on the new sources for those interactions – they may not be able to handle the additional stress without infrastructure upgrades. Or, in the case of removing load the new sources may be over-powered, motivating your enterprise administrators to reconfigure the hardware and/or network resources for those systems and to divert them where they are needed more.

### *2.3.5 Analyzing Legacy Data Sources*

Data analysis is usually the most difficult part of analyzing system interaction. The data structures of your system and the system(s) your interfaces to will often be different, the database vendors can vary, the types of data sources (for example relational databases versus XML Files versus IMS) will vary, and worse yet the informational semantics are also likely to be different. For example, consider an ice cream company. One database maintains a flavors table which contains rows for chocolate, strawberry, and vanilla. The replacement data source contains a table with the exact same layout which maintains the flavors mocha fudge, ultimate chocolate, double chocolate, wild strawberry, winter strawberry, French vanilla, and old-fashioned vanilla. The new table arguably supports chocolate, strawberry, and vanilla yet there is clearly a semantics problem which may be very difficult to overcome. Table below summarizes common legacy data challenges which you may encounter.

| Challenge | Description |
|---|---|
| **Data quality problems** | There is a wide range of quality problems when it comes to data. This includes: a column or table that is used for more than one purpose, missing or inconsistent data, incorrect formatting of data, multiple sources for the same data, important entities or relationships which are stored as text fields, data values that stray from their field descriptions and business rules, several key strategies for the same type of entity, different data types for similar columns, and varying default values. This list is nowhere near complete but it should give you a feeling for the challenges which you will face. |
| **Design problems** | For example access to data sources may not be well encapsulated or the encapsulation strategies may be difficult to use. |
| **Architecture problems** | Architecture problems are also a serious problem with many legacy assets. Common data-oriented problems include applications which are responsible for data cleansing (instead of the database) and varying timeliness of data. |

### *2.3.6 Analyzing Legacy Code*

Code analysis is also a significant challenge, as Table below overviews. The legacy assets may:

- Be written in languages which your team is not familiar with.
- Not follow a consistent set of coding standards.
- Be difficult to understand.
- Not have up-to-date source code available (or it may not be clear which version to use).

| Challenge | Description |
|---|---|

| | |
|---|---|
| Code quality problems | Similarly you are likely to find problems in legacy code, such as inconsistent naming conventions, inconsistent or missing internal documentation, inconsistent operation semantics and/or signatures, highly coupled and brittle code, low cohesion operations which do several unrelated things, and code that is simply difficult to read. |
| Design problems | Applications may be poorly layered, with the user interface directly accessing the database for example. An asset may be of high quality but its original design goals may be at odds with current project needs; for example it may be a batch system whereas you need real-time access. |
| Architecture problems | Architecture problems are also a serious problem with many legacy assets. Common problems include: incompatible or difficult to integrate (often proprietary) platforms, fragmented and/or redundant data sources, inflexible architectures which are difficult to change, lack of event notification making it difficult to support real-time integration, and insufficient security. |

4. *Keeping it Agile by:*

a. Keep it simple. Contract models are agile documents which are just barely good enough, they don't need to be perfect.

b. Work in an evolutionary manner. You don't need to develop all of the documentation for all of the system interactions up front. Instead you can work iteratively, fleshing out the contract model(s) a bit at a time. You should also work incrementally, creating the contract model(s) and then in turn the actual integration code as you need them.

c. Work closely with the legacy system owners. You need to work collaboratively and cooperatively with the legacy system owners if you are to succeed. Fundamentally they own the system which you need to access, they can easily prevent you from doing so. Furthermore, they are the experts, therefore they are the ones which should be actively involved with the legacy analysis efforts. This is yet another example of active stakeholder participation, where the stakeholders are the owners of the legacy asset(s).

d. Consider modeling a bit ahead. A good reason to "Model a Bit Ahead" is to ensure that a development team has sufficient information about an existing legacy asset which they need to integrate to.

e. Don't get hung up on the "one truth". Many projects go astray when the data professionals involved with them focus too heavily on the one truth above all else.

## 2.4 Object-oriented modeling for web applications

### *2.4.1 Object-oriented programming (OOP)*

Object-oriented programming (OOP) is a programming language model organized around "objects" rather than "actions" and data rather than logic. Historically, a program has been viewed as a logical procedure that takes input data, processes it, and produces output data.

The programming challenge was seen as how to write the logic, not how to define the data. Object-oriented programming takes the view that what we really care about are the objects we want to manipulate rather than the logic required to manipulate them. Examples of objects range from human beings (described by name, address, and so forth) to buildings and floors (whose properties can be described and managed) down to the little widgets on your computer desktop (such as buttons and scroll bars).

The first step in OOP is to identify all the objects you want to manipulate and how they relate to each other, an exercise often known as data modeling. Once you've identified an object, you generalize it as a class of objects (think of Plato's concept of the "ideal" chair that stands for all chairs) and define the kind of data it contains and any logic sequences that can manipulate it. Each distinct logic sequence is known as a method. A real instance of a class is called an "object" or, in some environments, an "instance of a class." The object or class instance is what you run in the computer. Its methods provide computer instructions and the class object characteristics provide relevant data. You communicate with objects - and they communicate with each other - with well-defined interfaces called messages.

The concepts and rules used in object-oriented programming provide these important benefits:

- The concept of a data class makes it possible to define subclasses of data objects that share some or all of the main class characteristics. Called inheritance, this property of OOP forces a more thorough data analysis, reduces development time, and ensures more accurate coding.

- Since a class defines only the data it needs to be concerned with, when an instance of that class (an object) is run, the code will not be able to accidentally access other program data. This characteristic of data hiding provides greater system security and avoids unintended data corruption.

- The definition of a class is reuseable not only by the program for which it is initially created but also by other object-oriented programs (and, for this reason, can be more easily distributed for use in networks).

- The concept of data classes allows a programmer to create any new data type that is not already defined in the language itself.

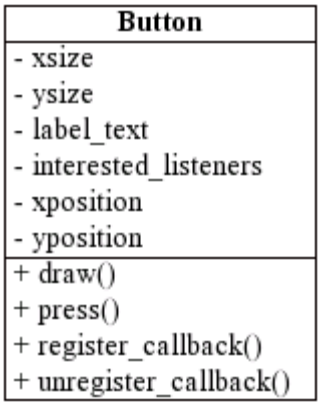
Object-oriented approach encourages the programmer to place data where it is not directly accessible by the rest of the program. Instead, the data is accessed by calling specially written functions, commonly called methods, which are either bundled in with the data or inherited from "class objects." These act as the intermediaries for retrieving or modifying the data they control. The programming construct that combines data with a set of methods for accessing and managing those data is called an object. The practice of using subroutines to examine or modify certain kinds of data, however, was also quite commonly used in non-OOP modular programming, well before the widespread use of object-oriented programming.

An object-oriented program will usually contain different types of objects, each type corresponding to a particular kind of complex data to be managed or perhaps to a real-world object or concept such as a bank account, a hockey player, or a bulldozer. A program might well contain multiple copies of each type of object, one for each of the real-world objects the program is dealing with. For instance, there could be one bank account object for each real-world account at a particular bank. Each copy of the bank account object would be alike in the methods it offers for manipulating or reading its data, but the data inside each object would differ reflecting the different history of each account.

Objects can be thought of as wrapping their data within a set of functions designed to ensure that the data are used appropriately, and to assist in that use. The object's methods will typically include checks and safeguards that are specific to the types of data the object contains. An object can also offer simple-to-use, standardized methods for performing particular operations on its data, while concealing the specifics of how those tasks are accomplished. In this way alterations can be made to the internal structure or methods of an object without requiring that the rest of the program be modified. This approach can also be used to offer standardized methods across different types of objects. As an example, several different types of objects might offer print methods. Each type of object might implement that print method in a different way, reflecting the different kinds of data each contains, but all the different print methods might be called in the same standardized manner from elsewhere in the program. These features become especially useful when more than one programmer is contributing code to a project or when the goal is to reuse code between projects.

***Class***

A class is a construct that is used as a blueprint to create instances of itself – referred to as class instances, class objects, instance objects or simply objects. A class defines constituent members which enable these class instances to have state and behavior. Data field members (member variables or instance variables) enable a class object to maintain state. Other kinds of members, especially methods, enable a class object's behavior.






***Instance***

An instance is an occurrence or a copy of an object, whether currently executing or not. Instances of a class share the same set of attributes, yet will typically differ in what those attributes contain. For example, a class "Employee" would describe the attributes common to all instances of the Employee class. For the purposes of the task being solved Employee objects may be generally alike, but vary in such attributes as "name" and "salary". The description of the class would itemize such attributes and define the operations or actions relevant for the class, such as "increase salary" or "change telephone number." One could then talk about one instance of the Employee object with name = "Jane Doe" and another instance of the Employee object with name = "John Doe".

***Object***

Object refers to a particular instance of a class. In the domain of object-oriented programming an object is usually taken to mean an ephemeral compilation of attributes (object elements) and behaviors (methods or subroutines) encapsulating an entity. In this way, while primitive or simple data types are still just single pieces of information, object-oriented objects are complex types that have multiple pieces of information and specific properties (or attributes). Instead of merely being assigned a value, (like int =10), objects have to be "constructed". In the real world, if a Ford Focus is an "object" - an instance of the car class, its physical properties and its function to drive would have been individually specified. Once the properties of the Ford Focus "object" had been specified into the form of the car class, it can be endlessly copied to create identical objects that look and function in just the same way. As an alternative example, animal is a superclass of primate and primate is a superclass of human. Individuals such as Joe Bloggs or John Doe would be particular examples or 'objects' of the human class, and consequently possess all the characteristics of the human class (and of the primate and animal superclasses as well).

Three properties characterize objects:

1. Identity: the property of an object that distinguishes it from other objects

2. State: describes the data stored in the object
3. Behavior: describes the methods in the object's interface by which the object can be used

***Method***

Method is a subroutine (or procedure or function) associated with a class. Methods define the behavior to be exhibited by instances of the associated class at program run time. Methods have the special property that at runtime, they have access to data stored in an instance of the class (or class instance or class object or object) they are associated with and are thereby able to control the state of the instance. The association between class and method is called binding. A method associated with a class is said to be bound to the class. Methods can be bound to a class at compile time (static binding) or to an object at runtime (dynamic binding).

Non-virtual methods are those which do not participate in polymorphism. Virtual methods are the means by which a class object can achieve polymorphic behavior.

Overloaded methods are those with the same name but different formal parameters or return value type, if the language supports overloading on return type.

Overridden methods are those that are redefined in a subclass and hide methods of a superclass.

An abstract method is one with only a signature and no implementation body. It is often used to specify that a subclass must provide an implementation of the method. Abstract methods are used to specify interfaces in some computer languages.

***Types of methods***

- Constructors - A constructor, is a class method that is called automatically at the beginning of an object's lifetime to initialize the object, a process called construction (or instantiation). Initialization may include acquisition of resources. A language may provide a means to control whether a constructor can be called implicitly (by the compiler) or only explicitly (by the programmer). Constructors may have parameters but usually do not return values in most languages.

- Destructors - A destructor is a class method that is called automatically at the end of an object's lifetime, a process called destruction. Destructors in most languages do not allow destructor method arguments nor return values. Destructors can be implemented such as to perform clean up chores and other tasks at object destruction.

- Class methods - Class methods are methods that are called on a class

- Static methods - Static methods neither require an instance of the class nor can they implicitly access the data (or this, self, Me, etc.) of such an instance. A static method is distinguished in some programming languages with the static keyword placed somewhere in the method's signature. Static methods are called "static" because they are resolved statically (i.e. at compile time) based on the class they are called on; and not dynamically, as in the case with instance methods which are resolved polymorphically based on the runtime type of the object. Therefore, static methods cannot be overridden.

- Accessor and mutator methods - An accessor method is a method that is usually small, simple and provides the sole means for the state of an object to be accessed (retrieved) from other parts of a program.

***Encapsulation***

Encapsulation is "the process of compartmentalizing the elements of an abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation." The purpose is to achieve potential for change: the internal mechanisms of the component can be improved without impact on other components, or the component can be replaced with a different one that supports the same public interface. Encapsulation also protects the integrity of the component, by preventing users from setting the internal data of the component into an invalid or inconsistent state. Another benefit of encapsulation is that it reduces system complexity and thus increases robustness, by limiting the interdependencies between software components.

Encapsulating software behind an interface allows the construction of objects that mimic the behavior and interactions of objects in the real world. For example, a simple digital alarm clock is a real-world object that a lay person can use and understand. They can understand what the alarm clock does, and how to use it through the provided interface (buttons and screen), without having to understand every part inside of the clock. Similarly, if you replaced the clock with a different model, the lay person could continue to use it in the same way, provided that the interface works the same.

***Inheritance***

Inheritance is a way to compartmentalize and reuse code by creating collections of attributes and behaviors called objects that can be based on previously created objects. In classical inheritance where objects are defined by classes, classes can inherit other classes. The new classes, known as subclasses (or derived classes), inherit attributes and behavior (i.e. previously coded algorithms) of the pre-existing classes, which are referred to as superclasses, ancestor classes or base classes. The inheritance relationships of classes gives rise to a hierarchy. In prototype-based programming, objects can be defined

directly from other objects without the need to define any classes, in which case this feature is called differential inheritance.

***Container***

Container is a class, a data structure, or an abstract data type (ADT) whose instances are collections of other objects. In other words; they are used for storing objects in an organized way following specific access rules. The size of the container depends on the number of the objects (elements) it contains. The underlying implementation of various types of containers may vary in space and time complexity allowing for flexibility in choosing the right implementation for a given scenario.

Container classes are expected to implement methods to do the following:

- create a new empty container (constructor),
- report the number of objects it stores (size),
- delete all the objects in the container (clear),
- insert new objects into the container,
- remove objects from it,
- provide access to the stored objects.

***Polymorphism***

Polymorphism is a programming language feature that allows values of different data types to be handled using a uniform interface. The concept of parametric polymorphism applies to both data types and functions. A function that can evaluate to or be applied to values of different types is known as a polymorphic function. A data type that can appear to be of a generalized type (e.g., a list with elements of arbitrary type) is designated polymorphic data type like the generalized type from which such specializations are made.

## 2.5 Unified Modeling Language

Unified Modeling Language (UML) is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created, by the Object Management Group. UML includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems.

The Unified Modeling Language (UML) is used to specify, visualize, modify, construct and document the artifacts of an object-oriented software-intensive system under development. UML offers a standard way to visualize a system's architectural blueprints, including elements such as:

- Activities - is a major task that must take place in order to fulfill an operation contract. An activity can represent:
    - the invocation of an operation
    - a step in a business process
    - An entire business process.
- Actors - specifies a role played by a user or any other system that interacts with the subject.
- Business processes - is a collection of related, structured activities or tasks that produce a specific service or product (serve a particular goal) for a particular customer or customers
- Database schemas
- Logical components - represents a modular part of a system, that encapsulates its content and whose manifestation is replaceable within its environment. A component defines its behavior in terms of provided and required interfaces
- Programming language statements
- Reusable software components.

UML combines techniques from data modeling (entity relationship diagrams), business modeling (work flows), object modeling, and component modeling. It can be used with all processes, throughout the software development life cycle, and across different implementation technologies. UML has synthesized the notations of the Booch method, the Object-modeling technique (OMT) and Object-oriented software engineering (OOSE) by fusing them into a single, common and widely usable modeling language. UML aims to be a standard modeling language which can model concurrent and distributed systems. UML is a de facto industry standard, and is evolving under the auspices of the Object Management Group (OMG).

UML models may be automatically transformed to other representations (e.g. Java) by means of QVT-like transformation languages. UML is extensible, with two mechanisms for customization: profiles and stereotypes.

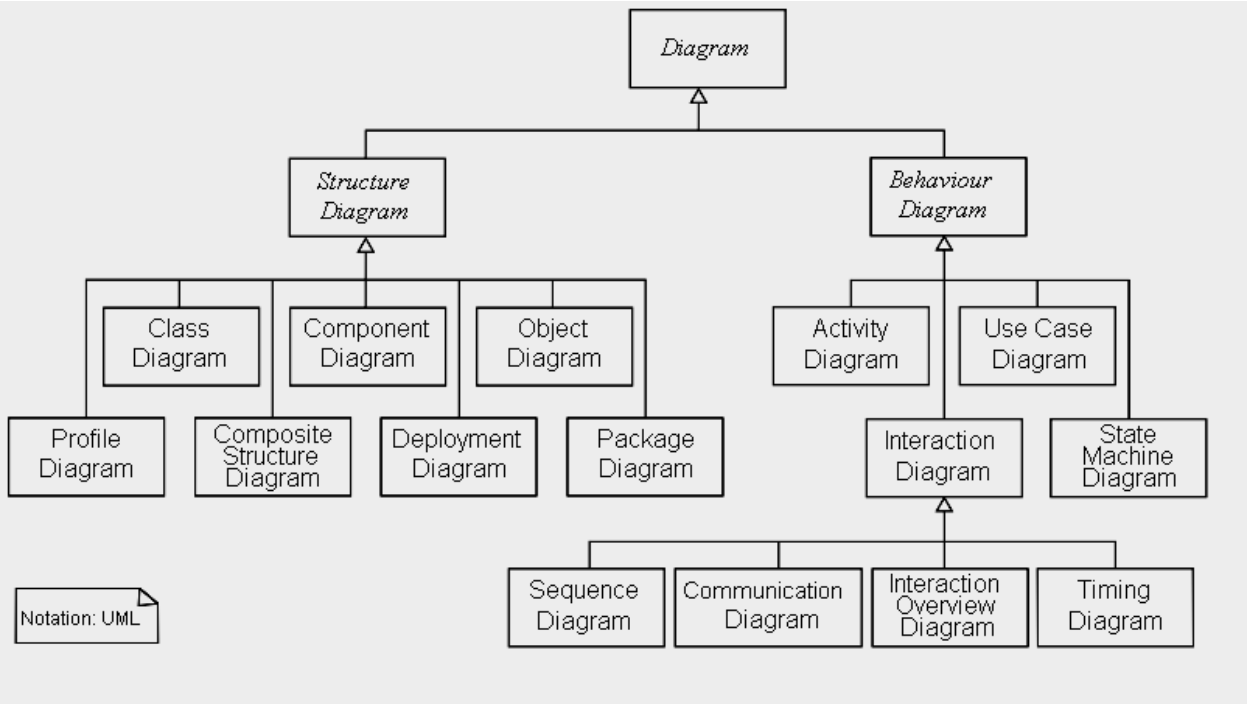
### *2.5.1 UML Diagrams*

UML diagrams represent two different views of a system model:

- Static (or structural) view: emphasizes the static structure of the system using objects, attributes, operations and relationships. The structural view includes class diagrams and composite structure diagrams.

- Dynamic (or behavioral) view: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams and state machine diagrams.

UML has 14 types of diagrams divided into two categories. Seven diagram types represent structural information, and the other seven represent general types of behavior, including four that represent different aspects of interactions. These diagrams can be categorized hierarchically as shown in the following class diagram:




1. **Structure diagrams**

Structure diagrams emphasize the things that must be present in the system being modeled. Since structure diagrams represent the structure, they are used extensively in documenting the software architecture of software systems.

- Class diagram: describes the structure of a system by showing the system's classes, their attributes, and the relationships among the classes.

- Component diagram: describes how a software system is split up into components and shows the dependencies among these components.

- Composite structure diagram: describes the internal structure of a class and the collaborations that this structure makes possible.
- Deployment diagram: describes the hardware used in system implementations and the execution environments and artifacts deployed on the hardware.
- Object diagram: shows a complete or partial view of the structure of an example modeled system at a specific time.
- Package diagram: describes how a system is split up into logical groupings by showing the dependencies among these groupings.
- Profile diagram: operates at the metamodel level to show stereotypes as classes with the <<stereotype>> stereotype, and profiles as packages with the <<profile>> stereotype. The extension relation (solid line with closed, filled arrowhead) indicates what metamodel element a given stereotype is extending.

2. **Behaviour diagrams**

Behaviour diagrams emphasize what must happen in the system being modelled. Since behaviour diagrams illustrate the behavior of a system, they are used extensively to describe the functionality of software systems.

- Activity diagram: describes the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.
- UML state machine diagram: describes the states and state transitions of the system.
- Use case diagram: describes the functionality provided by a system in terms of actors, their goals represented as use cases, and any dependencies among those use cases.

3. **Interaction diagrams**

Interaction diagrams, a subset of behaviour diagrams, emphasize the flow of control and data among the things in the system being modeled:

- Communication diagram: shows the interactions between objects or parts in terms of sequenced messages. They represent a combination of information taken from Class, Sequence, and Use Case Diagrams describing both the static structure and dynamic behavior of a system.
- Interaction overview diagram: provides an overview in which the nodes represent communication diagrams.

- Sequence diagram: shows how objects communicate with each other in terms of a sequence of messages. Also indicates the lifespans of objects relative to those messages.
- Timing diagrams: a specific type of interaction diagram where the focus is on timing constraints.



## Chapter Review Questions

1. Discuss the three tier internet architecture
2. What are legacy systems?
3. What are problems associated by legacy systems
4. What is UML?
5. Discuss the following terms in object oriented
    a. Class
    b. Instance
    c. Object
    d. Inheritance
    e. Polymorphism

## Suggested Further Reading

1. Alex H et all(2000), Professional active server Wrox publishers programmer to programmer series
2. Michael V. Ekedahl and William Newman(2003), Programming with Microsoft Visual Basic.NET: An Object-Oriented Approach, Course Technology, ISBN 0619239204

# CHAPTER THREE: ACTIVE SERVER PAGES (ASP)

***Learning Objectives:***

*By the end of this chapter the learner shall be able to;*

i. Explain what is ASP

ii. Installing ISS server

iii. Write and run ASP files

## 3.1 Introduction to ASP

Microsoft® Active Server Pages (ASP) is a server-side scripting technology that can be used to create dynamic and interactive Web applications. An ASP page is an HTML page that contains server-side scripts that are processed by the Web server before being sent to the user's browser. You can combine ASP with Extensible Markup Language (XML), Component Object Model (COM), and Hypertext Markup Language (HTML) to create powerful interactive Web sites.

Server-side scripts run when a browser requests an .asp file from the Web server. ASP is called by the Web server, which processes the requested file from top to bottom and executes any script commands. It then formats a standard Web page and sends it to the browser.

It is possible to extend your ASP scripts using COM components and XML. COM extends your scripting capabilities by providing a compact, reusable, and secure means of gaining access to information. You can call components from any script or programming language that supports Automation. XML is a meta-markup language that provides a format to describe structured data by using a set of tags.

Active Server Pages (ASP) was Microsoft's first server-side script-engine for dynamically-generated web pages. Initially released as an add-on to Internet Information Services (IIS) via the Windows NT 4.0 Option Pack, it was subsequently included as a free component of Windows Server (since the initial release of Windows 2000 Server). ASP.NET has superseded ASP.

ASP 2.0 provided six built-in objects: Application, ASPError, Request, Response, Server, and Session. Session, for example, represents a cookie-based session that maintains the state of variables from page to page. The Active Scripting engine's support of the Component Object Model (COM) enables ASP websites to access functionality in compiled libraries such as DLLs.

Web pages with the .asp file extension use ASP, although some web sites disguise their choice of scripting language for security purposes (e.g. still using the more common .htm or .html extension). Pages with the .aspx extension use compiled ASP.NET (based on Microsoft's .NET Framework), which makes them faster and more robust than server-side scripting in ASP, which is interpreted at run-time; however, ASP.NET pages may still include some ASP scripting. The introduction of ASP.NET led to use of the term Classic ASP for the original technology.



### *3.1.1 What is ASP?*

* ASP stands for Active Server Pages

* ASP is a Microsoft Technology

* ASP is a program that runs inside IIS (Internet Information Services)

* IIS comes as a free component with Windows 7

### *3.1.2 ASP Compatibility*

* To run IIS you must have Windows XP or 7

* ChiliASP is a technology that runs ASP without Windows OS

* InstantASP is another technology that runs ASP without Windows

### *3.1.3 What is an ASP File?*

* An ASP file is just the same as an HTML file

* An ASP file can contain text, HTML, XML, and scripts

* Scripts in an ASP file are executed on the server

* An ASP file has the file extension ".asp"

### *3.1.4 How Does ASP Differ from HTML?*

* When a browser requests an HTML file, the server returns the file

* When a browser requests an ASP file, IIS passes the request to the ASP engine. The ASP engine reads the ASP file, line by line, and executes the scripts in the file. Finally, the ASP file is returned to the browser as plain HTML

### *3.1.5 What can ASP do for you?*

* Dynamically edit, change, or add any content of a Web page

* Respond to user queries or data submitted from HTML forms

* Access any data or databases and return the results to a browser

* Customize a Web page to make it more useful for individual users

* The advantages of using ASP instead of CGI and Perl, are those of simplicity and speed

* Provide security - since ASP code cannot be viewed from the browser

* Clever ASP programming can minimize the network traffic

### *3.1.6 IIS - Internet Information Server*

IIS is a set of Internet-based services for servers created by Microsoft for use with Microsoft Windows. IIS comes with Windows 2000, XP, Vista, and Windows 7. It is also available for Windows NT. IIS is easy to install and ideal for developing and testing web applications.

### *3.1.7 PWS - Personal Web Server*

PWS is for older Windows system like Windows 95, 98, and NT. PWS is easy to install and can be used for developing and testing web applications including ASP. It is not recommended for running on the internet due to the fact that it is outdated and has security issues.

## 3.2 Running and Installing ISS in your machine

### *3.2.1 Your Windows PC as a Web Server*

- Your own PC can act as a web server if you install IIS or PWS
- IIS or PWS turns your computer into a web server
- Microsoft IIS and PWS are free web server components

### *3.2.2 How to Install IIS on Windows 7 and Windows Vista*

Follow these steps to install IIS:

1. Open the Control Panel from the Start menu
2. Double-click Programs and Features
3. Click "Turn Windows features on or off" (a link to the left)
4. Select the check box for Internet Information Services (IIS), and click OK

After you have installed IIS, make sure you install all patches for bugs and security problems. (Run Windows Update).



### *3.2.3 How to Install IIS on Windows XP and Windows 2000*

Follow these steps to install IIS:

1. On the Start menu, click Settings and select Control Panel
2. Double-click Add or Remove Programs
3. Click Add/Remove Windows Components
4. Click Internet Information Services (IIS)
5. Click Details
6. Select the check box for World Wide Web Service, and click OK
7. In Windows Component selection, click Next to install IIS

After you have installed IIS, make sure you install all patches for bugs and security problems. (Run Windows Update).

### *3.2.4 Test Your Web*

After you have installed IIS or PWS follow these steps:

1. Look for a new folder called Inetpub on your hard drive
2. Open the Inetpub folder, and find a folder named wwwroot
3. Create a new folder, like "MyWeb", under wwwroot
4. Write some ASP code and save the file as "test1.asp" in the new folder
5. Make sure your Web server is running (see below)
6. Open your browser and type "http://localhost/MyWeb/test1.asp", to view your first web page

Note: Look for the IIS (or PWS) symbol in your start menu or task bar. The program has functions for starting and stopping the web server, disable and enable ASP, and much more.

## 3.3 ASP Basic Syntax Rules

### *3.3.1 Write Output to a Browser*



An ASP file normally contains HTML tags, just like an HTML file. However, an ASP file can also contain server scripts, surrounded by the delimiters <% and %>. Server scripts are executed on the server, and can contain any expressions, statements, procedures, or operators valid for the scripting language you prefer to use.

*The response.write Command*

The response.write command is used to write output to a browser. The following example sends the text "Hello World" to the browser:

Example

```
<html>

<body>

<%

response.write("Hello World!")

%>

</body>

</html>
```

There is also a shorthand method for the response.write command. The following example also sends the text "Hello World" to the browser:

```
<html>

<body>

<%

="Hello World!"

%>

</body>

</html>
```

### *3.3.2 Using VBScript in ASP*

You can use several scripting languages in ASP. However, the default scripting language is VBScript:

```
<html>

<body>

<%

response.write("Hello World!")

%>

</body>

</html>
```

### *3.3.3 Using JavaScript in ASP*

To set JavaScript as the default scripting language for a particular page you must insert a language specification at the top of the page:

```
<%@ language="javascript"%>

<html>

<body>

<%

Response.Write("Hello World!")

%>

</body>

</html>
```

Note: JavaScript is case sensitive! You will have to write your ASP code with uppercase letters and lowercase letters when the language requires it.

### *3.3.4 Add some HTML tags to a text*

Example

```
<html>
```

```
<body>

<%

response.write("<h2>You can use HTML tags to format the text!</h2>")

%>

<%

response.write("<p style='color:#0000ff'>This text is styled with the style attribute!</p>")

%>

</body>

</html>
```

## 3.4 ASP Variables

A variable is used to store information.

### *3.4.1 Declare a variable*

Variables are used to store information. This example demonstrates how to declare a variable, assign a value to it, and use the value in a text.

```
<html>

<body>

<%

dim name

name="Donald Duck"

response.write("My name is: " & name)

%>

</body>

</html>
```

### *3.4.2 Declare an array*

Arrays are used to store a series of related data items. This example demonstrates how to declare an array that stores names.

```
<html>

<body>

<%

Dim famname(5),i

famname(0) = "Jan Egil"

famname(1) = "Tove"

famname(2) = "Hege"

famname(3) = "Stale"

famname(4) = "Kai Jim"

famname(5) = "Borge"

For i = 0 to 5

    response.write(famname(i) & "<br />")

Next

%>

</body>

</html>
```

### *3.4.3 Loop through the HTML headings*

How to loop through the six headings in HTML.

```
<html>

<body>

<%

dim i
```

```
for i=1 to 6
  response.write("<h" & i & ">Heading " & i & "</h" & i & ">")
next
%>
</body>
</html>
```



### *3.4.4 Time-based greeting using VBScript*

This example will display a different message to the user depending on the time on the server.

```
<html>
<body>
<%
dim h
h=hour(now())
response.write("<p>" & now())
response.write("</p>")
If h<12 then
  response.write("Good Morning!")
else
  response.write("Good day!")
end if
%>
</body>
</html>
```

### *3.4.5 Time-based greeting using JavaScript*

This example is the same as the one above, but the syntax is different.

```
<%@ language="javascript" %>

<html>

<body>

<%

var d=new Date()

var h=d.getHours()

Response.Write("<p>")

Response.Write(d)

Response.Write("</p>")

if (h<12)

  {

  Response.Write("Good Morning!")

  }

else

  {

  Response.Write("Good day!")

  }

%>

</body>

</html>
```

### *3.4.6 Lifetime of Variables*

A variable declared outside a procedure can be accessed and changed by any script in the ASP file. A variable declared inside a procedure is created and destroyed every time the procedure is executed. No

scripts outside the procedure can access or change the variable. To declare variables accessible to more than one ASP file, declare them as session variables or application variables.

*Session Variables*

Session variables are used to store information about ONE single user, and are available to all pages in one application. Typically information stored in session variables are name, id, and preferences.

*Application Variables*

Application variables are also available to all pages in one application. Application variables are used to store information about ALL users in one specific application.

## 3.5 ASP Procedures

In ASP you can call a JavaScript procedure from a VBScript and vice versa.

### *3.5.1 Procedures*

The ASP source code can contain procedures and functions:

Example

```
<html>

<head>

<%

sub vbproc(num1,num2)

response.write(num1*num2)

end sub

%>

</head>

<body>

<p>Result: <%call vbproc(3,4)%></p>

</body>

</html>
```

Insert the <%@ language="language" %> line above the <html> tag to write the procedure/function in another scripting language:

Example

```
<%@ language="javascript" %>

<html>

<head>

<%

function jsproc(num1,num2)

{

Response.Write(num1*num2)

}

%>

</head>

<body>

<p>Result: <%jsproc(3,4)%></p>

</body>

</html>
```

### *3.5.2 Differences between VBScript and JavaScript*

When calling a VBScript or a JavaScript procedure from an ASP file written in VBScript, you can use the "call" keyword followed by the procedure name. If a procedure requires parameters, the parameter list must be enclosed in parentheses when using the "call" keyword. If you omit the "call" keyword, the parameter list must not be enclosed in parentheses. If the procedure has no parameters, the parentheses are optional. When calling a JavaScript or a VBScript procedure from an ASP file written in JavaScript, always use parentheses after the procedure name.

### *3.5.3 Call procedures using VBScript*

How to call both a JavaScript procedure and a VBScript procedure in an ASP file.

```
<html>
<head>
<%
sub vbproc(num1,num2)
Response.Write(num1*num2)
end sub
%>
<script  language="javascript" runat="server">
function jsproc(num1,num2)
{
Response.Write(num1*num2)
}
</script>
</head>

<body>
<p>Result: <%call vbproc(3,4)%></p>
<p>Result: <%call jsproc(3,4)%></p>
</body>
</html>
```

## 3.6 ASP Forms and User Input

The Request.QueryString and Request.Form commands are used to retrieve user input from forms.

### *3.6.1 User Input*

The Request object can be used to retrieve user information from forms.

*Example HTML form*

<form method="get" action="simpleform.asp">

First Name: <input type="text" name="fname" /><br />

Last Name: <input type="text" name="lname" /><br /><br />

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

</form>

User input can be retrieved with the Request.QueryString or Request.Form command.

### *3.6.2 Request.QueryString*

The Request.QueryString command is used to collect values in a form with method="get".

Information sent from a form with the GET method is visible to everyone (it will be displayed in the browser's address bar) and has limits on the amount of information to send.

If a user typed "Bill" and "Gates" in the HTML form above, the URL sent to the server would look like this:

http://www.w3schools.com/simpleform.asp?fname=Bill&lname=Gates

Assume that "simpleform.asp" contains the following ASP script:

<body>

Welcome

<%

response.write(request.querystring("fname"))

response.write(" " & request.querystring("lname"))

%>

</body>

### *3.6.3 Request.Form*

The Request.Form command is used to collect values in a form with method="post".

Information sent from a form with the POST method is invisible to others and has no limits on the amount of information to send.

If a user typed "Bill" and "Gates" in the HTML form above, the URL sent to the server would look like this:

http://www.w3schools.com/simpleform.asp

Assume that "simpleform.asp" contains the following ASP script:

<body>

Welcome

<%

response.write(request.form("fname"))

response.write(" " & request.form("lname"))

%>

</body>

### *3.6.4 Form Validation*

User input should be validated on the browser whenever possible (by client scripts). Browser validation is faster and reduces the server load.

You should consider server validation if the user input will be inserted into a database. A good way to validate a form on the server is to post the form to itself, instead of jumping to a different page. The user will then get the error messages on the same page as the form. This makes it easier to discover the error.

### *3.6.5 A form with method="get"*

How to interact with the user, with the Request.QueryString command.

<html>

<body>

<form action="demo_reqquery.asp" method="get">

Your name: <input type="text" name="fname" size="20" />

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

```
</form>
<%
dim fname
fname=Request.QueryString("fname")
If fname<>"" Then
    Response.Write("Hello " & fname & "!<br />")
    Response.Write("How are you today?")
End If
%>
</body>
</html>
```

### *3.6.6 A form with method="post"*

How to interact with the user, with the Request.Form command.

```
<html>
<body>
<form action="demo_simpleform.asp" method="post">
Your name: <input type="text" name="fname" size="20" />
<input type="submit" value="Submit" />
</form>
<%
dim fname
fname=Request.Form("fname")
If fname<>"" Then
    Response.Write("Hello " & fname & "!<br />")
```

```
    Response.Write("How are you today?")
End If
%>
</body>
</html>
```

### *3.6.8 A form with radio buttons*

How to interact with the user, through radio buttons, with the Request.Form command.

```
<html>
<%
dim cars
cars=Request.Form("cars")
%>
<body>
<form action="demo_radiob.asp" method="post">
<p>Please select your favorite car:</p>
<input type="radio" name="cars"
<%if cars="Volvo" then Response.Write("checked")%>
value="Volvo">Volvo</input>
<br />
<input type="radio" name="cars"
<%if cars="Saab" then Response.Write("checked")%>
value="Saab">Saab</input>
<br />
<input type="radio" name="cars"
```

```
<%if cars="BMW" then Response.Write("checked")%>

value="BMW">BMW</input>

<br /><br />

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

</form>

<%

if cars<>"" then

  Response.Write("<p>Your favorite car is: " & cars & "</p>")

end if

%>

</body>

</html>
```

## 3.7 ASP Conditional Statements

Conditional statements are used to perform different actions based on different conditions.

### *3.7.1 The If...Then Statement*

The simplest technique used to validate a condition is to check whether it is true. This can be done using an If...Then statement. The syntax to use is:

    If ConditionToCheck is True Then Statement

The program examines a condition, in this case ConditionToCheck. This ConditionToCheck can be a simple expression or a combination of expressions. If the ConditionToCheck is true, then the program will execute the Statement.

There are two ways you can use the If...Then statement. If the conditional formula is short enough, you can write it on one line, like this:

    If ConditionToCheck is True Then Statement

If there are many statements to execute as a truthful result of the condition, you should write the statements on alternate lines. Of course, you can use this technique even if the condition you are

examining is short. In this case, one very important rule to keep is to terminate the conditional statement with End If. The syntax used is:

```
If ConditionToCheck is True Then
    Statement
End If
```

***Example***

```
<%@ Language="VBScript" %>
<html>

<head>
<title>CD Publisher</title>
</head>

<body>
<h1>CD Publisher</h1>
<form method="GET" action="cdpublisher2.asp">
 <table border="0" width="553">
  <tr>
   <td width="191">Number of CDs Ordered:</td>
   <td width="113"><input type="text" name="txtQuantity" size="10"
    value=<% =Request.QueryString("txtQuantity") %> >
   </td>
   <td width="67"><input type="submit" value="Calculate" name="btnCalculate"></td>
   <td width="162"><input type="reset" value="Start New Order" name="btnReset"></td>
  </tr>
```

```
  </table>
</form>
<%
    Dim Quantity
    Dim UnitPrice
    Dim TotalPrice
    Dim strCD

    Quantity = CInt(Request.QueryString("txtQuantity"))
    strCD = "CDs"
    If Quantity = 1 Then strCD = "CD"

    Response.Write("<p>Since you ordered " & Quantity & " " & strCD & ",<br>")
       UnitPrice = 8.50
    If Quantity < 50 Then UnitPrice = 6.25
    Response.Write("Each CD will cost:      " & UnitPrice & "<br>")
    TotalPrice = Quantity * UnitPrice
    Response.Write("And the total price is: " & TotalPrice & "</p>")
%>
</body>
</html>
```

### *3.7.2 The If...Then...Else Statement*

The If...Then statement offers only one alternative: to act if the condition is true. Whenever you would like to apply an alternative in case the condition is false, you can use the If...Then...Else statement. The formula of this statement is:

```
If ConditionToCheck is True Then
        Statement1
Else
        Statement2
End If
```

When this section of code executes, if the ConditionToCheck is true, then the first statement, Statement1, is executed. If the ConditionToCheck is false, the second statement, in this case Statement2, is executed.

***Example***

```
<%@ Language="VBScript" %>
<html>
<head>
<title>CD Publisher</title>
</head>
<body>
<h1>CD Publisher</h1>
<form method="GET" action="cdpublisher2.asp">
 <table border="0" width="553">
  <tr>
   <td width="191">Number of CDs Ordered:</td>
   <td width="113"><input type="text" name="txtQuantity" size="10"
    value=<% =Request.QueryString("txtQuantity") %> >
   </td>
   <td width="67"><input type="submit" value="Calculate" name="btnCalculate"></td>
   <td width="162"><input type="reset" value="Start New Order" name="btnReset"></td>
  </tr>
```

```
  </table>
</form>
<%
    Dim Quantity
    Dim UnitPrice
    Dim TotalPrice
    Dim strCD
    Quantity = CInt(Request.QueryString("txtQuantity"))
    strCD = "CDs"
    If Quantity = 1 Then strCD = "CD"
        Response.Write("<p>Since you ordered " & Quantity & " " & strCD & ",<br>")
        If Quantity < 20 Then
            UnitPrice = 20
    Else
            UnitPrice = 6.25
    End If
    Response.Write("Each CD will cost:      " & UnitPrice & "<br>")
    TotalPrice = Quantity * UnitPrice
    Response.Write("And the total price is: " & TotalPrice & "</p>")
%>
</body>
</html>
```

### *3.7.3 The If...Then...ElseIf Statement*

The If...Then...ElseIf statement acts like the If...Then...Else expression, except that it offers as many choices as necessary. The syntax to use is:

```
If Condition1 is True Then
    Statement1
ElseIf Condition2 is True Then
    Statement2
ElseIf Conditionk is True Then
    Statementk
End If
```

The program will first examine Condition1. If Condition1 is true, the program will execute Statment1 and stop examining conditions. If Condition1 is false, the program will examine Condition2 and act accordingly. Whenever a condition is false, the program will continue examining the conditions until it finds one. Once a true condition has been found and its statement executed, the program will terminate the conditional examination at End If.

There is still a possibility that none of the stated conditions is true. In this case, you should provide a "catch all" condition. This is done with a last Else section. The Else section must be the last in the list of conditions and would act if none of the primary conditions is true. The formula to use would be:

```
If Condition1 is True Then
    Statement1
ElseIf Condition2 is True Then
    Statement2
ElseIf Conditionk is True Then
    Statementk
Else
    CatchAllStatement
End If
```

***Example***

```
<%@ Language="VBScript" %>
```

```
<html>
<head>
<title>CD Publisher</title>
</head>
<body>
<h1>CD Publisher</h1>
<form method="GET" action="cdpublisher2.asp">
 <table border="0" width="553">
  <tr>
   <td width="191">Number of CDs Ordered:</td>
   <td width="113"><input type="text" name="txtQuantity" size="10"
    value=<% =Request.QueryString("txtQuantity") %> >
   </td>
   <td width="67"><input type="submit" value="Calculate" name="btnCalculate"></td>
   <td width="162"><input type="reset" value="Start New Order" name="btnReset"></td>
  </tr>
 </table>
</form>
<%
  Dim Quantity
  Dim UnitPrice
  Dim TotalPrice
  Dim strCD
  Quantity = CInt(Request.QueryString("txtQuantity"))
```

```
    strCD = "CDs"
    If Quantity = 1 Then strCD = "CD"
        Response.Write("<p>Since you ordered " & Quantity & " " & strCD & ",<br>")
        ' The price of one CD will depend on the number ordered
    ' The more the customer orders, the lower value each
    If Quantity < 20 Then
        UnitPrice = 20
    ElseIf Quantity < 50 Then
        UnitPrice = 15
    ElseIf Quantity < 100 Then
        UnitPrice = 12
    ElseIf Quantity < 500 Then
        UnitPrice = 8
    Else
        UnitPrice = 5
    End If
    Response.Write("Each CD will cost:      " & UnitPrice & "<br>")
    TotalPrice = Quantity * UnitPrice
    Response.Write("And the total price is: " & TotalPrice & "</p>")
%>
</body>
</html>
```

### *3.7.4 The Select Case Statement*

If you have a large number of conditions to examine, the If...Then...Else will go through each one of them. Microsoft Visual Basic offers the alternative of jumping to the statement that applies to the state of the condition.



The syntax of the Select Case is:

```
Select Case Expression
   Case Expression1
      Statement1
   Case Expression2
      Statement2
   Case Expressionk
      Statementk
End Select
```

The Expression will be examined and evaluated once. Then Microsoft Visual Basic will compare the result of this examination with the Expression of each case. Once it finds one that matches, it would execute the corresponding Statement.

If you anticipate that there could be no match between the Expression and one of the Expressions, you can use a Case Else statement at the end of the list. The formula to use would be:

```
Select Case Expression
   Case Expression1
      Statement1
   Case Expression2
      Statement2
   Case Expressionk
      Statementk
   Case Else
```

Statementk

End Select



## 3.8 ASP Loops

A loop is an expression used to repeat an action. Microsoft Visual Basic presents many variations of the loops and they combine the Do and the Loop keywords.

### *3.8.1 The Do...While Loop*

The syntax of the Do... While loop is:

Do While Condition

Statement(s)

Loop

This expression examines the Condition. If the condition is true, then it executes the Statement or statements. After executing the statement(s), it goes back to examine the Condition. AS LONG AS the Condition is true, the Statement will be executed and the Condition will be tested again. If the Condition is false or once the condition becomes false, the statement will not be executed and the the program will move on. As you may guess already, the Condition must provide a way for it to be true and to be false.

***example:***

```
<%@ Language="VBScript" %>

<html>

<head>

<title>Active Server Pages Tutorials</title>

</head>

<body>

<%

Dim Counter

Counter = 0

Do While Counter <= 12
```

```
    Response.Write(Counter & " ")

    Counter = Counter + 1

  Loop

%>

</body>

</html>
```



### *3.8.2 The Do...Loop...While Statement*

Since the Do...While statement tests the Condition first before executing the Statement, sometimes you will want the program to execute the Statement first, then go back and test the Condition. Microsoft Visual Basic offers a reverse to the formula as follows:

```
Do

  Statement(s)

Loop While Condition
```

In this case, the Statement or Statements will be executed first. Then the Condition will be tested. If the Condition is true, the program will execute the Statement again. The program will continue this examination-execution as long as the Condition is true. The big difference here is that even if the Condition is false, the program will have executed the Condition at least once.

***example:***

```
<%@ Language="VBScript" %>

<html>

<head>

<title>Active Server Pages Tutorials</title>

</head>

<body>

<%

  Dim Counter
```

```
    Counter = 2.25
  Do
      Response.Write(Counter & "<br>")
    Counter = Counter + 0.35
  Loop While Counter <= 6
%>
</body>
</html>
```

### *3.8.3 The Do...Until...Loop Statement*

An alternative to the Do...While loop is the Do...Until loop. Its syntax is:

```
Do Until Condition
  Statement(s)
Loop
```

This loop will first examine the Condition, instead of examining whether the Condition is true, it will test whether the Condition is false.

***example***:

```
<%@ Language="VBScript" %>
<html>
<head>
<title>Active Server Pages Tutorials</title>
</head>
<body>
<%
  Dim InterestRate
  InterestRate = 7.25
```

```
    Do Until InterestRate = 16
        Response.Write("Interest Rate: " & InterestRate & "%<br>")
        InterestRate = InterestRate + 0.25
    Loop
%>
</body>
</html>
```

### *3.8.4 The Do...Loop...Until Statement*

An alternative to the Do...Until...loop consists of executing the the Statement first. The syntax used is:

```
Do
    Statement(s)
Loop Until Condition
```

This expression executes the Statement first. After executing the Statement, it examines the Condition. If the Condition is False, then it goes back and executes the Statement again and re-check the Condition. Once the Condition becomes true, it would stop and move on; but as long as the Condition is False, the Statement would be executed.

***example***:

```
<%@ Language="VBScript" %>
<html>
<head>
<title>Active Server Pages Tutorials</title>
</head>
<body>
```

```
<%
   Dim InterestRate
   InterestRate = 7.25
   Do
      Response.Write("Interest Rate: " & InterestRate & "%<br>")
      InterestRate = InterestRate + 0.25
   Loop Until InterestRate = 16
%>
</body>
</html>
```

## 3.9 ASP Loop Counters

The looping statements we reviewed above are used when you don't know or can't anticipate the number of times a condition needs to be checked in order to execute a statement. If you know with certainty how many times you want to execute a statement, you can use another form of loops that use the For...Next expression.

### *3.9.1 The For...To...Next Loop*

One of the loop counters you can use is For...To...Next. Its syntax is:

```
For Counter = Start To End
  Statement(s)
Next
```

Used for counting, the expression begins counting at the Start point. Then it examines whether the current value (after starting to count) is greater than End. If that's the case, it then executes the Statement(s). Next, it increments the value of Counter by 1 and examines the condition again. This process goes on until the value of Counter becomes equal to the End value. Once this condition is reached, the looping stops.

***example***:

```
<%@ Language="VBScript" %>
```

```
<html>

<head>

<title>Active Server Pages Tutorials</title>

</head>

<body>

<%

   Dim Counter

      For Counter = 0 to 26

      Response.Write(Counter & " ")

   Next

%>

</body>

</html>
```

### *3.9.2 Stepping the Counting Loop*

The formula above will increment the counting by 1 at the end of each statement. If you want to control how the incrementing processes, you can set your own, using the Step option. Here is the formula:

```
For Counter = Start To End Step Increment

  Statement(s)

Next Counter
```

You can set the incrementing value to your choice. If the value of Increment is positive, the Counter will be added its value. This means that you can give it a negative value, in which case the Counter will be subtracted the set value.

***example***:

```
<%@ Language="VBScript" %>

<html>
```

```
<head>
<title>Active Server Pages Tutorials</title>
</head>
<body>

<%
   Dim Counter
      For Counter = 0 to 46 Step 2
      Response.Write(Counter & " ")
   Next
%>
</body>
</html>
```

## 3.10 ASP Cookies

A cookie is often used to identify a user. A cookie is often used to identify a user. A cookie is a small file that the server embeds on the user's computer. Each time the same computer requests a page with a browser, it will send the cookie too. With ASP, you can both create and retrieve cookie values.

### *3.10.1 How to Create a Cookie?*

The "Response.Cookies" command is used to create cookies.

**Note:** The Response.Cookies command must appear BEFORE the <html> tag.

In the example below, we will create a cookie named "firstname" and assign the value "Alex" to it:

```
<%
Response.Cookies("firstname")="Alex"
%>
```

It is also possible to assign properties to a cookie, like setting a date when the cookie should expire:

```
<%
Response.Cookies("firstname")="Alex"
Response.Cookies("firstname").Expires=#May 10,2012#
%>
```

### *3.10.2 How to Retrieve a Cookie Value?*

The "Request.Cookies" command is used to retrieve a cookie value.

In the example below, we retrieve the value of the cookie named "firstname" and display it on a page:

```
<%
fname=Request.Cookies("firstname")
response.write("Firstname=" & fname)
%>
```

### *3.10.3 A Cookie with Keys*

If a cookie contains a collection of multiple values, we say that the cookie has Keys.

In the example below, we will create a cookie collection named "user". The "user" cookie has Keys that contains information about a user:

```
<%
Response.Cookies("user")("firstname")="John"
Response.Cookies("user")("lastname")="Smith"
Response.Cookies("user")("country")="Norway"
Response.Cookies("user")("age")="25"
%>
```

### *3.10.4 Read all Cookies*

Look at the following code:

```
<%
Response.Cookies("firstname")="Alex"
Response.Cookies("user")("firstname")="John"
Response.Cookies("user")("lastname")="Smith"
Response.Cookies("user")("country")="Norway"
```

```
Response.Cookies("user")("age")="25"
%>
```

Assume that your server has sent all the cookies above to a user.



Now we want to read all the cookies sent to a user. The example below shows how to do it (note that the code below checks if a cookie has Keys with the HasKeys property):

```
<html>
<body>

<%
dim x,y
for each x in Request.Cookies
 response.write("<p>")
 if Request.Cookies(x).HasKeys then
  for each y in Request.Cookies(x)
   response.write(x & ":" & y & "=" & Request.Cookies(x)(y))
   response.write("<br />")
  next
 else
  Response.Write(x & "=" & Request.Cookies(x) & "<br />")
 end if
 response.write "</p>"
next
%>

</body>
</html>
```

***Example: How to create a Welcome cookie.***

```
<%
dim numvisits
response.cookies("NumVisits").Expires=date+365
numvisits=request.cookies("NumVisits")

if numvisits="" then
```

```
  response.cookies("NumVisits")=1
  response.write("Welcome! This is the first time you are visiting this Web page.")
else
  response.cookies("NumVisits")=numvisits+1
  response.write("You have visited this ")
  response.write("Web page " & numvisits)
  if numvisits=1 then
    response.write " time before!"
  else
    response.write " times before!"
  end if
end if
%>
<html>
<body>
</body>
</html>
```



### *3.10.5 What if a Browser Does NOT Support Cookies?*

If your application deals with browsers that do not support cookies, you will have to use other methods to pass information from one page to another in your application. There are two ways of doing this:

1. Add parameters to a URL

You can add parameters to a URL:

```
<a href="welcome.asp?fname=John&lname=Smith">Go to Welcome Page</a>
```

And retrieve the values in the "welcome.asp" file like this:

```
<%
fname=Request.querystring("fname")
lname=Request.querystring("lname")
response.write("<p>Hello " & fname & " " & lname & "!</p>")
response.write("<p>Welcome to my Web site!</p>")
%>
```

2. Use a form

You can use a form. The form passes the user input to "welcome.asp" when the user clicks on the Submit button:

```
<form method="post" action="welcome.asp">
First Name: <input type="text" name="fname" value="" />
Last Name: <input type="text" name="lname" value="" />
<input type="submit" value="Submit" />
</form>
```

Retrieve the values in the "welcome.asp" file like this:

```
<%
fname=Request.form("fname")
lname=Request.form("lname")
response.write("<p>Hello " & fname & " " & lname & "!</p>")
response.write("<p>Welcome to my Web site!</p>")
%>
```

## 3.11 ASP Session Object

A Session object stores information about, or change settings for a user session.

### *3.11.1 The Session object*

When you are working with an application on your computer, you open it, do some changes and then you close it. This is much like a Session. The computer knows who you are. It knows when you open the application and when you close it. However, on the internet there is one problem: the web server does not know who you are and what you do, because the HTTP address doesn't maintain state.

ASP solves this problem by creating a unique cookie for each user. The cookie is sent to the user's computer and it contains information that identifies the user. This interface is called the Session object.

The Session object stores information about, or change settings for a user session.

Variables stored in a Session object hold information about one single user, and are available to all pages in one application. Common information stored in session variables are name, id, and preferences. The

server creates a new Session object for each new user, and destroys the Session object when the session expires.



### *3.11.2 When does a Session Start?*

A session starts when:

- A new user requests an ASP file, and the Global.asa file includes a Session_OnStart procedure
- A value is stored in a Session variable
- A user requests an ASP file, and the Global.asa file uses the <object> tag to instantiate an object with session scope

### *3.11.3 When does a Session End?*

A session ends if a user has not requested or refreshed a page in the application for a specified period. By default, this is 20 minutes.

If you want to set a timeout interval that is shorter or longer than the default, use the **Timeout** property.

The example below sets a timeout interval of 5 minutes:

```
<%
Session.Timeout=5
%>
```

Use the **Abandon** method to end a session immediately:

```
<%
Session.Abandon
%>
```

**Note:** The main problem with sessions is WHEN they should end. We do not know if the user's last request was the final one or not. So we do not know how long we should keep the session "alive". Waiting too long for an idle session uses up resources on the server, but if the session is deleted too soon the user has to start all over again because the server has deleted all the information. Finding the right timeout interval can be difficult!

**Tip:** Only store SMALL amounts of data in session variables!

### *3.11.4 Store and Retrieve Session Variables*

The most important thing about the Session object is that you can store variables in it.

The example below will set the Session variable *username* to "Donald Duck" and the Session variable *age* to "50":

```
<%
Session("username")="Donald Duck"
Session("age")=50
%>
```

When the value is stored in a session variable it can be reached from ANY page in the ASP application:

```
Welcome <%Response.Write(Session("username"))%>
```

The line above returns: "Welcome Donald Duck".

You can also store user preferences in the Session object, and then access that preference to choose what page to return to the user.

The example below specifies a text-only version of the page if the user has a low screen resolution:

```
<%If Session("screenres")="low" Then%>
  This is the text version of the page
<%Else%>
  This is the multimedia version of the page
<%End If%>
```

### *3.11.5 Remove Session Variables*

The Contents collection contains all session variables. It is possible to remove a session variable with the Remove method. The example below removes the session variable "sale" if the value of the session variable "age" is lower than 18:

```
<%
If Session.Contents("age")<18 then
  Session.Contents.Remove("sale")
End If
%>
```

To remove all variables in a session, use the RemoveAll method:

```
<%
Session.Contents.RemoveAll()
%>
```



### *3.11.6 Loop Through the Contents Collection*

The Contents collection contains all session variables. You can loop through the Contents collection, to see what's stored in it:

```
<%
Session("username")="Donald Duck"
Session("age")=50

dim i
For Each i in Session.Contents
  Response.Write(i & "<br />")
Next
%>
```

Result:

```
username
age
```

If you do not know the number of items in the Contents collection, you can use the Count property:

```
<%
dim i
dim j
j=Session.Contents.Count
Response.Write("Session variables: " & j)
For i=1 to j
  Response.Write(Session.Contents(i) & "<br />")
Next
%>
```

Result:

Session variables: 2

Donald Duck

50



### *3.11.7 Loop Through the StaticObjects Collection*

You can loop through the StaticObjects collection, to see the values of all objects stored in the Session object:

```
<%
dim i
For Each i in Session.StaticObjects
  Response.Write(i & "<br />")
Next
%>
```

## 3.12 ASP Application Object

A group of ASP files that work together to perform some purpose is called an application.

### *3.12.1 Application Object*

An application on the Web may consists of several ASP files that work together to perform some purpose. The Application object is used to tie these files together. The Application object is used to store and access variables from any page, just like the Session object. The difference is that ALL users share ONE Application object (with Sessions there is ONE Session object for EACH user). The Application object holds information that will be used by many pages in the application (like database connection information). The information can be accessed from any page. The information can also be changed in one place, and the changes will automatically be reflected on all pages.

### *3.12.2 Store and Retrieve Application Variables*

Application variables can be accessed and changed by any page in an application. You can create Application variables in "Global.asa" like this:

```
<script language="vbscript" runat="server">

Sub Application_OnStart
```

```
application("vartime")=""

application("users")=1

End Sub

</script>
```

In the example above we have created two Application variables: "vartime" and "users". You can access the value of an Application variable like this:

There are

```
<%

Response.Write(Application("users"))

%>
```

active connections.

### *3.12.3 Loop Through the Contents Collection*

The Contents collection contains all application variables. You can loop through the Contents collection, to see what's stored in it:

```
<%

dim i

For Each i in Application.Contents

  Response.Write(i & "<br />")

Next

%>
```

If you do not know the number of items in the Contents collection, you can use the Count property:

```
<%
```

```
dim i

dim j

j=Application.Contents.Count

For i=1 to j

  Response.Write(Application.Contents(i) & "<br />")

Next

%>
```

### *3.12.4 Loop Through the StaticObjects Collection*

You can loop through the StaticObjects collection, to see the values of all objects stored in the Application object:

```
<%

dim i

For Each i in Application.StaticObjects

  Response.Write(i & "<br />")

Next

%>
```

### *3.12.5 Lock and Unlock*

You can lock an application with the "Lock" method. When an application is locked, the users cannot change the Application variables (other than the one currently accessing it). You can unlock an application with the "Unlock" method. This method removes the lock from the Application variable:

```
<%

Application.Lock
```

'do some application object operations

Application.Unlock

%>



## 3.13 ASP Including Files

### *3.13.1 The #include Directive*

You can insert the content of one ASP file into another ASP file before the server executes it, with the #include directive. The #include directive is used to create functions, headers, footers, or elements that will be reused on multiple pages.

### *3.13.2 How to Use the #include Directive*

Here is a file called "mypage.asp":

```
<html>
<body>
<h3>Words of Wisdom:</h3>
<p><!--#include file="wisdom.inc"--></p>
<h3>The time is:</h3>
<p><!--#include file="time.inc"--></p>
</body>
</html>
```

Here is the "wisdom.inc" file:

"One should never increase, beyond what is necessary,
the number of entities required to explain anything."

Here is the "time.inc" file:

```
<%
Response.Write(Time)
%>
```

If you look at the source code in a browser, it will look something like this:

```
<html>
<body>
<h3>Words of Wisdom:</h3>
<p>"One should never increase, beyond what is necessary,
the number of entities required to explain anything."</p>
<h3>The time is:</h3>
<p>11:33:42 AM</p>
</body>
</html>
```

### *3.13.3 Syntax for Including Files*

To include a file in an ASP page, place the #include directive inside comment tags:

```
<!--#include virtual="somefilename"-->
```

or

```
<!--#include file ="somefilename"-->
```

### *3.13.4 The Virtual Keyword*

Use the virtual keyword to indicate a path beginning with a virtual directory.

If a file named "header.inc" resides in a virtual directory named /html, the following line would insert the contents of "header.inc":

```
<!-- #include virtual ="/html/header.inc" -->
```

### *3.13.5 The File Keyword*

Use the file keyword to indicate a relative path. A relative path begins with the directory that contains the including file. If you have a file in the html directory, and the file "header.inc" resides in html\headers, the following line would insert "header.inc" in your file:

```
<!-- #include file ="headers\header.inc" -->
```

Note that the path to the included file (headers\header.inc) is relative to the including file. If the file containing this #include statement is not in the html directory, the statement will not work.

***Tips and Notes***

In the sections above we have used the file extension ".inc" for included files. Notice that if a user tries to browse an INC file directly, its content will be displayed. If your included file contains confidential information or information you do not want any users to see, it is better to use an ASP extension. The source code in an ASP file will not be visible after the interpretation. An included file can also include other files, and one ASP file can include the same file more than once.

**Important:** Included files are processed and inserted before the scripts are executed. The following script will NOT work because ASP executes the #include directive before it assigns a value to the variable:

```
<%
fname="header.inc"
%>
<!--#include file="<%fname%>"-->
```

You cannot open or close a script delimiter in an INC file. The following script will NOT work:

```
<%
For i = 1 To n
  <!--#include file="count.inc"-->
Next
%>
```

But this script will work:

```
<% For i = 1 to n %>
  <!--#include file="count.inc" -->
<% Next %>
```

## 3.14 ASP The Global.asa file

### *3.14.1 The Global.asa file*

The Global.asa file is an optional file that can contain declarations of objects, variables, and methods that can be accessed by every page in an ASP application. All valid browser scripts (JavaScript, VBScript, JScript, PerlScript, etc.) can be used within Global.asa.

The Global.asa file can contain only the following:

- Application events
- Session events
- <object> declarations
- TypeLibrary declarations
- the #include directive

Note: The Global.asa file must be stored in the root directory of the ASP application, and each application can only have one Global.asa file.

### *3.14.2 Events in Global.asa*

In Global.asa you can tell the application and session objects what to do when the application/session starts and what to do when the application/session ends. The code for this is placed in event handlers. The Global.asa file can contain four types of events:

**Application_OnStart** - Occurs when the FIRST user calls the first page in an ASP application. This event occurs after the Web server is restarted or after the Global.asa file is edited. The "Session_OnStart" event occurs immediately after this event.

**Session_OnStart** - This event occurs EVERY time a NEW user requests his or her first page in the ASP application.

**Session_OnEnd** - This event occurs EVERY time a user ends a session. A user-session ends after a page has not been requested by the user for a specified time (by default this is 20 minutes).

**Application_OnEnd** - This event occurs after the LAST user has ended the session. Typically, this event occurs when a Web server stops. This procedure is used to clean up settings after the Application stops, like delete records or write information to text files.

A Global.asa file could look something like this:

```
<script language="vbscript" runat="server">

sub Application_OnStart

'some code

end sub

sub Application_OnEnd

'some code
```

```
end sub


sub Session_OnStart

'some code

end sub

sub Session_OnEnd

'some code

end sub

</script>
```

Note: Because we cannot use the ASP script delimiters (<% and %>) to insert scripts in the Global.asa file, we put subroutines inside an HTML <script> element.

### *3.14.3 <object> Declarations*

It is possible to create objects with session or application scope in Global.asa by using the <object> tag.

Note: The <object> tag should be outside the <script> tag!

Syntax

```
<object runat="server" scope="scope" id="id" {progid="progID"|classid="classID"}>

....

</object>
```

| **Parameter** | **Description** |
|---|---|
| scope | Sets the scope of the object (either Session or Application) |
| id | Specifies a unique id for the object |
| ProgID | An id associated with a class id. The format for ProgID is [Vendor.]Component[.Version]<br><br>Either ProgID or ClassID must be specified. |
| ClassID | Specifies a unique id for a COM class object. |

| | |
|---|---|
| | Either ProgID or ClassID must be specified. |



*Examples*

*The first example creates an object of session scope named "MyAd" by using the ProgID parameter:*

*<object runat="server" scope="session" id="MyAd" progid="MSWC.AdRotator">*

*</object>*

*The second example creates an object of application scope named "MyConnection" by using the ClassID parameter:*

*<object runat="server" scope="application" id="MyConnection"*

*classid="Clsid:8AD3067A-B3FC-11CF-A560-00A0C9081C21">*

*</object>*

*The objects declared in the Global.asa file can be used by any script in the application:*

*GLOBAL.ASA:*

*<object runat="server" scope="session" id="MyAd" progid="MSWC.AdRotator">*

*</object>*

*You could reference the object "MyAd" from any page in the ASP application:*

*SOME .ASP FILE:*

*<%=MyAd.GetAdvertisement("/banners/adrot.txt")%>*

### *3.14.4 TypeLibrary Declarations*

A TypeLibrary is a container for the contents of a DLL file corresponding to a COM object. By including a call to the TypeLibrary in the Global.asa file, the constants of the COM object can be accessed, and errors can be better reported by the ASP code. If your Web application relies on COM objects that have declared data types in type libraries, you can declare the type libraries in Global.asa.

Syntax

<!--METADATA TYPE="TypeLib"

file="filename" uuid="id" version="number" lcid="localeid"

-->

*id*"
-->



| Parameter | Description |
|---|---|
| file | Specifies an absolute path to a type library.<br><br>Either the file parameter or the uuid parameter is required |
| uuid | Specifies a unique identifier for the type library.<br><br>Either the file parameter or the uuid parameter is required |
| version | Optional. Used for selecting version. If the requested version is not found, then the most recent version is used |
| lcid | Optional. The locale identifier to be used for the type library |

### *3.15.5 Error Values*

The server can return one of the following error messages:

| Error Code | Description |
|---|---|
| ASP 0222 | Invalid type library specification |
| ASP 0223 | Type library not found |
| ASP 0224 | Type library cannot be loaded |
| ASP 0225 | Type library cannot be wrapped |

**Note:** METADATA tags can appear anywhere in the Global.asa file (both inside and outside <script> tags). However, it is recommended that METADATA tags appear near the top of the Global.asa file.

***Restrictions***

Restrictions on what you can include in the Global.asa file:

- You cannot display text written in the Global.asa file. This file can't display information

- You can only use Server and Application objects in the Application_OnStart and Application_OnEnd subroutines. In the Session_OnEnd subroutine, you can use Server, Application, and Session objects. In the Session_OnStart subroutine you can use any built-in object



### *3.14.6 How to use the Subroutines*

Global.asa is often used to initialize variables. The example below shows how to detect the exact time a visitor first arrives on a Web site. The time is stored in a Session variable named "started", and the value of the "started" variable can be accessed from any ASP page in the application:

```
<script language="vbscript" runat="server">

sub Session_OnStart

Session("started")=now()

end sub

</script>
```

Global.asa can also be used to control page access. The example below shows how to redirect every new visitor to another page, in this case to a page called "newpage.asp":

```
<script language="vbscript" runat="server">

sub Session_OnStart

Response.Redirect("newpage.asp")

end sub

</script>
```

And you can include functions in the Global.asa file.

In the example below the Application_OnStart subroutine occurs when the Web server starts. Then the Application_OnStart subroutine calls another subroutine named "getcustomers". The "getcustomers" subroutine opens a database and retrieves a record set from the "customers" table. The record set is assigned to an array, where it can be accessed from any ASP page without querying the database.



```
<script language="vbscript" runat="server">

sub Application_OnStart

getcustomers

end sub

sub getcustomers

set conn=Server.CreateObject("ADODB.Connection")

conn.Provider="Microsoft.Jet.OLEDB.4.0"

conn.Open "c:/webdata/northwind.mdb"

set rs=conn.execute("select name from customers")

Application("customers")=rs.GetRows

rs.Close

conn.Close

end sub

</script>
```

### *3.14.7 Global.asa Example*

In this example we will create a Global.asa file that counts the number of current visitors.

- The Application_OnStart sets the Application variable "visitors" to 0 when the server starts
- The Session_OnStart subroutine adds one to the variable "visitors" every time a new visitor arrives

- The Session_OnEnd subroutine subtracts one from "visitors" each time this subroutine is triggered



The Global.asa file:

```
<script language="vbscript" runat="server">

Sub Application_OnStart

Application("visitors")=0

End Sub

Sub Session_OnStart

Application.Lock

Application("visitors")=Application("visitors")+1

Application.UnLock

End Sub

Sub Session_OnEnd

Application.Lock

Application("visitors")=Application("visitors")-1

Application.UnLock

End Sub

</script>
```

To display the number of current visitors in an ASP file:

```
<html>

<head>

</head>
```

```
<body>

<p>There are <%response.write(Application("visitors"))%> online now!</p>

</body>

</html>
```

## 3.15 ASP Sending e-mail with CDOSYS

DOSYS is a built-in component in ASP. This component is used to send e-mails with ASP.

### *3.15.1 Sending e-mail with CDOSYS*

CDO (Collaboration Data Objects) is a Microsoft technology that is designed to simplify the creation of messaging applications. CDOSYS is a built-in component in ASP. We will show you how to use this component to send e-mail with ASP.

### *3.15.2 How about CDONTs?*

Microsoft has discontinued the use of CDONTs on Windows 2000, Windows XP and Windows 2003. If you have used CDONTs in your ASP applications, you should update the code and use the new CDO technology.

Examples using CDOSYS

Sending a text e-mail:

```
<%

Set myMail=CreateObject("CDO.Message")

myMail.Subject="Sending email with CDO"

myMail.From="mymail@mydomain.com"

myMail.To="someone@somedomain.com"

myMail.TextBody="This is a message."

myMail.Send

set myMail=nothing

%>
```

Sending a text e-mail with Bcc and CC fields:

```
<%
Set myMail=CreateObject("CDO.Message")
myMail.Subject="Sending email with CDO"
myMail.From="mymail@mydomain.com"
myMail.To="someone@somedomain.com"
myMail.Bcc="someoneelse@somedomain.com"
myMail.Cc="someoneelse2@somedomain.com"
myMail.TextBody="This is a message."
myMail.Send
set myMail=nothing
%>
```

Sending an HTML e-mail:

```
<%
Set myMail=CreateObject("CDO.Message")
myMail.Subject="Sending email with CDO"
myMail.From="mymail@mydomain.com"
myMail.To="someone@somedomain.com"
myMail.HTMLBody = "<h1>This is a message.</h1>"
myMail.Send
set myMail=nothing
%>
```

Sending an HTML e-mail that sends a webpage from a website:

```
<%
Set myMail=CreateObject("CDO.Message")
myMail.Subject="Sending email with CDO"
myMail.From="mymail@mydomain.com"
myMail.To="someone@somedomain.com"
myMail.CreateMHTMLBody "http://www.w3schools.com/asp/"
myMail.Send
set myMail=nothing
%>
```

Sending an HTML e-mail that sends a webpage from a file on your computer:

```
<%
Set myMail=CreateObject("CDO.Message")
myMail.Subject="Sending email with CDO"
myMail.From="mymail@mydomain.com"
myMail.To="someone@somedomain.com"
myMail.CreateMHTMLBody "file://c:/mydocuments/test.htm"
myMail.Send
set myMail=nothing
%>
```

Sending a text e-mail with an Attachment:

```
<%
Set myMail=CreateObject("CDO.Message")
myMail.Subject="Sending email with CDO"
myMail.From="mymail@mydomain.com"
```

```
myMail.To="someone@somedomain.com"
myMail.TextBody="This is a message."
myMail.AddAttachment "c:\mydocuments\test.txt"
myMail.Send
set myMail=nothing
%>
```

Sending a text e-mail using a remote server:

```
<%
Set myMail=CreateObject("CDO.Message")
myMail.Subject="Sending email with CDO"
myMail.From="mymail@mydomain.com"
myMail.To="someone@somedomain.com"
myMail.TextBody="This is a message."
myMail.Configuration.Fields.Item _
("http://schemas.microsoft.com/cdo/configuration/sendusing")=2
'Name or IP of remote SMTP server
myMail.Configuration.Fields.Item _
("http://schemas.microsoft.com/cdo/configuration/smtpserver")="smtp.server.com"
'Server port
myMail.Configuration.Fields.Item _
("http://schemas.microsoft.com/cdo/configuration/smtpserverport")=25
myMail.Configuration.Fields.Update
myMail.Send
set myMail=nothing
```

%>



## 3.16 Object-Oriented Programming in ASP

One of the nicest features of Active Server Pages is it's support for simple object-oriented programming (OOP). Creating object-oriented code makes your applications easier to read and easier to manage. In most cases, it will take a little bit more work than creating global methods and variables, but in the long run, it will more than make up for that effort.

### *3.16.1 Class Declaration*

Let's look at an example of how to declare a class in ASP:

```
Class clsImaginary

    Private mReal        ' real part

    Private mImag        ' imaginary part


    ' Constructor for the class

    Private Sub Class_Initialize

        mReal = 0.0

        mImag = 0.0

    End Sub

    ' Destructor for the class

    Private Sub Class_Terminate

        ' do cleanup code here

    End Sub


    ' methods

    Public Sub Initialize(vReal, vImag)
```

```
        mReal = vReal

        mImag = vImag

    End Sub


    Public Function Value

        Value = mReal & "+" & mImag & "i"

    End Function


    ' properties

    Public Property Get Real

        Real = mReal

    End Property


    Public Property Get Imag

        Imag = mImag

    End Property

End Class
```

This code contains the definition of a class encapsulated between the ***Class*** and the ***End Class*** keywords. It includes two private member variables, a constructor, a useless destructor, two methods and two properties. The order of sections doesn't matter at all except where readability is concerned. This ordering because it makes most declarations easy to find.

All elements within the class declaration contain a ***Private*** or ***Public*** access modifier. Private means that the method or variable can only be accessed with the class declaration (between the Class and End Class statements.) Public means that the method or variable can be accessed from either within the class or through the instance of a class (as we'll see later.)

Another thing you should take note of is that there can only be one constructor named ***Class_Initialize*** and it cannot take any arguments. This means I cannot create a constructor that takes a real and imaginary

component to initialize the member variables. That is the reason I needed to create the ***Initialize*** method to seed the initial value for the real number.

The constructor method (which must be named ***Class_Initialize***) will be invoked automatically when your class is created. Likewise the destructor (which must be named ***Class_Terminate***) will be invoked automatically when your class is destroyed. Our destructor method doesn't do anything because there is no need for it (so we could have just left this method out of the class declaration. A good use for a destructor is cleaning up resources that are in use such as a connection to a database or an open file handle.

The methods in the class declaration look just like global function declarations with the exception of the Public or Private modifiers. They behave exactly the same except that they must be invoked through the use of a class instance.

Within any of the method or property declarations, you still have access to all global methods and variables. The benefit of encapsulating them within a class is that you don't have to worry about name collisions. The class also creates it's own namespace allowing you to use short and simple names for your methods, properties and member variables.

Since classes are intended to be reusable pieces of code, it makes sense to place a class declaration or set of multiple class declarations within a separate file. Unlike languages like Java, there is no need to make the name of your file match your class name. This file can be included into your scripts that need to use the class. I usually create a base folder named lib that contains all of my include files.

### *3.16.2 Using a Class*

An example of making use of a class is shown below:

```
Dim oImag

Set oImag = New clsImaginary

Response.Write "Before Initialization: "

Response.Write oImag.Value

Call oImag.Initialize(3.443, 1.43)

Response.Write "

After Initialization: "

Response.Write oImag.Value
```

As you can see in the example, the steps are:

- Declare a variable
- ***Set*** the variable to be an instance of the class by using the ***New*** keyword
- Invoke a method of the class by using the dot (.) operator

There is no need to explicitly destory an instance of a class unless you have a specific terminiator that you want to be invoked. In this case you can use the following code to destroy the class:

```
Set oImag = Nothing
```

### *3.16.3 Member Variables*

Member variables are special variables that are encapsulated within your class declaration. They are not accessible outside of a class declaration unless you declare them with the Public modifier in which case they can only be accessed through a class instance.

I prefer to prefix my member variables with the letter "m", but you can use any naming convention you like. Previously, I used a letter to denote the value type being stored in the variable (ie: "s" for string, "n" for integer, "f" for float, "b" for boolean, "c" for char and "o" for object.)

When accessing member variables within a property or method, there is no need to use a this or similar keyword. These variables are simply accessed by using their variable name as shown in our example.

You cannot declare constant variables (using the Const keyword) within a class. One alternative would be to create a public member variable, but that would not really be a true constant because the value could be modified. A better option would be to create a property which only supports the Get method as shown below:

```
' constant value PI (as a property)

Public Property Get PI

    PI = 3.1415926

End Property
```

### *3.16.4 Constructors*

Constructors should always be declared as subroutines (using the Sub keyword.) Additionally, you should always declare your subroutines using the Private modifier since they will never be called directly using a class instance.

The use of a constructor and destructor declaration is totally optional. You can declare one or the other or neither if you choose. Most often, I use a constructor to initialize my member variables without a destructor.

### *3.16.5 Methods*

Methods are the real workhorse of the class. They perform all the heavy lifting of the class. They are basically functions and subroutines that are encapsulated within the class declaration. By using a Public or Private access modifier, you can hide or expose the method to the outside world.

Because the method must be accessed via a class instance, the method name can be very short. When using procedure programming, you would have to prefix your function names with a module prefix such as imagValue(). Because the class declaration create a kind of namespace, we can use the shorter method name Value.

Consider not using too many arguments within your method declaration. It is better to keep your argument list the same and just add new member variables and properties to pass values to the member variable. This will allow you to create new versions of your class that are backwards compatiable with older code you have written.

### *3.16.6 Properties*

Properties are used for accessing data within your class. They can be used for both reading and storing a data value. There are three different property types that can be defined:

- Get - Read a property value
- Let - Store a property value
- Set - Set a property value to an object instance

You will notice that there is only one property type for retrieving or reading a property value (Get). Because of this, you can use Get to return either a regular variable or an object instance. We've already seen in the previous example how the Property Get declaration works.

When declaring a Let or a Set property, you should only declare one argument which is used to pass the value or object instance you wish to assign. I typically use a variable named vValue but you can use

whatever you like. An example of using Property Let is included below. Set works the same way except you change Let to Set and the value must be assigned using the Set keyword.

Instead of making a member variable public and letting anyone store any type of value, keeping it private allows you to control what type of data goes into a variable and what type of data is read out.

Take for instance, a variable that represents the month of a year. This should hold a value between 1 and 12. We would like to make sure that only these values can be stored in the variable. Here is what the code would look like to do this:

```
Class clsImaginary

    Private mMonth          ' month of the year

    Private mImag           ' imaginary part

    ' month property

    Public Property Let TheMonth(vValue)

        If IsNumeric(vValue) Then

            If CInt(vValue) >= 1 And CInt(vValue) <= 12 Then

                mMonth = CInt(vValue)

            End If

        End if

    End Property

End Class
```

### *3.16.7 Summary*

This article summarizes the object-oriented features of the Active Server Pages language. ASP provides a basic framework that allows us to create powerful classes without too much effort. Object-oriented language features in ASP include:

- Member variables
- Constructors and Desctructors
- Methods (functions and subroutines)

- Private and Public access modifiers
- Let, Get and Set properties

Object-oriented features not supported in ASP:

- Constructors with initializers
- Method overloading
- Member constants (property serves this purpose)
- Operator overloading
- Inheritance
- True namespaces

Use of classes in your applications will help you avoid name collisions and to encapsulate critical data and methods that should only be used within a class. By using properties instead of global subroutines, you can keep your function arguments the same while adding new properties to your class. This will help to keep your methods backwards compatible with older code.

## Chapter Review Questions

1. What is ASP?
2. Differentiate between post  and get method
3. How do you use Request.QueryString?
4. Write and ASP code to illustrate the following conditional statements
    a. If
    b. If…Else
    c. Switch
5. Write and ASP code to illustrate the following loops
    a. Do while
    b. for

## Suggested Further Reading


1. Alex H et all(2000), Professional active server Wrox publishers programmer to programmer series
2. Michael V. Ekedahl and William Newman(2003), Programming with Microsoft Visual Basic.NET: An Object-Oriented Approach, Course Technology, ISBN 0619239204
3. Buser, Kauffman, Llibre, et al (1999), Beginning ASP 3.0, WROX Press

# CHAPTER FOUR: SAMPLE PAPERS

# Mt Kenya University

**UNIVERSITY EXAMINATION**

**SCHOOL OF PURE AND APPLIED SCIENCES**

**DEPARTMENT OF INFORMATION TECHNOLOGY**

**BACHELOR OF BUSINESS INFORMATION TECHNOLOGY**

**END OF SEMESTER EXAMINATION**

**BIT 3207: ADVANCED WEB DESIGN AND DEVELOPMENT FOR BUSINESS WORLD**

TIME: 2HRS

**Instructions:** Question ONE is **COMPULSORY** and any other TWO from section B. **SECTION A.**

**Question 1**

a. Outline the layers of TCP/IP protocol (6 Marks)

b. What is ASP? (2 Marks)

c. Name and briefly describe any FOUR (4) internet application areas (8 Marks)

d. Differentiate between the client side and server side script (4 Marks)

e. Using ASP response.write command write an ASP code to display "hello world" on the screen (10 Marks)

**Question 2**

a. Without a computer network the Internet would not have been possible. Explain the properties of a computer network (10 Marks)

b. Write an ASP code to illustrate the implementation of class (10 Marks)

**Question 3**

a. Differentiate between session variables and application variables (4 Marks)

b. What is VBScript? (2 Marks)

c. Write VBScript code to display "hello world" on the screen (6 Marks)

d. Using an ASP code example what is a Do..Loop? (4 Marks)

e. Using an ASP code example what is a Case Statement? (4 Marks)

**Question 4**

a. In Object Oriented programming can you explain the following terms (10 Marks)

i. Method

ii. Object

iii. Class

iv. Polymorphism

v. Inheritance

b. Discuss the three-tier architecture in web development environment (10 Marks)

**Question 5**

a. During web design we are mostly concerned with the "look and feel" of our website and you explain 5 Principles of interface design (10 Marks)

b. Using For loop and Tables write the ASP and HTML code to display the following window (10 Marks)