NAAC ACCREDITED

तेजस्वि नावधीतमस्तु

ISO 9001:2008 & 14001:2004

FAIRFIELD

Institute of Management & Technology

'A' Grade Institute by DHE, Govt. of NCT Delhi and Approved by the Bar Council of India and NCTE

Reference Material for Three Years

Bachelor of Computer Application

Code : 020

Semester – III

FIMT Campus, Kapashera, New Delhi-110037, Phones : 011-25063208/09/10/11, 25066256/ 57/58/59/60
Fax : 011-250 63212 Mob. : 09312352942, 09811568155 E-mail : fimtoffice@gmail.com Website : www.fimt-ggsipu.org

INDEX

Three Years

Bachelor of Computer Application

Code : 020

Semester – III

MATHEMATICS-III (201)

Mathematics-III (BCA-201)

<u>Unit-I (Measures of Central Tendency & Dispersion)</u>

<u>Definition</u>

Collecting data can be easy and fun. But sometimes it can be hard to tell other people about what you have found. That's why we use statistics. Two kinds of statistics are frequently used to describe data. They are measures of central tendency and dispersion. These are often called descriptive statistics because they can help you describe your data.

Mean median and mode

These are all measures of central tendency. They help summarize a bunch of scores with a single number. Suppose you want to describe a bunch of data that you collected to a friend for a particular variable like height of students in your class. One way would be to read each height you recorded to your friend. Your friend would listen to all of the heights and then come to a conclusion about how tall students generally are in your class But this would take too much time. Especially if you are in a class of 200 or 300 students! Another way to communicate with your friend would be to use measures of central tendency like the mean, median and mode. They help you summarize bunches of numbers with one or just a few numbers. They make telling people about your data easy.

Range, variance and standard deviation

These are all measures of dispersion. These help you to know the spread of scores within a bunch of scores. Are the scores really close together or are they really far apart? For example, if you were describing the heights of students in your class to a friend, they might want to know how much the heights vary. Are all the men about 5 feet 11 inches within a few centimeters or so? Or is there a lot of variation where some men are 5 feet and others are 6 foot 5 inches? Measures of dispersion like the range, variance and standard deviation tell you about the spread of scores in a data set. Like central tendency, they help you summarize a bunch of numbers with one or just a few numbers.

Importance & Limitation

In any research, enormous data is collected and, to describe it meaningfully, one needs to summarise the same. The bulkiness of the data can be reduced by organizing it into a frequency table or histogram. The following popper user interface control may not be accessible. Tab to the next button to revert the control to an accessible version.

Destroy user interface control Frequency distribution organizes the heap of data into a few meaningful categories. Collected data can also be summarized as a single index/value, which represents the entire data. These measures may also help in the comparison of data.

CENTRAL TENDENCY

Central tendency is defined as “the statistical measure that identifies a single value as representative of an entire distribution.” The following popper user interface control may not be accessible. Tab to the next button to revert the control to an accessible version.

Destroy user interface control It aims to provide an accurate description of the entire data. It is the single value that is most typical/ representative of the collected data. The term “number crunching” is used to illustrate this aspect of data description. The mean, median and mode are the three commonly used measures of central tendency.

MEAN

Mean is the most commonly used measure of central tendency. There are different types of mean, viz. arithmetic mean, weighted mean, geometric mean (GM) and harmonic mean (HM). If mentioned without an adjective (as mean), it generally refers to the arithmetic mean.

Arithmetic mean

Arithmetic mean (or, simply, “mean”) is nothing but the average. It is computed by adding all the values in the data set divided by the number of observations in it. If we have the raw data, mean is given by the formula

Mean $$\bar{X} = \frac{\Sigma X}{n}$$

Where, ∑ (the uppercase Greek letter sigma), *X* refers to summation, refers to the individual value and n is the number of observations in the sample (sample size). The research articles published in journals do not provide raw data and, in such a situation, the readers can compute the mean by calculating it from the frequency distribution (if provided).

$$\text{Mean} \quad \bar{X} = \frac{\Sigma f X}{n}$$

Where, *f* is the frequency and *X* is the midpoint of the class interval and n is the number of observations.[

The following popper user interface control may not be accessible. Tab to the next button to revert the control to an accessible version.

<u>**Destroy user interface control**</u> The standard statistical notations (in relation to measures of central tendency) are mentioned. Readers are cautioned that the mean calculated from the frequency distribution is not exactly the same as that calculated from the raw data. It approaches the mean calculated from the raw data as the number of intervals increase.[

The following popper user interface control may not be accessible. Tab to the next button to revert the control to an accessible version.

<u>**Destroy user interface control**</u>

Standard statistical notation

ADVANTAGES

The mean uses every value in the data and hence is a good representative of the data. The irony in this is that most of the times this value never appears in the raw data. Repeated samples drawn from the same population tend to have similar means. The mean is therefore the measure of central tendency that best resists the fluctuation between different samples. The following popper user interface control may not be accessible. Tab to the next button to revert the control to an accessible version.

Destroy user interface control

It is closely related to standard deviation, the most common measure of dispersion.

DISADVANTAGES

The important disadvantage of mean is that it is sensitive to extreme values/outliers, especially when the sample size is small. The following popper user interface control may not be accessible. Tab to the next button to revert the control to an accessible version.

Destroy user interface control Therefore, it is not an appropriate measure of central tendency for skewed distribution. The following popper user interface control may not be accessible. Tab to the next button to revert the control to an accessible version.

Destroy user interface control

Mean cannot be calculated for nominal or non nominal ordinal data. Even though mean can be calculated for numerical ordinal data, many times it does not give a meaningful value, e.g. stage of cancer.

Weighted mean

Weighted mean is calculated when certain values in a data set are more important than the others. The following popper user interface control may not be accessible. Tab to the next button to revert the control to an accessible version.

Destroy user interface control A weight w_i is attached to each of the values x_i to reflect this importance.

$$\text{Weighted mean} = \frac{\Sigma wx}{\Sigma w}$$

For example, When weighted mean is used to represent the average duration of stay by a patient in a hospital, the total number of cases presenting to each ward is taken as the weight.

Geometric Mean

It is defined as the arithmetic mean of the values taken on a log scale. It is also expressed as the nth root of the product of an observation.

$$\text{Geometric mean (GM)} = \sqrt[n]{(x_1)(x_2)\ldots(x_n)}$$

$$\text{Log (GM)} = \frac{\Sigma(\log x)}{n}$$

GM is an appropriate measure when values change exponentially and in case of skewed distribution that can be made symmetrical by a log transformation. GM is more commonly used in microbiological and serological research. One important disadvantage of GM is that it cannot be used if any of the values are zero or negative.

Harmonic mean

It is the reciprocal of the arithmetic mean of the observations

<u>Measures of dispersion</u>

The measure of central tendency, as discussed in the previous chapter tells us only about the characteristics of a particular series. They do not describe anything on the observations or data entirely. In other words, measures of central tendency do not tell anything about the variations that exist in the data of a particular series. To make the concept, let discuss an example. It was found by using formula of mean that the average depth of a river is 6 feet. One cannot confidently enter into the river because in some places the depth may be 12 feet or it may have 3 feet. Thus this type of interpretation by using the measures of central tendency sometimes proves to be useless. Hence the measure of central tendency alone to measure the characteristics of a series of observations is not sufficient to draw a valid conclusion. With the central value one must know as to how the data is distributed. Different sets of data may have the same measures of central tendency but differ greatly in terms of variation. For this knowledge of central value is not enough to appreciate the nature of distribution of values. Thus there is the requirement of some additional measures along with the measures of central tendency which will describe the spread of the entire set of values along with the central value. One such measure is popularly called as dispersion or variation. The study of dispersion will enables us to know whether a series is homogeneous (where all the observations remains around the central value) or the observations is heterogeneous (there will be variations in the observations around the central value like 1, 50, 20, 28 etc., where the central value is 33). Hence it can be said that a measure of dispersion describes the spread or scattering of the individual values of a series around its central value.

Experts opine different opinion on why the variations in a distribution are so important to consider? Following are some views on validity of the measure of dispersion:

1. Measures of variation provide the researchers some additional information about the behavior of the series along with the measures of central tendency. With this information one can judge the reliability of the value that is derived by using the measure of central tendency. If the data of the series are widely dispersed, the central location is less representatives of the data as a whole. On the other hand, when the data of a series is less dispersed, the central location is more representative to the entire series. In other wards, a high degree of variation would mean little uniformity whereas a low degree of variation would mean greater uniformity.
2. When the data of a series are widely dispersed, it creates practical problems in executing data. Measure of dispersion helps in understanding and tackling the widely dispersed data.
3. It facilitates to determine the nature and cause of variation in order to control the variation itself.
4. Measures of variation enable comparison to be made of two or more series with regard to their variability.

DEFINATION:

Following are some definitions defined by different experts on measures of dispersion. L.R. Connor defines measures of dispersion as �dispersion is the measure extended to which individual items vary�. Similarly, Brookes and Dick opines it as �dispersion or spread is the degree of the scatter or the variation of the variables about a central value�. Robert H. Wessel defines it as �measures which indicate the spread of the values are called measures of dispersion�. From all these definition it is clear that dispersion measures more or less describes the spread or scattering of the individual values of a series around its central value.

METHODS OF MEASURING DISPERSION:

Dispersion of a series of data can be calculated by using following four widely used methods . Dispersion when measured on basis of the difference between two extreme values selected from a series of data. The two well known measures are

1. The Range
2. The Inter-quartile Range or Quartile Deviation

Dispersion when measured on basis of average deviation from some measure of central tendency. The well known measures are

1. The Mean/average deviation
2. The Standard Deviation and
3. The Coefficient of variation and
4. The Gini coefficient and the Lorenz curve

Collection of data and formation of frequency distribution In statistics, a frequency distribution is an arrangement of the values that one or more variables take in a sample. Each entry in the table contains the frequency or count of the occurrences of values within a particular group or interval, and in this way, the table summarizes the distribution of values in the sample.

Univariate frequency tables

Rank	Degree of agreement	Number
1	Strongly agree	20
2	Agree somewhat	30
3	Not sure	20
4	Disagree somewhat	15
5	Strongly disagree	15

A different tabulation scheme aggregates values into bins such that each bin encompasses a range of values. For example, the heights of the students in a class could be organized into the following frequency table.

Height range	Number of students	Cumulative number
25	25	
5.0–5.5 feet	35	60
5.5–6 feet	20	80
6.0–6.5 feet	20	100

A frequency distribution shows us a summarized grouping of data divided into mutually exclusive classes and the number of occurrences in a class. It is a way of showing unorganized data e.g. to show results of an election, income of people for a certain region, sales of a product within a certain period, student loan amounts of graduates, etc. Some of the graphs that can be used with frequency distributions are histograms, line graphs, bar charts and pie charts. Frequency distributions are used for both qualitative and quantitative data.

Joint frequency distributions

Bivariate joint frequency distributions are often presented as (two-way) contingency tables:

Two-way contingency table with marginal frequencies

	Dance	Sports	TV	Total
Men	2	10	8	20
Women	16	6	8	30
Total	18	16	16	50

The *total* row and *total* column report the marginal frequencies or marginal distribution, while the body of the table reports the joint frequencies.

Applications

Managing and operating on frequency tabulated data is much simpler than operation on raw data. There are simple algorithms to calculate median, mean, standard deviation etc. from these tables. Statistical hypothesis testing is founded on the assessment of differences and similarities between frequency distributions.

This assessment involves measures of central tendency or averages, such as the mean and median, and measures of variability or statistical dispersion, such as the standard deviation or variance. A frequency distribution is said to be skewed when its mean and median are different.

The kurtosis of a frequency distribution is the concentration of scores at the mean, or how peaked the distribution appears if depicted graphically—for example, in a histogram. If the distribution is more peaked than the normal distribution it is said to be leptokurtic; if less peaked it is said to be platykurtic. Letter frequency distributions are also used in frequency analysis to crack codes and are referred to the relative frequency of letters in different languages.

Graphic presentation of frequency distribution—graphics, bars, histogram, Diagrammatic

Graphical representation is done of the data available this being a very important step of statistical analysis. We will be discussing the organization of data. The word 'Data' is plural for 'datum'; datum means facts. Statistically the term is used for numerical facts such as measures of height, weight and scores on achievement and intelligence tests.

Tests, experiments and surveys in education and psychology provide us valuable data, mostly in the shape of numerical scores. For understanding data available and deriving meaning and useful conclusion, the data have to be organized or arranged in some systematic way. This can be done by following ways:

1. Statistical tables

2. Rank order

3. Frequency distribution

Statistical tables

The data are tabulated or arranged into rows and columns of different heading. Such tables can list original raw scores as well as the percentages, means, standard deviations and so on. Example -

Table for group mean and S.D. of anxiety test of dancers and non dancers

Group	Mean	Standard deviation	N
Dancers	22.66	6.018	15
Non dancers	27.66	8.741	15

Rules for constructing tables:

1. Title of the table should be simple, concise and unambiguous. As a rule, it should appear on the table.

2. The table should be suitably divided into columns and rows according to the nature of data and purpose. These columns and rows should be arranged in a logical order to facilitate comparison.

3. The heading of each columns or row should be as brief as possible. Two or more columns or rows with similar headings may be grouped under a common heading to avoid repetition and we may have subheadings or captions.

4. Subtotal for each separate classification and a general total for all combined classes are to be given. These totals should be given at the bottom or right of the concerned items.

5. The units in which the data are given must invariably be mentioned.

6. Necessary footnotes should be providing essential explanation of the points to ambiguous representation of the tabulated data must be given at the bottom of the table.

7. The sources from where the data have been received should be given at the end of the table.

8. In tabulating long columns of figures, space should be left after every five or ten rows.

9. If the numbers tabulated have more than three significant figure, the digit should be grouped in threes. For ex.- 4394756 as 4 394 756.

10. For all purposes and by all means, the table should be as simple as possible so that it may be studied by the readers with minimum possible strain and create a clear picture and interpretations of the data.

Frequency Distribution

The organization of the data according to rank order does not help us to summarize a series of raw scores. It also does not tell us the frequency of the raw scores. In frequency distribution we group the data into an arbitrarily chosen groups or classes. It is also seen that how many times a particular score or group of scores occurs in the given data. This is known as the frequency distribution of numerical data.

Construction of Frequency distribution table

Finding the range:

First of all the range of the series to be grouped is found. it is done by subtracting the lowest score from the highest. In the present problem the range of the distribution is 46-12, i.e. 34

Determining class interval:

After finding range we find class interval represented by '*i*'. The formula for I is

i = Range/ no. of class interval desired

I = 34/8

I = 4.25

We decide to take class interval to be 5.

Writing the contents of the frequency distribution table:

Writing the classes of the distribution.

In the first column we write the classes of distribution. First of all the lowest class is settled and afterwards other subsequent classes are written down. In this case we take 10-14 as the lowest class, then we have higher classes as 15-19, 20-24, and so on up to 45-49.

Tallying the scores into proper classes.

The scores given are tallied into proper classes in the second column then the tallies are counted against each class to obtain the frequency of the class. Example-

Class interval	Tallies	Frequency for Non-dancers
45-49	l	1
40-44	0	0
35-39	ll	2
30-34	lll	3
25-29	llll	4
20-24	ll	2
15-19	ll	2
10-14	l	1

Total frequency (N) = 15

GRAPHICAL REPRESENTATION OF DATA

The statistical data may be presented in a more attractive form appealing to the eye with the help of some graphic aids, i.e. Pictures and graphs. Such presentation carries a lot of communication power. A mere glimpse of thee picture and graphs may enable the viewer to have an immediate and meaningful grasp of the large amount of data.

Ungrouped data may be represented through a bar diagram, pie diagram, pictograph and line graph.

METHOD FOR CONSTRUCTING A HISTOGRAM

1. The scores in the form of actual class limits as 19.5-24.5, 24.5-29.5 and so on are taken as examples in the construction of a histogram rather than written class limits as 20-24, 25-30.

2. It is customary to take two extra intervals of classes one below and above the grouped intervals.

3. Now we take the actual lower limits of all the class intervals and try to plot them on the x axis. The lower limit of the lowest class interval is taken at the intersecting point of x axis and y axis.

4. Frequencies of the distribution are plotted on the y axis.

5. Each class interval with its specific frequency is represented by separate rectangle. The base of each rectangle is the width of the class interval. And the height is representative of the frequency of that class or interval.

6. Care should be taken to select the appropriate units of representation along the x and y axis. Both the axis and the y axis must not be too short or too long.

METHOD FOR CONSTRUTING A FREQUENCY POLYGON

1. As in histogram two extra class interval is taken, one above and other below the given class interval.

2. The mid-points of the class interval is calculated.

3. The midpoint is calculated along the x axis and the corresponding frequencies are plotted along the y axis.

4. The various points given by the plotting are joined by lines to give frequency polygon.

DIFFERENCE BETWEEN HISTOGRAM AND FREQUENCY POLYGON

Histogram is a bar graph while frequency polygon is a line graph. Frequency polygon is more useful and practical. In frequency polygon it is easy to know the trends of the distribution; we are unable to do so in histogram. Histogram gives a very clear and accurate picture of the relative proportion of the frequency from interval to interval.

METHOD FOR CONSTRUTING A CUMULATIVE FREQUENCY GRAPH

1. First of all we calculate the actual upper and lower limits of the class intervals i.e. if the class interval is 20-24 then upper limit is 24.5 and the lower limit is 19.5.

2. We must know select a suitable scale as per the range of the class interval and plot the actual upper limits on the x axis and the respective cumulative frequency on y axis.

3. All the plotted points are then joined by successive straight lines resulting a line graph.

4. To plot the origin of the x axis an extra class interval is taken with cumulative frequency zero is taken.

Measures of central tendency- mean, median and mode

A measure of central tendency is a single value that attempts to describe a set of data by identifying the central position within that set of data. As such, measures of central tendency are sometimes called measures of central location. They are also classed as summary statistics. The mean (often called the average) is most likely the measure of central tendency that you are most familiar with, but there are others, such as the median and the mode. The mean, median and mode are all valid measures of central tendency, but under different conditions, some measures of central tendency become more appropriate to use than others. In the following sections, we will look at the mean, mode and median, and learn how to calculate them and under what conditions they are most appropriate to be used.

Mean (Arithmetic)

The mean (or average) is the most popular and well known measure of central tendency. It can be used with both discrete and continuous data, although its use is most often with continuous data (see our Types of Variable guide for data types).

The mean is equal to the sum of all the values in the data set divided by the number of values in the data set. So, if we have n values in a data set and they have values x_1, x_2, ..., x_n, the sample mean, usually denoted by $\bar{x}$ (pronounced x bar), is:

$$\bar{x} = \frac{(x_1 + x_2 + \cdots + x_n)}{n}$$

This formula is usually written in a slightly different manner using the Greek capitol letter, $\sum$, pronounced "sigma", which means "sum of...":

$$\bar{x} = \frac{\sum x}{n}$$

You may have noticed that the above formula refers to the sample mean. So, why have we called it a sample mean? This is because, in statistics, samples and populations have very different meanings and these differences are very important, even if, in the case of the mean, they are calculated in the same way. To acknowledge that we are calculating the population mean and not the sample mean, we use the Greek lower case letter "mu", denoted as µ:

$$\mu = \frac{\sum x}{n}$$

The mean is essentially a model of your data set. It is the value that is most common. You will notice, however, that the mean is not often one of the actual values that you have observed in your data set. However, one of its important properties is that it minimizes error in the prediction of any one value in your data set. That is, it is the value that produces the lowest amount of error from all other values in the data set. An important property of the mean is that it includes every value in your data set as part of the calculation. In addition, the mean is the only measure of central tendency where the sum of the deviations of each value from the mean is always zero.

When not to use the mean

The mean has one main disadvantage: it is particularly susceptible to the influence of outliers. These are values that are unusual compared to the rest of the data set by being especially small or large in numerical value. For example, consider the wages of staff at a factory below:

Staff	1	2	3	4	5	6	7	8	9	10
Salary	15k	18k	16k	14k	15k	15k	12k	17k	90k	95k

The mean salary for these ten staff is $30.7k. However, inspecting the raw data suggests that this mean value might not be the best way to accurately reflect the typical salary of a worker, as most workers have salaries in the $12k to 18k range. The mean is being skewed by the two large salaries. Therefore, in this situation, we would like to have a better measure of central tendency. As we will find out later, taking the median would be a better measure of central tendency in this situation.

Another time when we usually prefer the median over the mean (or mode) is when our data is skewed (i.e., the frequency distribution for our data is skewed). If we consider the normal distribution - as this is the most frequently assessed in statistics - when the data is perfectly normal, the mean, median and mode are identical. Moreover, they all represent the most typical value in the data set. However, as the data becomes skewed the mean loses its ability to provide the best central location for the data because the skewed data is dragging it away from the typical value. However, the median best retains this position and is not as strongly influenced by the skewed values. This is explained in more detail in the skewed distribution section later in this guide.

Median

The median is the middle score for a set of data that has been arranged in order of magnitude. The median is less affected by outliers and skewed data. In order to calculate the median, suppose we have the data below:

65	55	89	56	35	14	56	55	87	45	92

We first need to rearrange that data into order of magnitude (smallest first):

14	35	45	55	55	56	56	65	87	89	92

Our median mark is the middle mark - in this case, 56 (highlighted in bold). It is the middle mark because there are 5 scores before it and 5 scores after it. This works fine when you have an odd number of scores, but what happens when you have an even number of scores? What if you had only 10 scores? Well, you simply have to take the middle two scores and average the result. So, if we look at the example below:

65	55	89	56	35	14	56	55	87	45

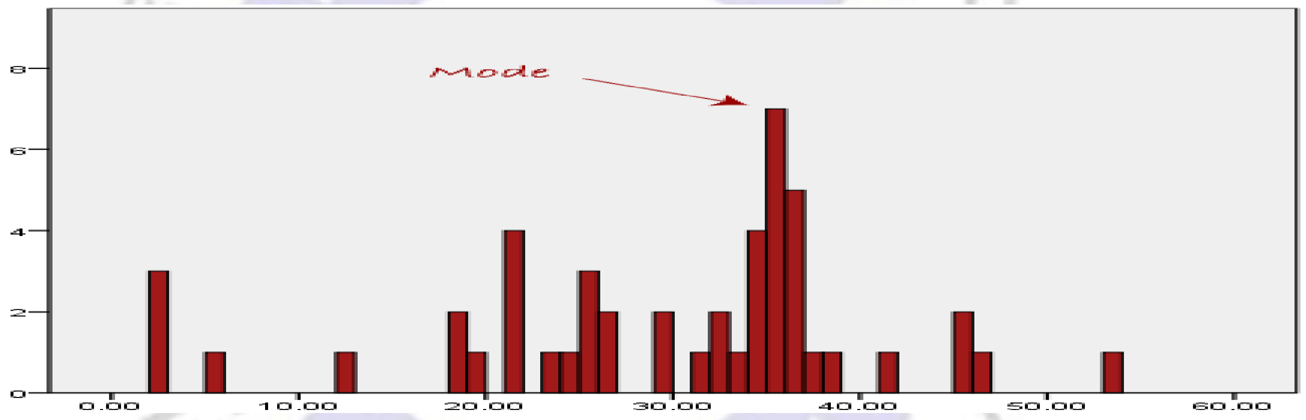
We again rearrange that data into order of magnitude (smallest first):

14	35	45	55	55	56	56	65	87	89	92

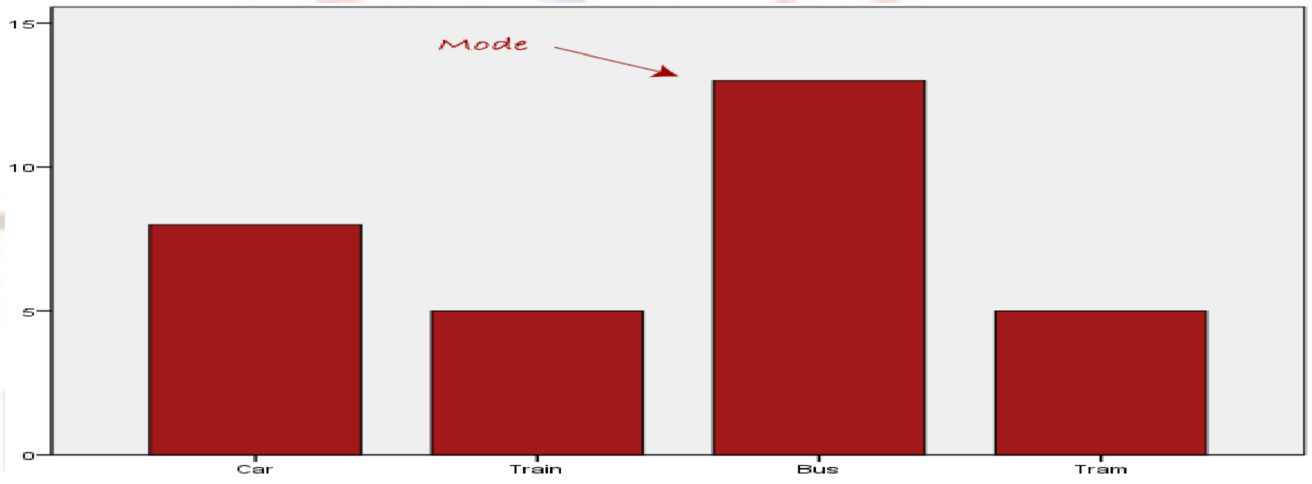
Only now we have to take the 5th and 6th score in our data set and average them to get a median of 55.5.

Mode

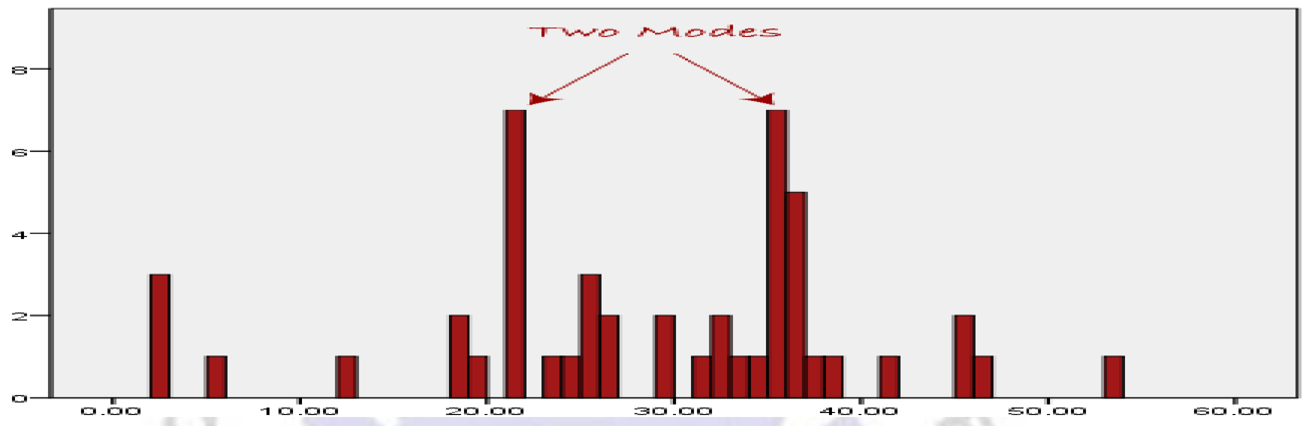
The mode is the most frequent score in our data set. On a histogram it represents the highest bar in a bar chart or histogram. You can, therefore, sometimes consider the mode as being the most popular option. An example of a mode is presented below:



Normally, the mode is used for categorical data where we wish to know which is the most common category, as illustrated below:

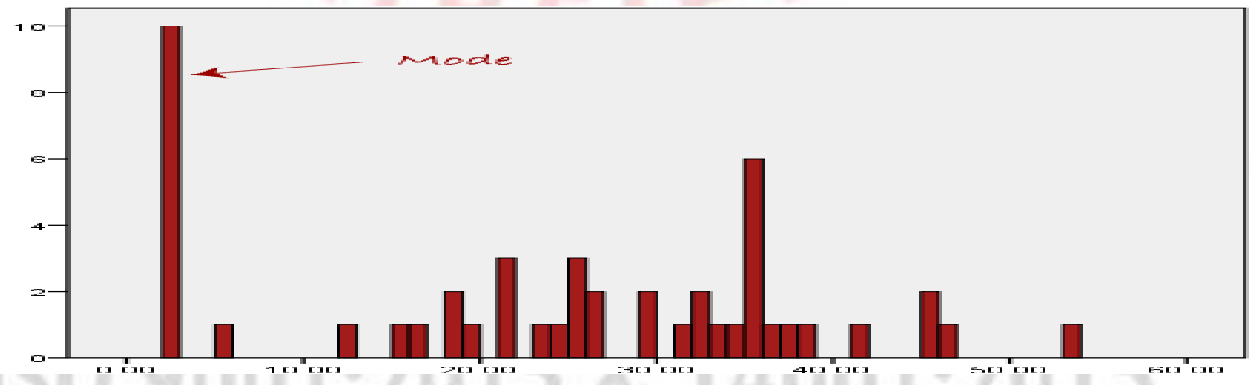


We can see above that the most common form of transport, in this particular data set, is the bus. However, one of the problems with the mode is that it is not unique, so it leaves us with problems when we have two or more values that share the highest frequency, such as below:



We are now stuck as to which mode best describes the central tendency of the data. This is particularly problematic when we have continuous data because we are more likely not to have any one value that is more frequent than the other. For example, consider measuring 30 peoples' weight (to the nearest 0.1 kg). How likely is it that we will find two or more people with exactly the same weight (e.g., 67.4 kg)? The answer, is probably very unlikely - many people might be close, but with such a small sample (30 people) and a large range of possible weights, you are unlikely to find two people with exactly the same weight; that is, to the nearest 0.1 kg. This is why the mode is very rarely used with continuous data.

Another problem with the mode is that it will not provide us with a very good measure of central tendency when the most common mark is far away from the rest of the data in the data set, as depicted in the diagram below:



In the above diagram the mode has a value of 2. We can clearly see, however, that the mode is not representative of the data, which is mostly concentrated around the 20 to 30 value range. To use the mode to describe the central tendency of this data set would be misleading.

Partition values-quartiles, deciles and percentiles & Measures of variation- range, IQR, quartile, deciles and percentiles

The computation of these partition values is done exactly in the same manner as the computation of the Median. In a series of individual observations and in a discrete series, the values of the lower (Q_1) and the upper (Q_9) quartiles would be the value of (N+1) /4) th and (3(N+1) / 4 items respectively.

The values of the deciles in such series would be as follows.

$$D_1 = \text{value of } \left(\frac{N+1}{10}\right)^{th} \text{ item}$$

$$D_2 = \text{value of } \frac{2(N+1)^{th}}{10} \text{ item}$$

$$D_8 = \text{value of } \frac{8\,(N+1)^{th}}{10} \text{ item and so on.}$$

$$P_1 = \text{value of} \left(\frac{N+1}{100}\right)^{th} \text{ item}$$

$$P_{40} = \text{value of } \frac{40\,(N+1)^{th}}{100} \text{ item}$$

$$P_{99} = \text{value of } \frac{99\,(N+1)^{th}}{100} \text{ item and so on.}$$

The values of the percentiles would be In continuous series in the calculation of quartiles, deciles and percentiles (N+1)/4 , (N+1)/10 and (N+1)/100 would be replaced by N/4 ,N/10 and N/100 respectively. The values would have to be interpolated here as was done in case of the computation of Median.

UNIT-II (Correlation & Regression)

Correlation Coefficient

Correlation coefficient may refer to:

- Pearson product-moment correlation coefficient, also known as *r*, *R*, or Pearson's *r*, a measure of the strength and direction of the linear relationship between two variables that is defined as the (sample) covariance of the variables divided by the product of their (sample) standard deviations.

- Infraclass correlation, a descriptive statistic that can be used when quantitative measurements are made on units that are organized into groups; describes how strongly units in the same group resemble each other.
- Rank correlation, the study of relationships between rankings of different variables or different rankings of the same variable
    - Spearman's rank correlation coefficient, a measure of how well the relationship between two variables can be described by a monotonic function
    - Kendall tau rank correlation coefficient, a measure of the portion of ranks that match between two data sets.
    - Goodman and Kruskal's gamma, a measure of the strength of association of the cross tabulated data when both variables are measured at the ordinal level.

Assumptions of correlation analysis

An inspection of a scatter plot can give an impression of whether two variables are related and the direction of their relationship. But it alone is not sufficient to determine whether there is an association between two variables. The relationship depicted in the scatter plot needs to be described qualitatively. Descriptive statistics that express the degree of relation between two variables are called correlation coefficients. A commonly employed correlation coefficient for scores at the interval or ratio level of measurement is the Pearson product-moment correlation coefficient, or Pearson’s r.

The Pearson's r is a descriptive statistic that describes the linear relationship between two or more variables, each measured for the same collection of individuals. An "individual" is not necessarily a person: it might be an automobile, a place, a family, a university, etc. For example, the two variables might be the heights of a man and of his son; there, the "individual" is the pair (father, son). Such pairs of measurements are called bivariate data. Observations of two or more variables per individual in general are called multivariate data. As with any sample of scores, the sample is drawn from a larger population of scores. The test for significance of Pearson's r assumes that a particular variable, X and another variable, Y, form a bivariate normal distribution in the population. A bivariate normal distribution possesses the following characteristics:

- The distribution of the X scores is normally distributed in the population sampled.
- The distribution of the Y scores is normally distributed in the population sampled.
- For each X score, the distribution of Y scores in the population is normal.
- For each Y score, the distribution of Y scores in the population is normal.

Coefficients of determination & correlation

In statistics, the coefficient of determination denoted R^2 and pronounced R squared, indicates how well data points fit a line or curve. It is a statistic used in the context of statistical models whose main purpose is either the prediction of future outcomes or the testing of hypotheses, on the basis of other related information. It provides a measure of how well observed outcomes are replicated by the model, as the proportion of total variation of outcomes explained by the model. There are several different definitions of R^2 which are only sometimes equivalent. One class of such cases includes that of linear regression. In this case, if an intercept is included then R^2 is simply the square of the sample correlation coefficient between the outcomes and their predicted values; in the case of simple linear regression, it is the squared correlation between the outcomes and the values of the single repressor being used for prediction. If an intercept is included and the number of explicators is more than one, R^2 is the square of the coefficient of multiple correlation. In such cases, the coefficient of determination ranges from 0 to 1. Important cases where the computational definition of R^2 can yield negative values, depending on the definition used, arise where the predictions which are being compared to the corresponding outcomes have not been derived from a model-fitting procedure using those data, and where linear regression is conducted without including an intercept. Additionally, negative values of R^2 may occur when fitting non-linear functions to data. In cases where negative values arise, the mean of the data provides a better fit to the outcomes than do the fitted function values, according to this particular criterion.

Measurement of correlation- Karl Pearson's Methods

In statistics, the Pearson product-moment correlation coefficient (r) is a common measure of the correlation between two variables X and Y. When measured in a population the Pearson Product Moment correlation is designated by the Greek letter rho (?). When computed in a sample, it is designated by the letter "r" and is sometimes called "Pearson's r." Pearson's correlation reflects the degree of linear relationship between two variables. It ranges from +1 to -1. A correlation of +1 means that there is a perfect positive linear relationship between variables. A correlation of -1 means that there is a perfect negative linear relationship between variables.

A correlation of 0 means there is no linear relationship between the two variables. Correlations are rarely if ever 0, 1, or -1. If you get a certain outcome it could indicate whether correlations were negative or positive.

Mathematical Formula:-- : The quantity r, called the linear correlation coefficient, measures the strength and the direction of a linear relationship between two variables. The linear correlation coefficient is sometimes referred to as the Pearson product moment correlation coefficient in honor of its developer Karl Pearson.

The mathematical formula for computing r is:

$$r = \frac{N\Sigma xy - (\Sigma x)(\Sigma y)}{\sqrt{[N\Sigma x^2 - (\Sigma x)^2][N\Sigma y^2 - (\Sigma y)^2]}}$$

Where:

N	=	number of pairs of scores
Σxy	=	sum of the products of paired scores
Σx	=	sum of x scores
Σy	=	sum of y scores
Σx^2	=	sum of squared x scores
Σy^2	=	sum of squared y scores

Spearman’s rank correlation

There are two methods to calculate Spearman's rank-order correlation depending on whether: (1) your data does not have tied ranks or (2) your data has tied ranks. The formula for when there are no tied ranks is:

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

where d_i = difference in paired ranks and *n* = number of cases. The formula to use when there are tied ranks is:

$$\rho = \frac{\sum_i (x_i - \bar{x})(y_i - \bar{y})}{\sqrt{\sum_i (x_i - \bar{x})^2 \sum_i (y_i - \bar{y})^2}}$$

Where *i* = paired score.

Concurrent deviation the correlation coefficient

Sometimes it is desired to study the correlation between two series in a very casual manner and in such cases no particular attention is needed so far as precision is concerned. In such cases it is enough to calculate the coefficient of concurrent deviations. I n this method correlation is calculated between the direction of deviations and not their magnitudes. As such only the direction of deviations is taken into account in the calculation of this coefficient, and their magnitude is ignored.

To calculate the coefficient of concurrent deviations, the deviations are not calculated from any average or by the method of moving averages but only their direction from the previous period, are noted down. The formula for the calculation of coefficient of concurrent deviations is given below:—

Coefficient of concurrent deviations or

rc = ±√(± ((2c-n)/n)

Where rc = stands for the coefficient of concurrent deviations, for the number of pairs of concurrent deviations and n for the number of pairs of deviations. The value of this coefficient of correlation also varies between 1. The plus and minus signs given in the formula should be Carefully noted. If the value of (2c-n)/n) is negative its square root±± 2c-nn

Where cannot be calculated and so a minus sign is placed before the sign of the root so that the square root may be calculated and the minus sign may be kept before the value of the coefficient of correlation.Thus the steps in the calculation of this coefficient are:

(1) Find out the direction of change of x—variable. It means find out whether the second figure in the series is more than the first figure, if it is so the direction is. If the second figure is less than the first the direction is —. If both the figures are equal the direction is 0. Similarly find the direction of the third figure from the second, of the fourth figure from the third and so on. It is denoted by (dx).

(2) Find out the direction of change of y-variable in the same manner as discussed above. It is denoted by (dy).

(3) multiply dx and dy and determine the value of c , which would be the number of positive products of dxdy (—x—) or (X+).

(4) Use the formula given above to obtain the value of the Coefficient or r.

Pitfalls and limitations associated with regression & correlation analysis: real world application using IT tools

We are all familiar with the disparaging quotes about statistics (including "There are three kinds of lies: lies, damned lies, and statistics", attributed to either Mark Twain or Disraeli, depending on whom you ask), and it's no secret that many people harbor a vague distrust of statistics as commonly used. Why should this be the case? It may be assumed that those of us at this conference take our work seriously and value the fruits of our efforts. So, are all those people just paranoid about statistics, or are we as statisticians really kidding ourselves as to our importance in the cosmic scheme of things?

It may be helpful to consider some aspects of statistical thought which might lead many people to be distrustful of it. First of all, statistics requires the ability to consider things from a probabilistic perspective, employing quantitative technical concepts such as "confidence", "reliability", "significance". This is in contrast to the way non-mathematicians often cast problems: logical, concrete, often dichotomous conceptualizations are the norm: right or wrong, large or small, this or that.

Additionally, many non-mathematicians hold quantitative data in a sort of awe. They have been lead to believe that numbers are, or at least should be, unquestionably correct. Consider the sort of math problems people are exposed to in secondary school, and even in introductory college math courses: there is a clearly defined method for finding the answer, and that answer is the only acceptable one. It comes, then, as a shock that different research studies can produce very different, often contradictory results. If the statistical methods used are really supposed to represent reality, how can it be that different studies produce different results? In order to resolve this paradox, many naive observers conclude that statistics must not really provide reliable (in the nontechnical sense) indicators of reality after all. And, the logic goes, if statistics aren't "right", they must be "wrong". It is easy to see how even intelligent, well-educated people can become cynical if they don't understand the subtleties of statistical reasoning and analysis. Now, I'm not going to say much about this "public relations crisis" directly, but it does provide a motivation for examining the way we practice our trade. The best thing we can do, in the long run, is make sure we're using our tools properly, and that our conclusions are warranted. I will present some of the most frequent misuses and abuses of statistical methods, and how to avoid or remedy them. Of course, these issues will be familiar to most statisticians; however, they are the sorts of things that can get easily overlooked when the pressure is on to produce results and meet deadlines. If this workshop helps you to apply the basics of statistical reasoning to improve the quality of your product, it will have served its purpose. We can consider three broad classes of statistical pitfalls. The first involves sources of bias. These are conditions or circumstances which affect the external validity of statistical results. The second category is errors in methodology, which can lead to inaccurate or invalid results. The third class of problems concerns interpretation of results, or how statistical results are applied (or misapplied) to real world issues.

Sources of Bias

The core value of statistical methodology is its ability to assist one in making inferences about a large group (a population) based on observations of a smaller subset of that group (a sample). In order for this to work correctly, a couple of things have to be true: the sample must be similar to the target population in all relevant aspects; and certain aspects of the measured variables must conform to assumptions which underlie the statistical procedures to be applied.

Representative sampling

This is one of the most fundamental tenets of inferential statistics: the observed sample must be representative of the target population in order for inferences to be valid. Of course, the problem comes in applying this principle to real situations. The ideal scenario would be where the sample is chosen by selecting members of the population at random, with each member having an equal probability of being selected for the sample. Barring this, one usually tries to be sure that the sample "parallels" the population with respect to certain key characteristics which are thought to be important to the investigation at hand, as with a stratified sampling procedure.

While this may be feasible for certain manufacturing processes, it is much more problematic for studying people. For instance, consider the construction of a job applicant screening instrument: the population about which you want to know something is the pool of all possible job applicants. You surely won't have access to the entire population--you only have access to a certain number of applicants who apply within a certain period of time. So you must hope that the group you happen to pick isn't somehow different from the target population. An example of a problematic sample would be if the instrument were developed during an economic recession; it is reasonable to assume that people applying for jobs during a recession might be different as a group from those applying during a period of economic growth (even if one can't specify exactly what those differences might be). In this case, you'd want to exercise caution when using the instrument during better economic times. There are also ways to account for, or "control", differences between groups statistically, as with the inclusion of covariates in a linear model. Unfortunately, as Levin (1985) points out, there are problems with this approach, too. One can never be sure one has accounted for all the important variables, and inclusion of such controls depends on certain assumptions which may or may not be satisfied in a given situation (see below for more on assumptions).

Statistical assumptions

The validity of a statistical procedure depends on certain assumptions it makes about various aspects of the problem. For instance, a well-known linear method such as analysis of variance (ANOVA) depends on the assumption of normality and independence. The first of these is probably the lesser concern, since there is evidence that the most common ANOVA designs are relatively insensitive to moderate violations of the normality assumption (see Kirk, 1982). Unfortunately, this offers an almost irresistible temptation to ignore *any* non-normality, no matter how bad the situation is. The robustness of statistical techniques only goes so far--"robustness" is not a license to ignore the assumption. If the distributions are non-normal, try to figure out why; if it's due to a measurement artifact (e.g. a floor or ceiling effect), try to develop a better measurement device (if possible). Another possible method for dealing with unusual distributions is to apply a transformation. However, this has dangers as well; an ill-considered transformation can do more harm than good in terms of interpretability of results.

The assumption regarding independence of observations is more troublesome, both because it underlies nearly all of the most commonly used statistical procedures, and because it is so frequently violated in practice. Observations which are linked in some way--parts manufactured on the same machine, students in the same classroom, consumers at the same mall--all may show some dependencies. Therefore, if you apply some statistical test across students in different classrooms, say to assess the relationship between different textbook types and test scores, you're introducing bias into your results. This occurs because, in our example, the kids in the class presumably interact with each other, chat, talk about the new books they're using, and so influence each other's responses to the test. This will cause the results of your statistical test (e.g. correlations or p-values) to be inaccurate.

One way to try to get around this is to aggregate cases to the higher level, e.g. use classrooms as the unit of analysis, rather than students. Unfortunately this requires sacrificing a lot of statistical power, making a Type II error more likely. Happily, methods have been developed recently which allow simultaneous modeling of data which is hierarchically organized (as in our example with students nested within classrooms). One of the papers presented at this conference (Christiansen & Morris) introduces these methods. Additionally, interested readers are referred for good overviews of these hierarchical models.

Errors in methodology

There are a number of ways that statistical techniques can be misapplied to problems in the real world. Three of the most common hazards are designing experiments with insufficient power, ignoring measurement error, and performing multiple comparisons.

Statistical Power

This topic has become quite in vogue lately, at least in the academic community; indeed, some federal funding agencies seem to consider any research proposal incomplete unless it contains a comprehensive power analysis. This graph will help illustrate the concept of power in an experiment. In the figure, the vertical dotted line represents the point-null hypothesis, and the solid vertical line represents a criterion of significance, i.e. the point at which you claim a difference is significant.

Unit-III (Linear Programming & Queuing)

Concept a assumptions usage in business decision making linear programming problem

If you've only got limited resources at your disposal, then it's helpful to calculate how best to maximize those resources - whether that's time, money, or space. Let's say, for example, that you have 50 square feet of office space to use for storage. Your budget is $200, and there are a variety of cabinet types and sizes from which to choose. How do you optimize the space you have available, and stay within the allotted budget? Or suppose you have three delivery trucks, and 10 drop-off points. How do you plan the most efficient route and schedule for these trucks?

Or consider that you manufacture three products using the same basic raw materials. However, as each product uses different amounts of material, some are more expensive to produce than others. A few of the materials are perishable, and need to be used quickly. How much of each product should you manufacture to minimize your cost? And which combination produces the least waste? Questions like these may seem very complex. With so many variables and constraints to take into consideration, how do you decide what to do? The answer is to use linear programming.

Linear programming is a mathematical technique that determines the best way to use available resources. Managers use the process to help make decisions about the most efficient use of limited resources - like money, time, materials, and machinery.

Methods of solving graphical & simplex

Graphical and Simplex Methods of Linear Programming, The graphical method is the more popular method to use because they are easy to use and understand. Working with only a few variables at a time they allow operations managers to compare projected demand to existing capacity. The graphical method is a trial and error approach that can be easily done by a manager or even a clerical staff. Since it is trial and error though, it does not necessarily generate the optimal plan. One downside of this method though is that it can only be used with two variables at the maximum. The graphical method is broken down into the following five steps:

1) Determine the demand in each period.
2) Determine the capacity for regular time, over time, and subcontracting each period.
3) Find labor costs, hiring and labor costs, and inventory holding costs.
4) Consider company policy that may apply to the workers or to stock levels
5) Develop alternative plans and examine their total costs.

When a company has a LP problem with more than two variables it turns to the simplex method. This method can handle any number of variables as well as for certain give the optimal solution. In the simplex method we examine corner points in a methodical fashion until we arrive at the best solution which is either the highest profit or lowest cost.
LP is used in a wide variety of companies in numerous applications. Airline companies use it to schedule their flights to maximize profit. Another use is for firms to figure out how much of a certain product to manufacture in order to maximize total profits. It also is used by hospitals in order to figure out the most economic diet for patients. It is also a useful tool to figure out labor scheduling for a specific time period. Other applications include product mix planning, distribution networks, truck routing, financial portfolios, and corporate restructuring.
All LP problems have four properties in common.

Duality: concept, significance, usage & application in business decision making

The term *structure* referred generally to "rules and resources" and more specifically to "the structuring properties allowing the 'binding' of time-space in social systems". These properties make it possible for similar social practices to exist across time and space and that lend them "systemic" form. Agents—groups or individuals—draw upon these structures to perform social actions through embedded memory, called *memory traces*.

Memory traces are thus the vehicle through which social actions are carried out. Structure is also, however, the result of these social practices. Thus, Giddens conceives of the *duality of structure* as being:

The essential recursiveness of social life, as constituted in social practices: structure is both medium and outcome of reproduction of practices. Structure enters simultaneously into the constitution of the agent and social practices, and 'exists' in the generating moments of this constitution.

Giddens uses "the duality of structure" to emphasize structure's nature as both medium and outcome. Structures exist both internally within agents as memory traces that are the product of phenomenological and hermeneutic inheritance and externally as the manifestation of social actions. Similarly, social structures contain agents and/or are the product of past actions of agents. Giddens holds this duality, alongside "structure" and "system," as the core of structuration theory. His theory has been adopted by those with structuralism inclinations, but who wish to situate such structures in human practice rather than to reify them as an ideal type or material property. (This is different, for example, from actor–network theory which appears to grant a certain autonomy to technical artifacts.) Social systems have patterns of social relation that change over time; the changing nature of space and time determines the interaction of social relations and therefore structure. Hitherto, social structures or models were either taken to be beyond the realm of human control—the positivistic approach—or posit that action creates them—the interpretive approach. The duality of structure emphasizes that they are different sides to the same central question of how social order is created. Gregory McLennan suggested renaming this process "the duality of structure *and agency*", since both aspects are involved in using and producing social actions

Change

Sewell provided a useful summary that included one of the theory's less specified aspects: the question "Why are structural transformations possible?" He claimed that Giddens' overruled on rules and modified Giddens' argument by re-defining "resources" as the embodiment of cultural schemas. He argued that change arises from the multiplicity of structures, the *transposable* nature of schemas, the unpredictability of resource accumulation, the polysemy of resources and the intersection of structures.

The existence of multiple structures implies that the knowledgeable agents whose actions produce systems are capable of applying different schemas to contexts with differing resources, contrary to the conception of a universal habitués (learned dispositions, skills and ways of acting). He wrote that "Societies are based on practices that derived from many distinct structures, which exist at different levels, operate in different modalities, and are themselves based on widely varying types and quantities of resources. ...It is never true that all of them are homologous." Originally from Bourdieu, *transposable* schemas can be "applied to a wide and not fully predictable range of cases outside the context in which they were initially learned." That capacity "is inherent in the knowledge of cultural schemas that characterizes all minimally competent members of society."

Agents may modify schemas even though their use does not predictably accumulate resources. For example, the effect of a joke is never quite certain, but a comedian may alter it based on the amount of laughter it garners regardless of this variability. Agents may interpret a particular resource according to different schemas. E.g., a commander could attribute his wealth to military prowess, while others could see it as a blessing from the gods or a coincidental initial advantage. Structures often overlap, confusing interpretation (e.g., the structure of capitalist society includes production from both private property and worker solidarity).

Technology

This theory was adapted and augmented by researchers interested in the relationship between technology and social structures (see Theories of technology), such as information technology in organizations. DeSanctis and Poole proposed an "adaptive structuration theory" with respect to the emergence and use of group decision support systems. In particular, they chose Giddens' notion of modalities to consider how technology is used with respect to its "spirit". "Appropriations" are the immediate, visible actions that reveal deeper structuration processes and are enacted with "moves". Appropriations may be fait Wanda Orlikowski applied her critique of the duality of structure to hful or unfaithful, be instrumental and be used with various attitudes. technology: "The duality of technology identifies prior views of technology as either objective force or as socially constructed product–as a false dichotomy." She compared this to previous models (the technological imperative, strategic choice, and technology as a trigger) and considered the importance of meaning, power, norms, and interpretive flexibility. Orlikowski later replaced the notion of embedded properties for enactment (use). The "practice lens" shows how people enact structures which shape their use of technology that they employ in their practices.

While Orlikowski's work focused on corporations, it is equally applicable to the technology cultures that have emerged in smaller community-based organizations, and can be adapted through the *gender sensitivity lens* in approaches to technology governance. Workman, Ford and Allen rearticulated structuration theory as *structuration agency theory* for modeling socio-biologically inspired structuration in security software. Software agents join humans to engage in social actions of information exchange, giving and receiving instructions, responding to other agents, and pursuing goals individually or jointly.

Business

Pavlou and Majchrzak argued that research on business-to-business e-commerce portrayed technology as overly deterministic. The authors employed structuration theory to re-examine outcomes such as economic/business success as well as trust, coordination, innovation, and shared knowledge. They looked beyond technology into organizational structure and practices, and examined the effects on the structure of adapting to new technologies. The authors held that technology needs to be aligned and compatible with the existing "trustworthy" practices and organizational and market structure. The authors recommended measuring long-term adaptations using ethnography, monitoring and other methods to observe causal relationships and generate better predictions.

Group communication

Poole, Seibold, and McPhee wrote that "group structuration theory, provides "a theory of group interaction commensurate with the complexities of the phenomenon. The theory attempts to integrate macrosocial theories and individuals or small groups, as well as how to avoid the binary categorization of either "stable" or "emergent" groups. Waldeck et al. concluded that the theory needs to better predict outcomes, rather than merely explaining them. *Decision rules* support decision-making, which produces a communication pattern that can be directly observable. Research has not yet examined the "rational" function of group communication and decision-making (i.e., how well it achieves goals), nor structural production or constraints. Researchers must empirically demonstrate the recursivity of action and structure, examine how structures stabilize and change over time due to group communication, and may want to integrate argumentation research.

Public relations

Falkheimer claimed that integrating structuration theory into public relations (PR) strategies could result in a less agency-driven business, return theoretical focus to the role of power structures in PR, and reject massive PR campaigns in favor of a more "holistic understanding of how PR may be used in local contexts both as a reproductive and [transformational] social instrument." Falkheimer portrayed PR as a method of communication and action whereby social systems emerge and reproduce. Structuration theory reinvigorates the study of space and time in PR theory. Applied structuration theory may emphasize community-based approaches, storytelling, rituals, and informal communication systems. Moreover, structuration theory integrates all organizational members in PR actions, integrating PR into all organizational levels rather than a separate office.

Queuing Models: Basic structure of queuing models

Everyone has experienced waiting in line, whether at a fast-food restaurant, on the phone for technical help, at the doctor's office or in the drive-through lane of a bank. Sometimes, it is a pleasant experience, but many times it can be extremely frustrating for both the customer and the store manager. Given the intensity of competition today, a customer waiting too long in line is potentially a lost customer. Understanding the nature of lines or "queues" and learning how to manage them is one of the most important areas in operations management.

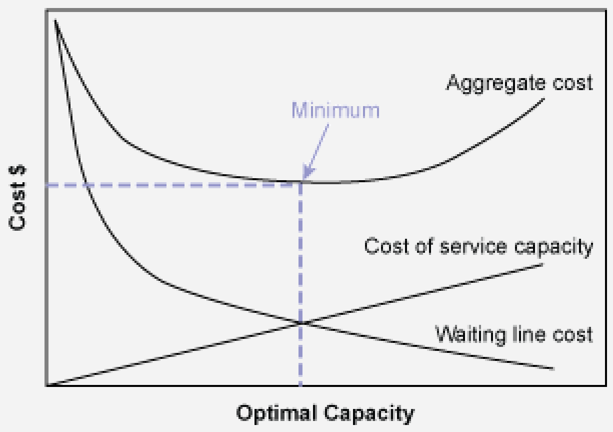
Queues are basic to both external (customer-facing) and internal business processes, which include staffing, scheduling and inventory levels. For this reason, businesses often utilize queuing theory as a competitive advantage. Fortunately, Six Sigma professionals – through their knowledge of probability distributions, process mapping and basic process improvement techniques – can help organizations design and implement robust queuing models to create this competitive advantage.

The Cost of Waiting in Line

The problem in virtually every queuing situation is a trade-off decision. The manager must weigh the added cost of providing more rapid service (i.e., more checkout counters, more production staff) against the inherent cost of waiting. For example, if employees are spending their time manually entering data, a business manager or process improvement expert could compare the cost of investing in bar-code scanners against the benefits of increased productivity. Likewise, if customers are walking away disgusted because of insufficient customer support personnel, the business could compare the cost of hiring more staff to the value of increased revenues and maintaining customer loyalty.

The relationship between service capacity and queuing cost can be expressed graphically (Figure 1). Initially, the cost of waiting in line is at a maximum when the organization is at minimal service capacity. As service capacity increases, there is a reduction in the number of customers in the line and in their wait times, which decreases queuing cost. The optimal total cost is found at the intersection between the service capacity and waiting line curves.

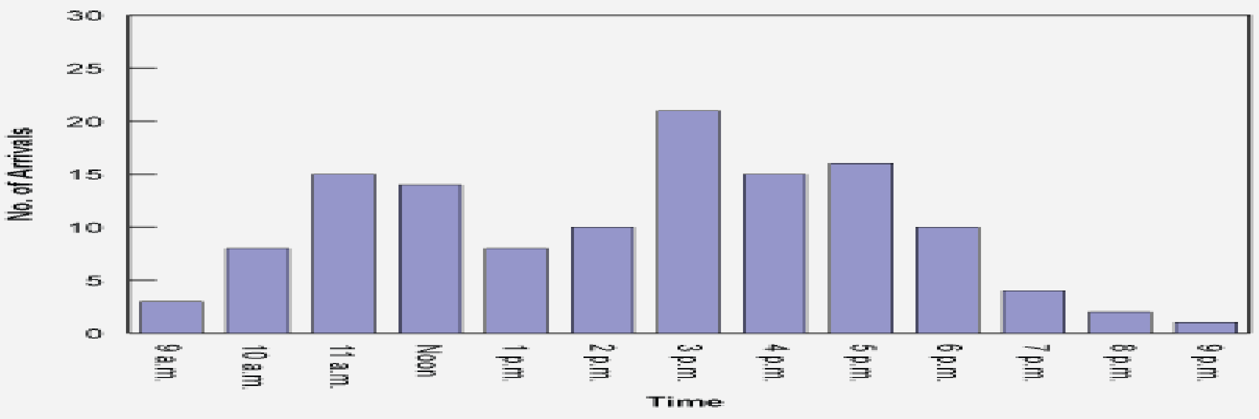
Figure 1: Service Capacity vs. Cost



Queuing Theory

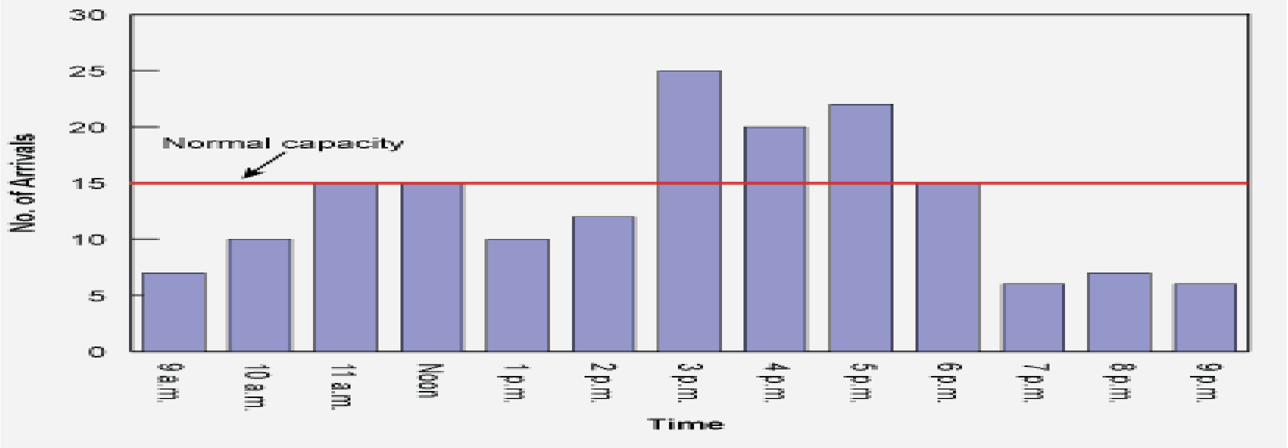
Queuing theory, the mathematical study of waiting in lines, is a branch of operations research because the results often are used when making business decisions about the resources needed to provide service. At its most basic level, queuing theory involves arrivals at a facility (i.e., computer store, pharmacy, bank) and service requirements of that facility (i.e., technicians, pharmacists, tellers). The number of arrivals generally fluctuates over the course of the hours that the facility is available for business (Figure 2).

Figure 2: Number of Arrivals at Facility



Customers demand varying degrees of service, some of which can exceed normal capacity (Figure 3). The store manager or business owner can exercise some control over arrivals. For example, the simplest arrival-control mechanism is the posting of business hours. Other common techniques include lowering prices on typically slow days to balance customer traffic throughout the week and establishing appointments with specific times for customers. The point is that queues are within the control of the system management and design.

Figure 3: Service Requirements



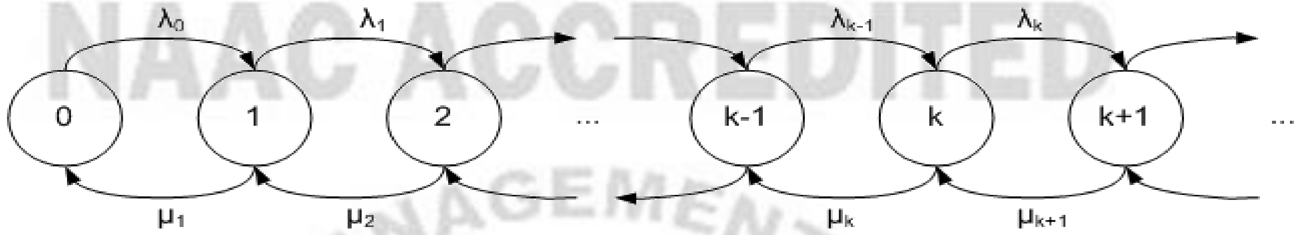
Queuing management consists of three major components:

1. How customers arrive
2. How customers are serviced
3. The condition of the customer exiting the system

Birth-Death queuing models and its steady state solution

The birth–death process is a special case of continuous-time Markov process where the state transitions are of only two types: "births" which increase the state variable by one and "deaths" which decrease the state by one. The model's name comes from a common application, the use of such models to represent the current size of a population where the transitions are literal births and deaths. Birth–death processes have many applications in demography, queueing theory, performance engineering, epidemiology or in biology. They may be used, for example to study the evolution of bacteria, the number of people with a disease within a population, or the number of customers in line at the supermarket.

When a birth occurs, the process goes from state n to $n + 1$. When a death occurs, the process goes from state n to state $n - 1$. The process is specified by birth rates $\{\lambda_i\}_{i=0\dots\infty}$ and death rates $\{\mu_i\}_{i=1\dots\infty}$.



<u>M/M/1 and M/M/C models with infinite/finite waiting space</u>

In queuing theory the birth–death process is the most fundamental example of a queuing model, the *M/M/C/K/∞/FIFO* (in complete Kendall's notation) queue. This is a queue with Poisson arrivals, drawn from an infinite population, and *C* servers with exponentially distributed service time with *K* places in the queue. Despite the assumption of an infinite population this model is a good model for various telecommunication systems.

M/M/1 queue

The M/M/1 is a single server queue with an infinite buffer size. In a non-random environment the birth–death process in queuing models tend to be long-term averages, so the average rate of arrival is given as $\lambda$and the average service time as $1/\mu$. The birth and death process is a M/M/1 queue when,

$$\lambda_i = \lambda \text{ and } \mu_i = \mu \text{ for all } i.$$

The difference equations for the probability that the system is in state *k* at time *t* are,

$$p_0'(t) = \mu_1 p_1(t) - \lambda_0 p_0(t)$$

$$p_k'(t) = \lambda_{k-1} p_{k-1}(t) + \mu_{k+1} p_{k+1}(t) - (\lambda_k + \mu_k) p_k(t)$$

M/M/c queue

The M/M/c is a multi-server queue with C servers and an infinite buffer. This differs from the M/M/1 queue only in the service time which now becomes

$$\mu_i = i\mu \text{ for } i \leq C$$

and

$$\mu_i = C\mu \text{ for } i \geq C$$

with

$$\lambda_i = \lambda \text{ for all } i.$$

M/M/1/K queue

The M/M/1/K queue is a single server queue with a buffer of size K. This queue has applications in telecommunications, as well as in biology when a population has a capacity limit. In telecommunication we again use the parameters from the M/M/1 queue with,

$$\lambda_i = \lambda \text{ for } 0 \leq i < K$$
$$\lambda_i = 0 \text{ for } i \geq K$$
$$\mu_i = \mu \text{ for } 1 \leq i \leq K.$$

In biology, particularly the growth of bacteria, when the population is zero there is no ability to grow so,

$$\lambda_0 = 0.$$

Additionally if the capacity represents a limit where the population dies from over population,

$$\mu_K = 0.$$

The differential equations for the probability that the system is in state *k* at time *t* are,

$$p_0'(t) = \mu_1 p_1(t) - \lambda_0 p_0(t)$$
$$p_k'(t) = \lambda_{k-1} p_{k-1}(t) + \mu_{k+1} p_{k+1}(t) - (\lambda_k + \mu_k) p_k(t) \text{ for } k \leq K$$
$$p_k'(t) = 0 \text{ for } k > K$$

PERT & CPM

INTRODUCTION

Basically, CPM (Critical Path Method) and PERT (Programme Evaluation Review Technique) are project management techniques, which have been created out of the need of Western industrial and military establishments to plan, schedule and control complex projects.

Brief History of CPM/PERT

CPM/PERT or Network Analysis as the technique is sometimes called, developed along two parallel streams, one industrial and the other military. CPM was the discovery of M.R.Walker of E.I.Du Pont de Nemours & Co. and J.E.Kelly of Remington Rand, circa 1957. The computation was designed for the UNIVAC-I computer. The first test was made in 1958, when CPM was applied to the construction of a new chemical plant. In March 1959, the method was applied to a maintenance shut-down at the Du Pont works in Louisville, Kentucky. Unproductive time was reduced from 125 to 93 hours. PERT was devised in 1958 for the POLARIS missile program by the Program Evaluation Branch of the Special Projects office of the U.S.Navy, helped by the Lockheed Missile Systems division and the Consultant firm of Booz-Allen & Hamilton. The calculations were so arranged so that they could be carried out on the IBM Naval Ordinance Research Computer (NORC) at Dahlgren, Virginia.

Planning, Scheduling & Control

Planning, Scheduling (or organising) and Control are considered to be basic Managerial functions, and CPM/PERT has been rightfully accorded due importance in the literature on Operations Research and Quantitative Analysis. Far more than the technical benefits, it was found that PERT/CPM provided a focus around which managers could brain-storm and put their ideas together. It proved to be a great communication medium by which thinkers and planners at one level could communicate their ideas, their doubts and fears to another level. Most important, it became a useful tool for evaluating the performance of individuals and teams.

There are many variations of CPM/PERT which have been useful in planning costs, scheduling manpower and machine time. CPM/PERT can answer the following important questions: How long will the entire project take to be completed? What are the risks involved? Which are the critical activities or tasks in the project which could delay the entire project if they were not completed on time?

Is the project on schedule, behind schedule or ahead of schedule?

If the project has to be finished earlier than planned, what is the best way to do this at the least cost?

The Framework for PERT and CPM

Essentially, there are six steps which are common to both the techniques. The procedure is listed below:

1. Define the Project and all of it's significant activities or tasks. The Project (made up of several tasks) should have only a single start activity and a single finish activity.
2. Develop the relationships among the activities. Decide which activities must precede and which must follow others.
3. Draw the "Network" connecting all the activities. Each Activity should have unique event numbers. Dummy arrows are used where required to avoid giving the same numbering to two activities.
4. Assign time and/or cost estimates to each activity
5. Compute the longest time path through the network. This is called the critical path.
6. Use the Network to help plan, schedule, monitor and control the project.

The Key Concept used by CPM/PERT is that a small set of activities, which make up the longest path through the activity network control the entire project. If these "critical" activities could be identified and assigned to responsible persons, management resources could be optimally used by concentrating on the few activities which determine the fate of the entire project.

Non-critical activities can be replanned, rescheduled and resources for them can be reallocated flexibly, without affecting the whole project.

<u>UNIT-IV (Transportation & Assignment Problem)</u>

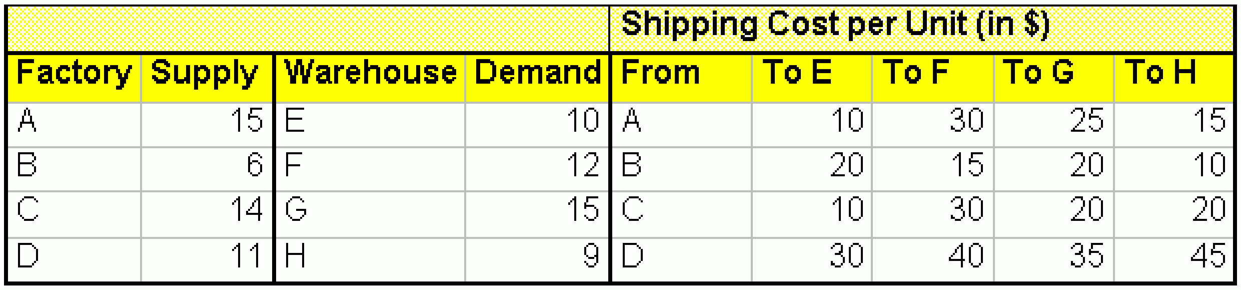
General structure of transportation problem, solution procedure for transportation problem, methods for finding initial solution, test for optimality, maximization of transportation problem, transportation problem

There is a type of linear programming problem that may be solved using a simplified version of the simplex technique called transportation method. Because of its major application in solving problems involving several product sources and several destinations of products, this type of problem is frequently called the transportation problem. It gets its name from its application to problems involving transporting products from several sources to several destinations. Although the formation can be used to represent more general assignment and scheduling problems as well as transportation and distribution problems. The two common objectives of such problems are either (1) minimize the cost of shipping *m* units to *n* destinations or (2) maximize the profit of shipping *m* units to *n* destinations.

Let us assume there are *m* sources supplying *n* destinations. Source capacities, destinations requirements and costs of material shipping from each source to each destination are given constantly. The transportation problem can be described using following linear programming mathematical model and usually it appears in a transportation tableau.

There are three general steps in solving transportation problems.

We will now discuss each one in the context of a simple example. Suppose one company has four factories supplying four warehouses and its management wants to determine the minimum-cost shipping schedule for its weekly output of chests. Factory supply, warehouse demands, and shipping costs per one chest (unit) are shown.



				Shipping Cost per Unit (in $)				
Factory	Supply	Warehouse	Demand	From	To E	To F	To G	To H
A	15	E	10	A	10	30	25	15
B	6	F	12	B	20	15	20	10
C	14	G	15	C	10	30	20	20
D	11	H	9	D	30	40	35	45

”Data for Transportation Problem”

At first, it is necessary to prepare an initial feasible solution, which may be done in several different ways; the only requirement is that the destination needs be met within the constraints of source supply.

The Transportation Matrix

The transportation matrix for this example appears in Table 7.2, where supply availability at each factory is shown in the far right column and the warehouse demands are shown in the bottom row. The unit shipping costs are shown in the small boxes within the cells at the initiation of solving all cells are empty). It is important at this step to make sure that the total supply availabilities and total demand requirements are equal. Often there is an excess supply or demand. In such situations, for the transportation method to work, a dummy warehouse or factory must be added. Procedurally, this involves inserting an extra row (for an additional factory) or an extra column (for an ad warehouse). The amount of supply or demand required by the"dummy" equals the difference between the row and column totals.

In this case there is:

Total factory supply … 51

Total warehouse requirements … 52

This involves inserting an extra row - an additional factory. The amount of supply by the dummy equals the difference between the row and column totals. In this case there is 52 – 51 = 1. The cost figures in each cell of the dummy row would be set at zero so any units sent there would not incur a transportation cost. Theoretically, this adjustment is equivalent to the simplex procedure of inserting a slack variable in a constraint inequality to convert it to an equation, and, as in the simplex, the cost of the dummy would be zero in the objective function.

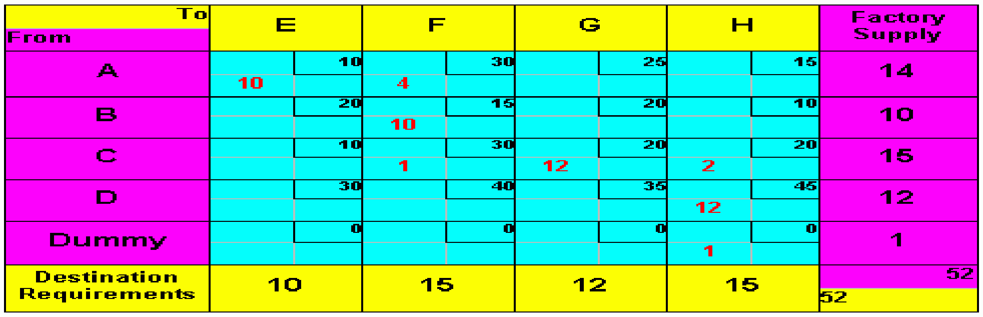
To / From	E	F	G	H	Factory Supply
A	10	30	25	15	14
B	20	15	20	10	10
C	10	30	20	20	15
D	30	40	35	45	12
Dummy	0	0	0	0	1
Destination Requirements	10	15	12	15	52 / 52

Initial Feasible Solution

Initial allocation entails assigning numbers to cells to satisfy supply and demand constraints. Next we will discuss several methods for doing this: the Northwest-Corner method, Least-Cost method, and Vogel's approximation method (VAM).

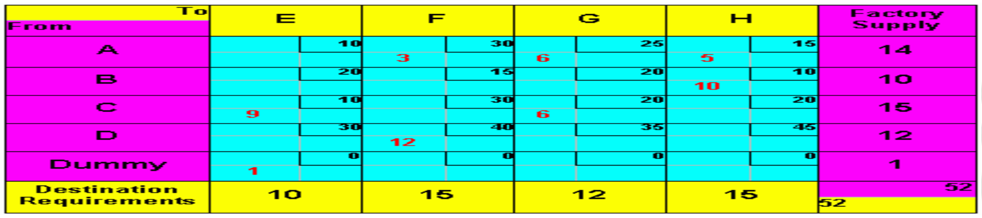
Table shows a northwest-corner assignment. (Cell A-E was assigned first, A-F second, B-F third, and so forth.) Total cost : 10*10 + 30*4 + 15*10 + 30*1 + 20*12 + 20*2 + 45*12 + 0*1 = 1220($). Inspection of Table indicates some high-cost cells were assigned and some low-cost cells bypassed by using the northwest-comer method. Indeed, this is to be expected since this method ignores costs in favor of following an easily programmable allocation algorithm. Table shows a least-cost assignment. (Cell Dummy-E was assigned first, C-E second, B-H third, A-H fourth, and so on.) Total cost : 30*3 + 25*6 + 15*5 +10*10 + 10*9 + 20*6 + 40*12 + 0*1= 1105 ($).

Table shows the VAM assignments. (Cell Dummy-G was assigned first, B-F second, C-E third, A-H fourth, and so on.) Note that this starting solution is very close to the optimal solution obtained after making all possible improvements (see next chapter) to the starting solution obtained using the northwest-comer method. (See Table 7.3.) Total cost: 15*14 + 15*10 + 10*10 + 20*4 + 20*1 + 40*5 + 35*7 + 0*1 = 1005 ($).



To From	E	F	G	H	Factory Supply
A	10 10	30 4	25	15	14
B	20	15 10	20	10	10
C	10	30 1	20 12	20 2	15
D	30	40	35	45 12	12
Dummy	0	0	0	0 1	1
Destination Requirements	10	15	12	15	52 52

”Northwest – Corner Assignment”



To From	E	F	G	H	Factory Supply
A	10	30 3	25 6	15 5	14
B	20	15	20	10 10	10
C	10 9	30	20 6	20	15
D	30	40 12	35	45	12
Dummy	0 1	0	0	0	1
Destination Requirements	10	15	12	15	52 52

"Least - Cost Assignment"

To / From	E		F		G		H		Factory Supply
A		10		30		25	14	15	14
B		20	10	15		20		10	10
C	10	10		30	4	20	1	20	15
D		30	5	40	7	35		45	12
Dummy		0		0	1	0		0	1
Destination Requirements	10		15		12		15		52 / 52

"VAM Assignment"

Develop Optimal Solution

To develop an optimal solution in a transportation problem involves evaluating each unused cell to determine whether a shift into it is advantageous from a total-cost stand point. If it is, the shift is made, and the process is repeated. When all cells have been evaluated and appropriate shifts made, the problem is solved. One approach to making this evaluation is the Stepping stone method. The term stepping stone appeared in early descriptions of the method, in which unused cells were referred to as "water" and used cells as "stones"— from the analogy of walking on a path of stones half-submerged in water. The stepping stone method was applied to the VAM initial solution, as shown

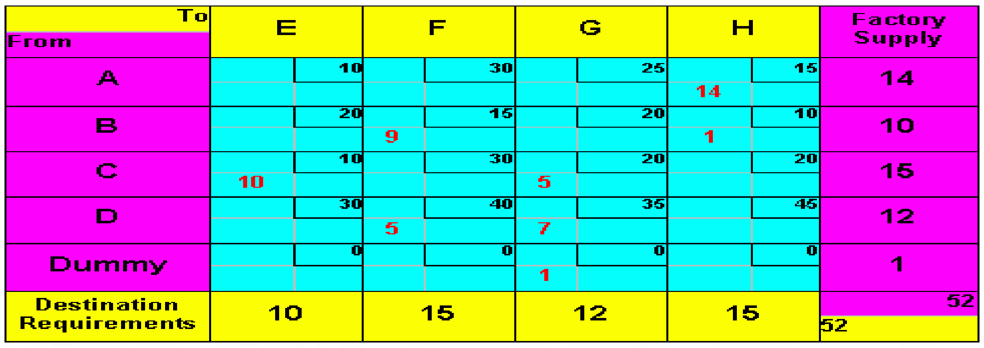
Table shows the optimal solutions reached by the Stepping stone method. Such solution is very close to the solution found using VAM method.

To / From	E		F		G		H		Factory Supply
A		10		30		25	14	15	14
B		20	10	15		20		10	10
C	10	10		30	4	20		20	15
D		30	4	40	8	35		45	12
Dummy		0	1	0		0		0	1
Destination Requirements	10		15		12		15		52 / 52

"Optimal Matrix, With Minimum Transportation Cost of $1,000."

Alternate Optimal Solutions

When the evaluation of any empty cell yields the same cost as the existing allocation, an alternate optimal solution exists (see Stepping Stone Method – alternate solutions). Assume that all other cells are optimally assigned. In such cases, management has additional flexibility and can invoke non transportation cost factors in deciding on a final shipping schedule.



From \ To	E	F	G	H	Factory Supply
A	10	30	25	15 / 14	14
B	20	15 / 9	20	10 / 1	10
C	10 / 10	30	20 / 5	20	15
D	30	40 / 5	35 / 7	45	12
Dummy	0	0	0 / 1	0	1
Destination Requirements	10	15	12	15	52 / 52

"Alternate Optimal Matrix for the Chest Transportation Problem, With Minimum Transportation Cost of $1,000.

Degeneracy

Degeneracy exists in a transportation problem when the number of filled cells is less than the number of rows plus the number of columns minus one (m + n - 1). Degeneracy may be observed either during the initial allocation when the first entry in a row or column satisfies both the row and column requirements or during the Stepping stone method application, when the added and subtracted values are equal. Degeneracy requires some adjustment in the matrix to evaluate the solution achieved. The form of this adjustment involves inserting some value in an empty cell so a closed path can be developed to evaluate other empty cells. This value may be thought of as an infinitely small amount, having no direct bearing on the cost of the solution. Procedurally, the value (often denoted by the Greek letter epsilon, - ε) is used in exactly the same manner as a real number except that it may initially be placed in any empty cell, even though row and column requirements have been met by real numbers. A degenerate transportation problem showing a Northwest Corner initial allocation is presented in Table 7.8, where we can see that if ε were not assigned to the matrix, it would be impossible to evaluate several cells. Once a ε has been inserted into the solution, it remains there until it is removed by subtraction or until a final solution is reached. While the choice of where to put an ε is arbitrary, it saves time if it is placed where it may be used to evaluate as many cells as possible without being shifted.

Transportation Problem with a Maximization as a Criterion

A fictive corporation A has a contract to supply motors for all tractors produced by a fictive corporation B. Corporation B manufactures the tractors at four locations around Central Europe: Prague, Warsaw, Budapest and Vienna.

Plans call for the following numbers of tractors to be produced at each location:

Prague 9 000

Warsaw 12 000

Budapest 9 000

Corporation A has three plants that can produce the motors. The plants and production capacities are

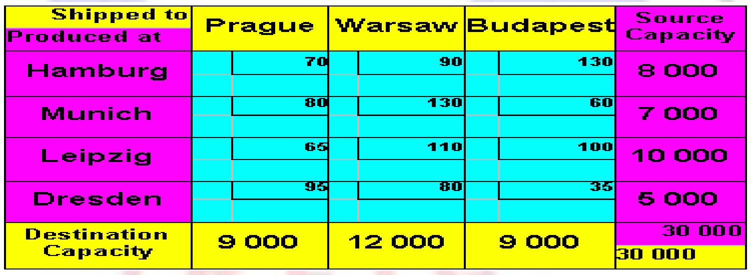
Hamburg 8 000

Munich 7 000

Leipzig 10 000

Dresden 5 000

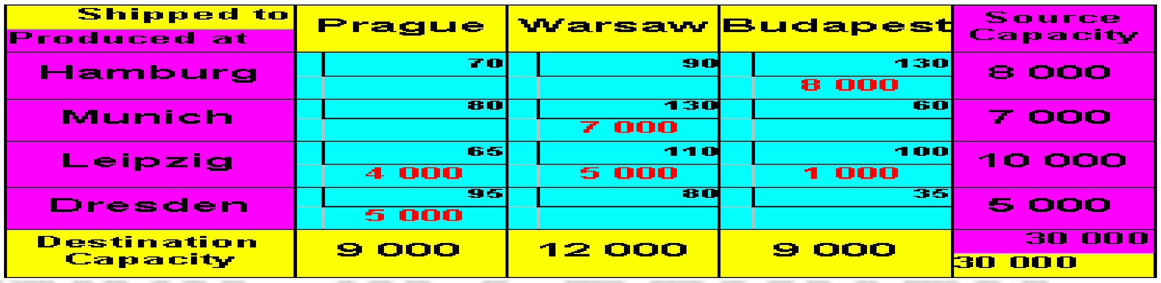
Due to varying production and transportation costs, the profit earns on each motor depends on where they were produced and where they were shipped. The following transportation table gives the accounting department estimates of the euro profit per unit (motor).



Shipped to / Produced at	Prague	Warsaw	Budapest	Source Capacity
Hamburg	70	90	130	8 000
Munich	80	130	60	7 000
Leipzig	65	110	100	10 000
Dresden	95	80	35	5 000
Destination Capacity	9 000	12 000	9 000	30 000 / 30 000

"The Euro Profit Per One Shipped Motor"

Table shows a highest - profit assignment (Least Cost method modification). In contrast to the Least – Cost method it allocates as much as possible to the highest-cost cell. (Cell Hamburg - Budapest was assigned first, Munich - Warsaw second, Leipzig - Warsaw third, Leipzig – Budapest fourth, Dresden – Prague fifth and Leipzig – Prague sixth.) Total profit : 3 335 000 euro.



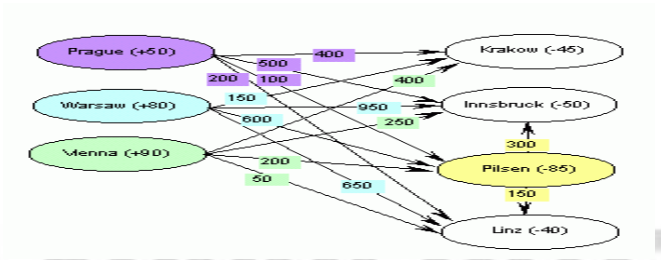
Shipped to / Produced at	Prague		Warsaw		Budapest		Source Capacity
Hamburg		70		90	8 000	130	8 000
Munich		80	7 000	130		60	7 000
Leipzig	4 000	65	5 000	110	1 000	100	10 000
Dresden	5 000	95		80		35	5 000
Destination Capacity	9 000		12 000		9 000		30 000 / 30 000

"Highest - Profit Assignment"

Applying the Stepping Stone method (modified for maximization purposes) to the initial solution we can see that no other transportation schedule can increase the profit and so the Highest – Profit initial allocation is also an optimal solution of this transportation problem.

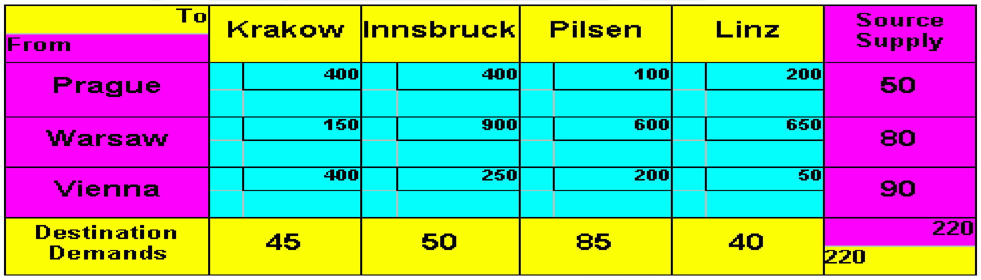
The Transshipment Problem

The transshipment problem is similar to the transportation problem except that in the transshipment problem it is possible to both ship into and out of the same node (point). For the transportation problem, you can ship only from supply points to demand points. For the transshipment problem, you can ship from one supply point to another or from one demand point to another. Actually, designating nodes as supply points or demand points becomes confusing when you can ship both into and out of a node. You can make the designations clearer if you classify nodes by their net stock position-excess (+), shortage (-), or 0. One reason to consider transshipping is that units can sometimes be shipped into one city at a very low cost and then transshipped to other cities. In some situations, this can be less expensive than direct shipment. Let's consider the balanced transportation problem as an example. Picture shows the net stock positions for the three warehouses and four customers. Say that it is possible to transship through Pilsen to both Innsbruck and Linz. The transportation cost from Pilsen to Innsbruck is 300 euro per unit, so it costs less to ship from Warsaw to Innsbruck by going through Pilsen. The direct cost is 950 euro, and the transshipping cost is 600 + 300 = 900 euro. Because the transportation cost is 300 euro from Pilsen to Innsbruck, the cost of transshipping from Prague through Pilsen to Innsbruck is 400 euro per unit. It is cheaper to transship from Prague through Pilsen than to ship directly from Prague to Innsbruck.



"Transshipment Example in the Form of a Network Model"

There are two possible conversions to a transportation model. In the first conversion, make each excess node a supply point and each shortage node a demand point. Then, find the cheapest method of shipping from surplus nodes to shortage nodes considering all transshipment possibilities. Let's perform the first conversion for the Picture 7.1 example. Because a transportation table Prague, Warsaw, and Vienna have excesses, they are the supply points. Because Krakow, Pilsen, Innsbruck, and Linz have shortages, they are the demand points. The cheapest cost from Warsaw to Innsbruck is 900 euro, transshipping through Pilsen. The cheapest cost from Prague to Innsbruck is 400 euro, transshipping through Pilsen too. The cheapest cost from all other supply points to demand points is obtained through direct shipment. Table 7.11 shows the balanced transportation table for this transshipment problem. For a simple transportation network, finding all of the cheapest routes from excess nodes to shortage nodes is easy. You can list all of the possible routes and select the cheapest. However, for a network with many nodes and arcs, listing all of the possible routes is difficult.



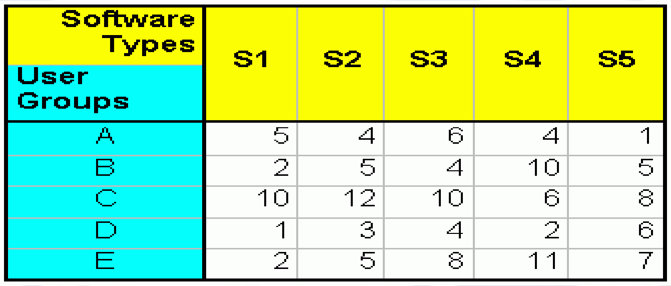
To / From	Krakow	Innsbruck	Pilsen	Linz	Source Supply
Prague	400	400	100	200	50
Warsaw	150	900	600	650	80
Vienna	400	250	200	50	90
Destination Demands	45	50	85	40	220 / 220

"The Transshipment Problem After Conversion to a Transportation Model"

The second conversion of a transshipment problem to a transportation model doesn't require finding all of the cheapest routes from excess nodes table to shortage nodes. The second conversion requires more supply and demand nodes than the first conversion, because the points where you can ship into and out of occur in the converted transportation problem twice – first as a supply point and second as a demand point.

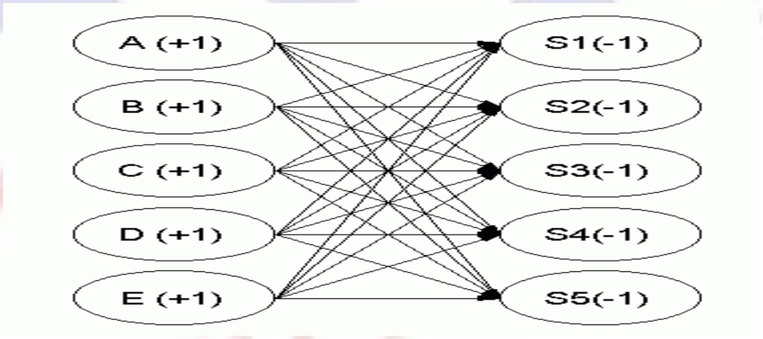
Assignment problem approach of the assignment model, solution methods of assignment problem, maximization in an assignment, unbalanced assignment problem, restriction on assignment.

Another transportation problem is the assignment problem. You can use this problem to assign tasks to people or jobs to machines. You can also use it to award contracts to bidders. Let's describe the assignment problem as assigning *n* tasks to *n* people. Each person must do one and only one task, and each task must be done by only one person. You represent each person and each task by a node. Number the people *1* to *n*, and number the tasks *1* to *n*. The assignment problem can be described simply using a verbal model or using a linear programming mathematical model . For example, say that five groups of computer users must be trained for five new types of software. Because the users have different computer skill levels, the total cost of trainings depends on the assignments.



Software Types / User Groups	S1	S2	S3	S4	S5
A	5	4	6	4	1
B	2	5	4	10	5
C	10	12	10	6	8
D	1	3	4	2	6
E	2	5	8	11	7

”Cost of Trainings According to the Groups of Users”



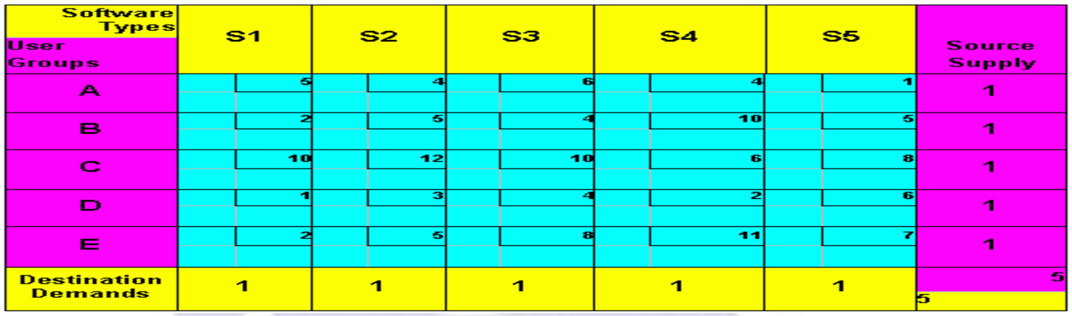
"Network Model for Assignment Problem"

Table shows the cost of training for each assignment of a user group (A through E) to a software type (S1 through S5). Picture is a network model of this problem. A balanced assignment problem has the same number of people and tasks. For a balanced assignment problem, the relationships are all equal. Each person must do a task.

For an unbalanced assignment problem with mor$\leq$people than tasks, some people don't have to do a task and the first class of constraints is of the type. In general, the simplex method does not guarantee that the optimal values of the decision variables are integers. Fortunately, for the assignment model, all of the corner point solutions have integer values for all of the variables. Therefore, when the simplex method determines the optimal corner point, all of the variable values are integers and the constraints require that the integers be either 1 or 0 (Boolean).

Conversion to a Balanced Transportation Table

It's not surprising that the variable values for corner point solutions to the assignment model are integers. The assignment model is a special case of the transportation problem, and the transportation problem has integer variable values for every corner point. For the assignment model, the number of supply and demand points are both *n*. The supply points correspond to each person, and the demand points correspond to each task. Furthermore, every supply amount is 1 and every demand amount is 1. There is one of each person and one of each task.



Software Types / User Groups	S1	S2	S3	S4	S5	Source Supply
A	5	4	6	4	1	1
B	2	5	4	10	5	1
C	10	12	10	6	8	1
D	1	3	4	2	6	1
E	2	5	8	11	7	1
Destination Demands	1	1	1	1	1	5 / 5

"The Computer Users Assignment Problem in the Transportation Table Format"

Table represents the computer users assignment problem in the balanced transportation table format. For the computer users assignment problem, you minimize the total cost of training. Because number of users = 5 and software types = 5, the transportation problem is balanced. You can use standard method for initial solution finding (Northwest-Corner method, Least-Cost method, or Vogel's approximation method (VAM) and a Stepping–Stone method for developing an optimal solution. Thinking of the transportation problem as shipping items between cities is helpful, but it's the mathematical structure that is important.

The assignment problem has the same mathematical structure as the transportation problem, even though it has nothing to do with shipping units. Note, that each feasible solution of assignment problem will always be strongly degenerated (number of nonzero variables will always be *n*)

COMPUTER ARCHITECTURE (203)

UNIT 1

REGISTER TRANSFER LANGUAGE

Digital systems are composed of modules that are constructed from digital components, such as registers, decoders, arithmetic elements, and control logic. The modules are interconnected with common data and control paths to form a digital computer system. The operations executed on data stored in registers are called *microoperations.* A micro operation is an elementary operation performed on the information stored in one or more registers Examples are shift, count, clear, and load Some of the digital components from before are registers that implement microoperations. The internal hardware organization of a digital computer is best defined by specifying. The set of registers it contains and their functions. The sequence of microoperations performed on the binary information stored The control that initiates the sequence of microoperations Use symbols, rather than words, to specify the sequence of microoperations The symbolic notation used is called a *register transfer language* A programming language is a procedure for writing symbols to specify a given computational process. Define symbols for various types of microoperations and describe associated hardware that can implement the microoperations.

REGISTER TRANSFER

Designate computer registers by capital letters to denote its function The register that holds an address for the memory unit is called MAR The program counter register is called PC IR is the instruction register and R1 is a processor register The individual flip-flops in an *n*-bit register are numbered in sequence from 0 to *n*-1 Refer to Figure 4.1 for the different representations of a register Designate information transfer from one register to another by

R2 ← R1 This statement implies that the hardware is available The outputs of the source must have a path to the inputs of the destination The destination register has a parallel load capability If the transfer is to occur only under a predetermined control condition, designate it by

If (P = 1) *then* (R2 ← R1)

or,

P:R2← R1,

where P is a control function that can be either 0 or 1

Every statement written in register transfer notation implies the presence of the required hardware construction. It is assumed that all transfers occur during a clock edge transition

All microoperations written on a single line are to be executed at the same time

T: R2 ← R1, R1 ← R2

BUS AND MEMORY TRANSFERS

Rather than connecting wires between all registers, a common bus is used A bus structure consists of a set of common lines, one for each bit of a register Control signals determine which register is selected by the bus during each transfer Multiplexers can be used to construct a common bus Multiplexers select the source register whose binary information is then placed on the bus The select lines are connected to the selection inputs of the multiplexers and choose the bits of one register

In general, a bus system will multiplex *k* registers of *n* bits each to produce an *n*-line common bus

This requires *n* multiplexers – one for each bit

The size of each multiplexer must be *k* x 1

The number of select lines required is *log k*

To transfer information from the bus to a register, the bus lines are connected to the inputs of all destination registers and the corresponding load control line must be activated Rather than listing each step as

BUS ← C, R1 ← BUS,

use R1 ← C, since the bus is implied

Instead of using multiplexers, *three-state gates* can be used to construct the bus system A three-state gate is a digital circuit that exhibits three states Two of the states are signals equivalent to logic 1 and 0 The third state is a *high-impedance* state – this behaves like an open circuit, which means the output is disconnected and does not have a logic significance

The three-state buffer gate has a normal input and a control input which determines the output state

With control 1, the output equals the normal input

With control 0, the gate goes to a high-impedance state This enables a large number of three-state gate outputs to be connected with wires to form a common bus line without endangering loading effects. Decoders are used to ensure that no more than one control input is active at any given time

This circuit can replace the multiplexer in Figure 4.3

To construct a common bus for four registers of *n* bits each using three-state buffers, we need *n* circuits with four buffers in each Only one decoder is necessary to select between the four registers

Designate a memory word by the letter M It is necessary to specify the address of M when writing memory transfer operations Designate the address register by AR and the data register by DR The read operation can be stated as:

Read: DR ← M[AR]

The write operation can be stated as:

Write: M[AR] ← R1

ARITHMETIC MICROOPERATIONS

There are four categories of the most common microoperations: Register transfer: transfer binary information from one register to another Arithmetic: perform arithmetic operations on numeric data stored in registers Logic: perform bit manipulation operations on non-numeric data stored in registers Shift: perform shift operations on data stored in registers The basic arithmetic microoperations are addition, subtraction, increment, decrement, and shift

Example of addition: R3 ← R1 +R2

Subtraction is most often implemented through complementation and addition

Example of subtraction: R3 ← R1 + ~~R2~~ + 1 (strikethrough denotes bar on top – 1's complement of R2)

Adding 1 to the 1's complement produces the 2's complement

Adding the contents of R1 to the 2's complement of R2 is equivalent to subtracting

Multiply and divide are not included as microoperations

A microoperation is one that can be executed by one clock pulse

Multiply (divide) is implemented by a sequence of add and shift microoperations (subtract and shift)

To implement the add microoperation with hardware, we need the registers that hold the data and the digital component that performs the addition

A full-adder adds two bits and a previous carry

A *binary adder* is a digital circuit that generates the arithmetic sum of two binary numbers of any length

A binary added is constructed with full-adder circuits connected in cascade

An *n*-bit binary adder requires *n* full-adders

The subtraction A-B can be carried out by the following steps

Take the 1's complement of B (invert each bit)

Get the 2's complement by adding 1

Add the result to A

The addition and subtraction operations can be combined into one common circuit by including an XOR gate with each full-adder

The increment microoperation adds one to a number in a register

This can be implemented by using a binary counter – every time the count enable is active, the count is incremented by one

If the increment is to be performed independent of a particular register, then use half-adders connected in cascade

An *n*-bit binary incrementer requires *n* half-adders

Each of the arithmetic microoperations can be implemented in one composite arithmetic circuit

The basic component is the parallel adder

Multiplexers are used to choose between the different operations

The output of the binary adder is calculated from the following sum:

$D = A + Y + C_{in}$

LOGIC MICROOPERATIONS

Logic operations specify binary operations for strings of bits stored in registers and treat each bit separately

Example: the XOR of R1 and R2 is symbolized by

$P: R1 \leftarrow R1 \oplus R2$

Example: R1 = 1010 and R2 = 1100

1010 Content of R1

1100 Content of R2

0110 Content of R1 after P = 1

Symbols used for logical microoperations:

OR: $\vee$

AND: $\wedge$

XOR: $\oplus$

The + sign has two different meanings: logical OR and summation

When + is in a microoperation, then summation

When + is in a control function, then OR

Example:

$P + Q: R1 \leftarrow R2 + R3, R4 \leftarrow R5 \vee R6$

There are 16 different logic operations that can be performed with two binary variables

The hardware implementation of logic microoperations requires that logic gates be inserted for each bit or pair of bits in the registers. All 16 microoperations can be derived from using four logic gates

Logic microoperations can be used to change bit values, delete a group of bits, or insert new bit values into a register

The *selective-set* operation sets to 1 the bits in A where there are corresponding 1's in B

1010 A before

1100 B (logic operand)

1110 A after

A ← A ∨ B

The *selective-complement* operation complements bits in A where there are corresponding 1's in B

1010 A before

1100 B (logic operand)

0110 A after

A ← A ⊕ B

The *selective-clear* operation clears to 0 the bits in A only where there are corresponding 1's in B

1010 A before

1100 B (logic operand)

0010 A after A ← A ∧ ~~B~~

The *mask* operation is similar to the selective-clear operation, except that the bits of A are cleared only where there are corresponding 0's in B

1010 A before

1100 B (logic operand)

1000 A after

A ← A ∧ B

The *insert* operation inserts a new value into a group of bits

This is done by first masking the bits to be replaced and then Oring them with the bits to be inserted

0110 1010 A before

0000 1111 B (mask)

0000 1010 A after masking

0000 1010 A before

1001 0000 B (insert)

1001 1010 A after insertion

The *clear* operation compares the bits in A and B and produces an all 0's result if the two number are equal

1010 A

1010 B

0000 $A \leftarrow A \oplus B$

SHIFT MICROOPERATIONS

Shift microoperations are used for serial transfer of data. They are also used in conjunction with arithmetic, logic, and other data-processing operations. There are three types of shifts: logical, circular, and arithmetic

A *logical shift* is one that transfers 0 through the serial input

The symbols *shl* and *shr* are for logical shift-left and shift-right by one position

$R1 \leftarrow \text{shl } R1$

The *circular shift* (aka rotate) circulates the bits of the register around the two ends without loss of information

The symbols *cil* and *cir* are for circular shift left and right

The *arithmetic shift* shifts a signed binary number to the left or right

To the left is multiplying by 2, to the right is dividing by 2

Arithmetic shifts must leave the sign bit unchanged

A sign reversal occurs if the bit in R_{n-1} changes in value after the shift

This happens if the multiplication causes an overflow

An overflow flip-flop V_s can be used to detect the overflow

$V_s = R_{n-1} \oplus R_{n-2}$

A bi-directional shift unit with parallel load could be used to implement this

Two clock pulses are necessary with this configuration: one to load the value and another to shift. In a processor unit with many registers it is more efficient to implement the shift operation with a combinational circuit. The content of a register to be shifted is first placed onto a common bus and the output is connected to the combinational shifter, the shifted number is then loaded back into the register. This can be constructed with multiplexers

ARITHMETIC LOGIC SHIFT UNIT

The *arithmetic logic unit (ALU)* is a common operational unit connected to a number of storage registers. To perform a microoperation, the contents of specified registers are placed in the inputs of the ALU. The ALU performs an operation and the result is then transferred to a destination register. The ALU is a combinational circuit so that the entire register transfer operation from the source registers through the ALU and into the destination register can be performed during one clock pulse period

BASIC COMPUTER ORGANIZATION AND DESIGN

COMPUTER REGISTER

Computer has different types of registers:

TYPES OF REGISTERS ARE AS FOLLOWINGS

MAR stand for *Memory Address Register*

This register holds the memory addresses of data and instructions. This register is used to access data and instructions from memory during the execution phase of an instruction. **Suppose CPU wants to store some data in the memory or to read the data from the memory. It places the address of the-required memory location in the MAR**.

Program Counter

The **program counter (PC)**, commonly called the **instruction pointer** (IP) in Intel x86 microprocessors, and sometimes called the **instruction address register**, or just part of the instruction sequencer in some computers, is a processor register.

It is a 16 bit special function register in the 8085 microprocessor. It keeps track of the the **next memory address** of the instruction that is to be executed once the execution of the current instruction is completed. **In other words, it holds the address of the memory location of the next instruction when the current instruction is executed by the microprocessor.**

Accumulator Register

This Register is used for storing the Results those are produced by the System. When the CPU will generate Some Results after the Processing then all the Results will be Stored into the **AC Register**.

Memory Data Register (MDR)

MDR is the register of a computer's control unit that contains the **data to be stored in the computer storage** (e.g. RAM), or the **data after a fetch from the computer storage**. It acts **like a buffer** and holds anything that is copied from the memory ready for the processor to use it. **MDR hold the information before it goes to the decoder.**

MDR which contains the data to be written into or readout of the addressed location. For example, to retrieve the contents of cell 123, we would load the value 123 (in binary, of course) into the MAR and perform a fetch operation. When the operation is done, a copy of the contents of cell 123 would be in the MDR. To store the value 98 into cell 4, we load a 4 into the MAR and a 98 into the MDR and perform a store. When the operation is completed the contents of cell 4 will have been set to 98, by discarding whatever was there previously. The MDR is a two-way register. When data is fetched from memory and placed into the MDR, it is written to in one direction.

When there is a write instruction, the data to be written is placed into the MDR from another CPU register, which then puts the data into memory. The Memory Data Register is half of a minimal interface between a micro program and computer storage, the other half is a memory address register.

Index Register

A hardware element which holds a number that can be added to (or, in some cases, subtracted from) the address portion of a computer instruction to form an effective address. Also known as **base register**. An index register in a computer's CPU is a processor register used for modifying operand addresses during the run of a program.

Memory Buffer Register

MBR stand for *Memory Buffer Register*. This register holds the contents of data or instruction read from, or written in memory. It means that this register is used to store data/instruction coming from the memory or going to the memory.

Data Register

A register used in microcomputers to temporarily store data being transmitted to or from a peripheral device.

INSTRUCTION CODE AND COMPUTER INSTRUCTIONS

Computer instructions are the basic components of a machine language program. They are also known as *macrooperations*, since each one is comprised of a sequences of microoperations. Each instruction initiates a sequence of microoperations that fetch operands from registers or memory, possibly perform arithmetic, logic, or shift operations, and store results in registers or memory. Instructions are encoded as binary *instruction codes*. Each instruction code contains of a *operation code*, or *opcode*, which designates the overall purpose of the instruction (e.g. add, subtract, move, input, etc.). The number of bits allocated for the opcode determined how many different instructions the architecture supports. In addition to the opcode, many instructions also contain one or more *operands*, which indicate where in registers or memory the data required for the operation is located. For example, and add instruction requires two operands, and a not instruction requires one.

```
15    12 11         6 5         0
+----------------------------------+
| Opcode |  Operand    |  Operand   |
+----------------------------------+
```

The opcode and operands are most often encoded as unsigned binary numbers in order to minimize the number of bits used to store them. For example, a 4-bit opcode encoded as a binary number could represent up to 16 different operations.

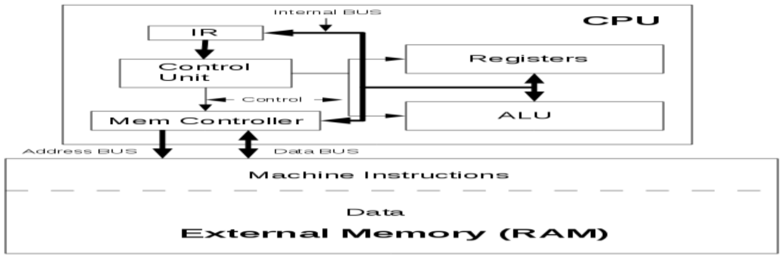
The *control unit* is responsible for decoding the opcode and operand bits in the instruction register, and then generating the control signals necessary to drive all other hardware in the CPU to perform the sequence of microoperations that comprise the instruction.

TIMING AND CONTROL

All sequential circuits in the Basic Computer CPU are driven by a master clock, with the exception of the INPR register. At each clock pulse, the control unit sends control signals to control inputs of the bus, the registers, and the ALU. Control unit design and implementation can be done by two general methods:

A *hardwired* control unit is designed from scratch using traditional digital logic design techniques to produce a minimal, optimized circuit. In other words, the control unit is like an ASIC (application-specific integrated circuit).

A *microprogrammed* control unit is built from some sort of ROM. The desired control signals are simply stored in the ROM, and retrieved in sequence to drive the microoperations needed by a particular instruction.



INSTRUCTION CYCLE: Memory reference, register reference and input out put instruction

The CPU performs a sequence of microoperations for each instruction. The sequence for each instruction of the Basic Computer can be refined into 4 abstract phases:

Fetch instruction

Decode

Fetch operand

Execute

Program execution can be represented as a top-down design:

Program execution

Instruction 1

Fetch instruction

Decode

Fetch operand

Execute

Instruction 2

Fetch instruction

Decode

Fetch operand

Execute

Instruction 3 ...

Program execution begins with:

PC ← address of first instruction, SC ← 0

After this, the SC is incremented at each clock cycle until an instruction is completed, and then it is cleared to begin the next instruction. This process repeats until a HLT instruction is executed, or until the power is shut off.

Instruction Fetch and Decode

The instruction fetch and decode phases are the same for all instructions, so the control functions and microoperations will be independent of the instruction code.

Everything that happens in this phase is driven entirely by timing variables T_0, T_1 and T_2. Hence, all control inputs in the CPU during fetch and decode are functions of these three variables alone.

T_0: AR ← PC

T_1: IR ← M[AR], PC ← PC + 1

T_2: D_{0-7} ← decoded IR(12-14), AR ← IR(0-11), I ← IR(15)

For every timing cycle, we assume SC ← SC + 1 unless it is stated that SC ← 0.

The operation D_{0-7} ← decoded IR(12-14) is not a register transfer like most of our microoperations, but is actually an inevitable consequence of loading a value into the IR register. Since the IR outputs 12-14 are directly connected to a decoder, the outputs of that decoder will change as soon as the new values of IR(12-14) propagate through the decoder.

Note that incrementing the PC at time T_1 assumes that the next instruction is at the next address. This may not be the case if the current instruction is a branch instruction. However, performing the increment here will save time if the next instruction immediately follows, and will do no harm if it doesn't. The incremented PC value is simply overwritten by branch instructions.

In hardware development, unlike serial software development, it is often advantageous to perform work that may not be necessary. Since we can perform multiple microoperations at the same time, we might was well do everything that *might* be useful at the earliest possible time. Likewise, loading AR with the address field from IR at T_2 is only useful if the instruction is a memory-reference instruction. We won't know this until T_3, but there is no reason to wait since there is no harm in loading AR immediately.

By time T_2, the opcode has been decoded by the decoder attached to IR(12-14), and the control signals D_{0-7} are available. At pulse T2, IR(15) is loaded into the I flip-flop. Hence, all of these signals are available for use at pulse T_3.

D_7 indicates that the opcode field is 111, and this is either a register or I/O instruction. (i.e. it is not a memory-reference instruction.)

The I bit allows us to distinguish between register and I/O instructions.

D_7' indicates a memory-reference instruction. In this case, the I bit determines the addressing mode.

What happens at time T_3 therefore depends on the two variables D_7 and I.

Register-reference:

$D_7I'T_3$: Execute register-reference instruction.

I/O:

D_7IT_3: Execute I/O instruction.

Memory-reference with indirect addressing:

$D_7'IT_3$: $AR \leftarrow M[AR]$

Memory-reference with direct addressing:

$D_7'I'T_3$: Nothing. Effective address is already in AR. This wastes a clock cycle when direct addressing is used, but it simplifies the memory-reference execute phase by ensuring that the CPU is in a known state at time T_4.

REGISTER-REFERENCE INSTRUCTION

The control function D_7I' indicates a register-reference instruction, but which one?

Regardless of which instruction it is, T_3 will be the last timing cycle for this instruction, and we will want the next clock pulse to start the timing cycle over at T_0 and fetch the next instruction.

Since $D_7I'T_3$ is common to all register-reference instructions, we will abbreviate it as simple 'r'.

$D_7I'T_3$: $SC \leftarrow 0$

r: $SC \leftarrow 0$

The CLA instruction is indicated by instruction code 7800_{16} (0 111 100000000000_2).

The leftmost 4 bits indicate only that this is a register-reference instruction. The rightmost 12 bits indicate that it is the CLA instruction.

How many register-reference instructions are possible with 12 bits to represent the opcode?

The Basic Computer does not encode the bits for register-reference instructions as a binary number, but instead uses them directly. Hence, only one of these bits can be 1 for a given instruction, and we are limited to 12 register-reference instructions.

For the register-reference execute phase, all control inputs in the CPU are functions of T_3, r and one of the variables B_0 through B_{11}, which come directly from IR(0-11).

CLA 10000000

CLE 01000000

CMA 00100000

HLT 00000001

Can we create register-reference instructions that combine two or more operations, e.g. CLACLE, CLACMA, CLASZA?

Since B_{11} indicates a CLA instruction the execute cycle for CLA is driven by the function rB_{11}.

rB_{11}: AC ← 0

MEMORY-REFERENCE INSTRUCTION

Note that the register transfers listed in Table 5-4 are not microoperations, but symbolic representations of the execute phase. For example, the following operation cannot be implemented directly on the Basic Computer:

AC ← AC + M[AR]

Each memory ref instruction is indicated by a unique D_i signal.

For the memory-reference execute phase, all control inputs in the CPU are functions of timing signals T_4 or later, I, and one of the variables D_0 through D_6.

The execute phase for memory-reference instructions begins at time T_4. The effective address was loaded into AR at time T_2 or T_3.

Several memory-reference instructions operate on AC and an operand from memory.

INPUT OUTPUT INSTRUCTIONS

An instruction in a computer program that causes transfer of data between peripheral devices and main memory, and enables the central processing unit to control the peripheral devices connected to it.

The I/O instructions of the 80386 provide access to the processor's I/O ports for the transfer of data to and from peripheral devices. These instructions have as one operand the address of a port in the I/O address space. There are two classes of I/O instruction: Those that transfer a single item (byte, word, or doubleword) located in a register.

Those that transfer strings of items (strings of bytes, words, or doublewords) located in memory. These are known as "string I/O instructions" or "block I/O instructions".

DESIGN OF ACCUMULATOR LOGIC: BASIC COMPUTER INSTRUCTION FORMAT

The Basic Computer has a 16-bit instruction code similar to the examples described above. It supports direct and indirect addressing modes.

How many bits are required to specify the addressing mode?

```
15 14 12 11      0
+------------------+
| I | OP | ADDRESS |
+------------------+
```

I = 0: direct

I = 1: indirect

UNIT2

CPU

GENERAL REGISTER ORGANIZATION

Memory access is very time-consuming, especially on today's computers, which typically have many wait-states. Compounding the technological limitations on memory speed is the fact than RAM memory banks are getting bigger. Bigger memory units means longer propagation delays in decoding the address.

Memory access is also expensive in terms of the instruction code size necessary to accommodate 32 or 64-bit addresses.

If many CPU registers are available for heavily used variables and intermediate results, we can avoid memory references much of the time, thus vastly increasing program execution speed, and reducing program size.

Figure 8-2 and the diagram below show the internal bus connections for a *general register* organized CPU. This CPU has up to 8 general-purpose registers, which we can call R0 through R7, and an ALU with up to 16 functions.

STACK ORGANIZATION

The Stack Data Structure

A stack is a LIFO (last-in, first-out) list. There are exactly two operations that can be performed on a stack.

Diagram of a stack

A push operation adds a new item to the top of the stack.

A pop operation removes an item from the top of the stack.

```
// sp points to next available element on the stack
// sp is initialized to 0
int     push(int stack[], int sp, int value)
{
if ( sp == STACK_SIZE )
return STACK_FULL;
else
stack[sp++] = value;
return SUCCESS;
}
int     pop(int stack[], int sp)
{
if ( sp == 0 )
return STACK_EMPTY;
else
return stack[--sp];
}
```

Stack Implementation

Register stack (Mano section is confusing)

Memory stack

Uses for Stacks

Subprogram Calls

Stacks are used heavily in subprogram calls for saving the return address, passing arguments, and as space for local variables.

Reverse Polish Notation

Infix notation: a * b + c * d

Infix is hard to evaluate programmatically. Polish mathematician Lukasiewicz invented prefix notation.

Infix: a + b

Prefix (Polish): + a b

Postfix (Reverse Polish): a b +

When converting to postfix, operands should remain in the same order.

Postfix can be evaluated by scanning right to left and using a stack.

a b * c d * +

Convert to RPN and evaluate: (a + b) * c / d * a - c * f

INSTRUCTION FORMATS

Opcode, address, addressing mode

Mano uses "address", I use "operand".

3-operand (memory-to-memory, register-memory, or load-store)

2-operand (memory-to-memory or register-memory)

1-address (accumulator-based)

0-operand (stack-organized)

Evaluate x = (a + b) * (c + d) in several assembly languages.

VAX (3-operand, memory-to-memory)

x86 (2-operand, register-memory)

MIPS (3-operand, load-store)

Mano (1-operand, accumulator-based)

ADDRESSING MODES

Effective address is the memory address where the operand is located.

Implied (accumulator)

Immediate: operand is part of instruction code

Register direct: operand is in a register

Register indirect: effective address is in a register

Auto-increment/auto-decrement: register indirect + update pointer

Direct address: Effective address is part of instruction code

Indirect address: Address containing effective address in instruction code

Indexed (a.k.a. offset): Effective address = content of a general register + offset.

PC-relative: Offset from PC is in instruction code. Very common. Allows for relocatable code. Special case of indexed using PC.

Base register: Another way of looking at indexed.

Data Transfer and Manipulation

As discussed in chapter 5, computers require certain minimum capabilities in order to be able to run any arbitrary program.

Data transfer: I/O, load-store, move

Data manipulation: add, subtract, multiply, divide, and, or, not, shift (logical, arithmetic, rotate)

Program control: branch, jump, skip, call, return, compare, test (and)

Program Control

Status Bit Conditions

Most CPU architectures maintain a number of status bits that indicate the results from the most recent ALU operation. These bits are usually stored in a *status register*, which is not directly accessible as an argument in machine instructions. The bits are set automatically by many instructions, and used by conditional branch instructions that follow.

DIFFERENCE BETWEEN RISC AND CISC

Bear in mind that these are just categories. Words are just labels and essentially empty.

CISC

Large instruction set

Usually memory-to-memory or register-memory

Sophisticated instructions

Instruction cycle takes many clock cycles

Many addressing modes

Examples

x86 Register-memory, 2-operand, 80 instructions on 8086/8088, many more added to 80186, 80286, 80386, 80486, Pentium (80586)

68k series Register-memory, 2-operand, 56 instructions on 68000, more added to 68010, etc.

VAX Memory-to-memory, 3-operand, 240 instructions, 16 addressing modes, 16 general registers. The polyf instruction evaluates a polynomial given a pointer to an array of coefficients, the order of the polynomial, and a value for x.

RISC

Reduced Instruction Set Computer

Small instruction set

Few addressing modes

Only load and store instructions can access memory

Simple, fixed-length instruction code format

Most instructions execute in 1 clock cycle (common exceptions are load, store, mul, div)

Hardwired control unit

Many registers

Examples:

Alpha

ARM

MIPS 60 instructions, load-store architecture, 3 addressing modes (register-direct, immediate, and offset). Most instructions limited to register direct, while a few use immediate for one operand. Load and store instructions have one register direct and one offset operand.

PowerPC

Sun Sparc

PIPELINE AND VECTOR PROCESSING

Parallel Processing

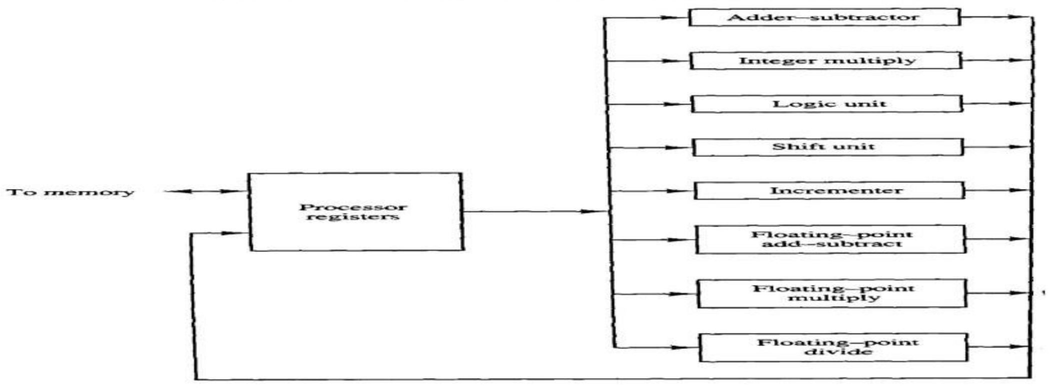
A parallel processing system is able to perform concurrent data processing to achieve faster execution time. The system may have two or more ALUs and be able to execute two or more instructions at the same time. Also, the system may have two or more processors operating concurrently. Goal is to increase the throughput – the amount of processing that can be accomplished during a given interval of time. Parallel processing increases the amount of hardware required

Example: the ALU can be separated into three units and the operands diverted to each unit under the supervision of a control unit

All units are independent of each other

A multifunctional organization is usually associated with a complex control unit to coordinate all the activities among the various components

Figure 9-1 Processor with multiple functional units.



Parallel processing can be classified from:

The internal organization of the processors

The interconnection structure between processors

The flow of information through the system

The number of instructions and data items that are manipulated simultaneously

The sequence of instructions read from memory is the instruction stream

The operations performed on the data in the processor is the data stream

Parallel processing may occur in the instruction stream, the data stream, or both

Computer classification:

Single instruction stream, single data stream – SISD

Single instruction stream, multiple data stream – SIMD

Multiple instruction stream, single data stream – MISD

Multiple instruction stream, multiple data stream – MIMD

SISD – Instructions are executed sequentially. Parallel processing may be achieved by means of multiple functional units or by pipeline processing

SIMD – Includes multiple processing units with a single control unit. All processors receive the same instruction, but operate on different data.

MIMD – A computer system capable of processing several programs at the same time.

We will consider parallel processing under the following main topics:

Pipeline processing

Vector processing

Array processors

Pipelining

Pipelining is a technique of decomposing a sequential process into suboperations, with each subprocess being executed in a special dedicated segment that operates concurrently with all other segments. Each segment performs partial processing dictated by the way the task is partitioned. The result obtained from the computation in each segment is transferred to the next segment in the pipeline. The final result is obtained after the data have passed through all segments. Can imagine that each segment consists of an input register followed by an combinational circuit. A clock is applied to all registers after enough time has elapsed to perform all segment activity

The information flows through the pipeline one step at a time

- Example: Ai * Bi + Ci for i = 1, 2, 3, …, 7

The suboperations performed in each segment are:

R1 ← Ai , R2 ← Bi

R3 ← R1 * R2, R4 ← Ci

R5 ← R3 + R4

Figure 9-2 Example of pipeline processing.

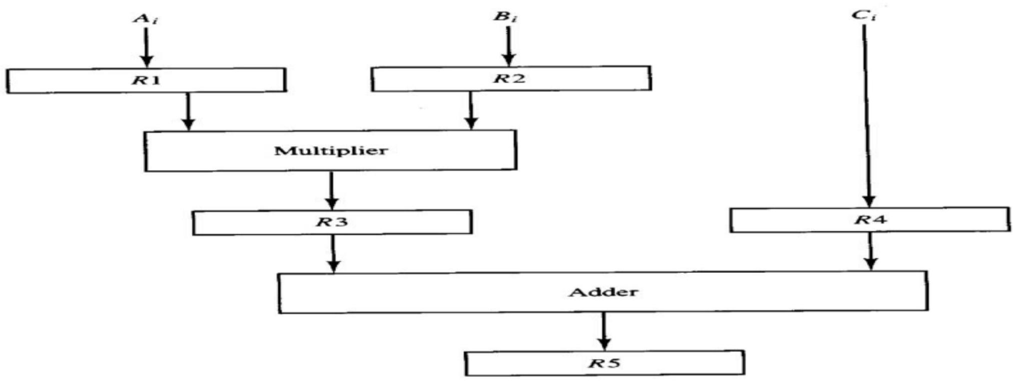
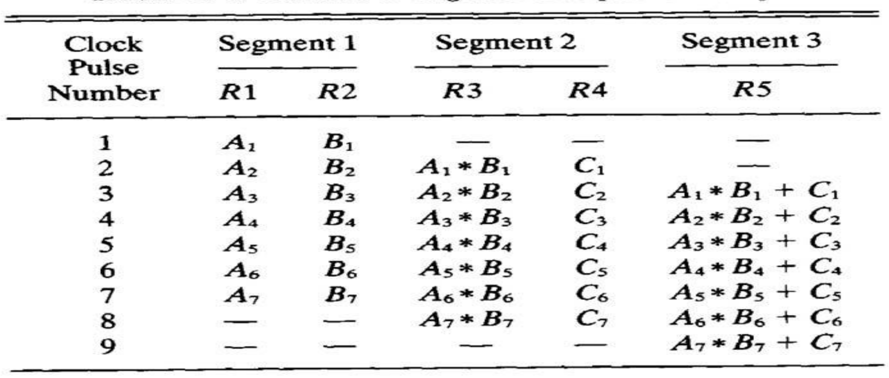


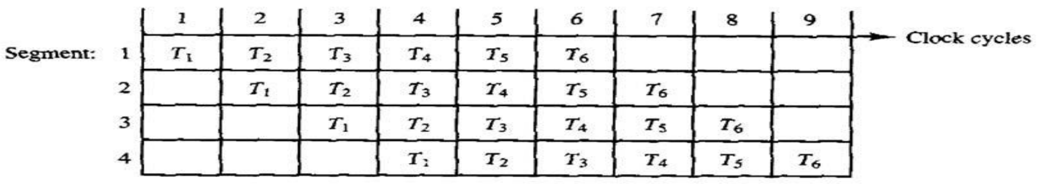
TABLE 9-1 Content of Registers in Pipeline Example



Clock Pulse Number	Segment 1		Segment 2		Segment 3
	R1	R2	R3	R4	R5
1	A_1	B_1	—	—	—
2	A_2	B_2	$A_1 * B_1$	C_1	—
3	A_3	B_3	$A_2 * B_2$	C_2	$A_1 * B_1 + C_1$
4	A_4	B_4	$A_3 * B_3$	C_3	$A_2 * B_2 + C_2$
5	A_5	B_5	$A_4 * B_4$	C_4	$A_3 * B_3 + C_3$
6	A_6	B_6	$A_5 * B_5$	C_5	$A_4 * B_4 + C_4$
7	A_7	B_7	$A_6 * B_6$	C_6	$A_5 * B_5 + C_5$
8	—	—	$A_7 * B_7$	C_7	$A_6 * B_6 + C_6$
9	—	—	—	—	$A_7 * B_7 + C_7$

Any operation that can be decomposed into a sequence of suboperations of about the same complexity can be implemented by a pipeline processor. The technique is efficient for those applications that need to repeat the same task much time with different sets of data. A task is the total operation performed going through all segments of a pipeline. The behavior of a pipeline can be illustrated with a space-time diagram. This shows the segment utilization as a function of time. Once the pipeline is full, it takes only one clock period to obtain an output

Figure 9-4 Space-time diagram for pipeline.



Segment:	1	2	3	4	5	6	7	8	9
1	T_1	T_2	T_3	T_4	T_5	T_6			
2		T_1	T_2	T_3	T_4	T_5	T_6		
3			T_1	T_2	T_3	T_4	T_5	T_6	
4				T_1	T_2	T_3	T_4	T_5	T_6

Clock cycles

Consider a k-segment pipeline with a clock cycle time tp to execute n tasks

The first task T1 requires time ktp to complete

The remaining n – 1 tasks finish at the rate of one task per clock cycle and will be completed after time (n – 1)tp

The total time to complete the n tasks is [k + n – 1]tp

The example of Figure 9-4 requires [4 + 6 – 1] clock cycles to finish

Consider a nonpipeline unit that performs the same operation and takes tn time to complete each task

The total time to complete n tasks would be ntn

The speedup of a pipeline processing over an equivalent nonpipeline processing is defined by the ratio

S = ntn .

(k + n – 1)tp

As the number of tasks increase, the speedup becomes

S = tn

tp

If we assume that the time to process a task is the same in both circuits, tn =k tp

S = ktn = k

tp

Therefore, the theoretical maximum speedup that a pipeline can provide is k

Example:

Cycle time = tp = 20 ns

# of segments = k = 4

# of tasks = n = 100

The pipeline system will take (k + n – 1)tp = (4 + 100 –1)20ns = 2060 ns

Assuming that tn = ktp = 4 * 20 = 80 ns,

A nonpipeline system requires nktp = 100 * 80 = 8000 ns

The speedup ratio = 8000/2060 = 3.88

The pipeline cannot operate at its maximum theoretical rate

One reason is that the clock cycle must be chosen to equal the time delay of the segment with the maximum propagation time

Pipeline organization is applicable for arithmetic operations and fetching instructions

ARITHMETIC PIPELINE

Pipeline arithmetic units are usually found in very high speed computers. They are used to implement floating-point operations, multiplication of fixed- point numbers, and similar computations encountered in scientific problems

Example for floating-point addition and subtraction

Inputs are two normalized floating-point binary numbers

X = A x 2a

Y = B x 2b

A and B are two fractions that represent the mantissas

a and b are the exponents

Four segments are used to perform the following:

Compare the exponents

Align the mantissas

Add or subtract the mantissas

Normalize the result

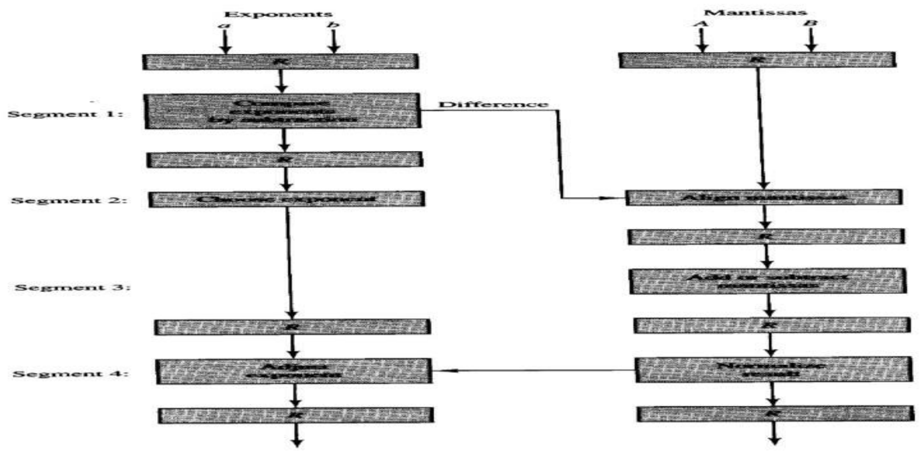


Figure 9-6 Pipeline for floating-point addition and subtraction.

X = 0.9504 x 103 and Y = 0.8200 x 102

The two exponents are subtracted in the first segment to obtain 3-2=1

The larger exponent 3 is chosen as the exponent of the result

Segment 2 shifts the mantissa of Y to the right to obtain Y = 0.0820 x 103

The mantissas are now aligned

Segment 3 produces the sum Z = 1.0324 x 103

Segment 4 normalizes the result by shifting the mantissa once to the right and incrementing the exponent by one to obtain Z = 0.10324 x 104

INSTRUCTION PIPELINE

An instruction pipeline reads consecutive instructions from memory while previous instructions are being executed in other segments. This causes the instruction fetch and execute phases to overlap and perform simultaneous operations. If a branch out of sequence occurs, the pipeline must be emptied and all the instructions that have been read from memory after the branch instruction must be discarded. Consider a computer with an instruction fetch unit and an instruction execution unit forming a two segment pipeline

A FIFO buffer can be used for the fetch segment

Thus, an instruction stream can be placed in a queue, waiting for decoding and processing by the execution segment. This reduces the average access time to memory for reading instructions. Whenever there is space in the buffer, the control unit initiates the next instruction fetch phase

The following steps are needed to process each instruction:

Fetch the instruction from memory

Decode the instruction

Calculate the effective address

Fetch the operands from memory

Execute the instruction

Store the result in the proper place

The pipeline may not perform at its maximum rate due to:

- Different segments taking different times to operate
- Some segment being skipped for certain operations
- Memory access conflicts

Example: Four-segment instruction pipeline

Assume that the decoding can be combined with calculating the EA in one segment

Assume that most of the instructions store the result in a register so that the execution and storing of the result can be combined in one segment

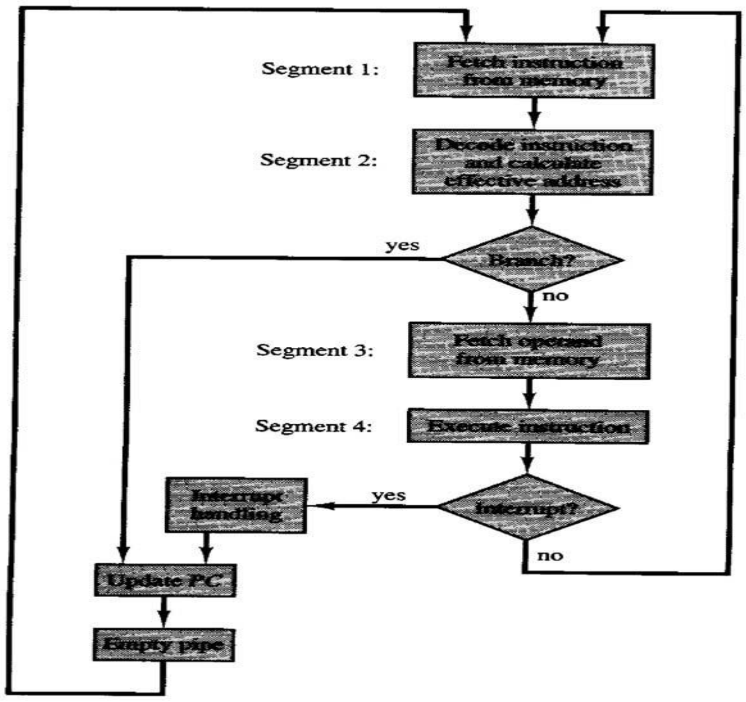


Figure 9-7 Four-segment CPU pipeline.

Up to four suboperations in the instruction cycle can overlap and up to four different instructions can be in progress of being processed at the same time It is assumed that the processor has separate instruction and data memories.

Reasons for the pipeline to deviate from its normal operation are:

Resource conflicts caused by access to memory by two segments at the same time. Data dependency conflicts arise when an instruction depends on the result of a previous instruction, but his result is not yet available. Branch difficulties arise from program control instructions that may change the value of PC

Methods to handle data dependency:

Hardware interlocks are circuits that detect instructions whose source operands are destinations of prior instructions. Detection causes the hardware to insert the required delays without altering the program sequence. Operand forwarding uses special hardware to detect a conflict and then avoid it by routing the data through special paths between pipeline segments. This requires additional hardware paths through multiplexers as well as the circuit to detect the conflict. Delayed load is a procedure that gives the responsibility for solving data conflicts to the compiler. The compiler is designed to detect a data conflict and reorder the instructions as necessary to delay the loading of the conflicting data by inserting no-operation instructions.

Methods to handle branch instructions:

Perfecting the target instruction in addition to the next instruction allows either instruction to be available. A branch target buffer is an associative memory included in the fetch segment of the branch instruction that stores the target instruction for a previously executed branch. It also stores the next few instructions after the branch target instruction. This way, the branch instructions that have occurred previously are readily available in the pipeline without interruption. The loop buffer is a variation of the BTB. It is a small very high speed register file maintained by the instruction fetch segment of the pipeline. Stores all branches within a loop segment. Branch prediction uses some additional logic to guess the outcome of a conditional branch instruction before it is executed. The pipeline then begins perfecting instructions from the predicted path. Delayed branch is used in most RISC processors so that the compiler rearranges the instructions to delay the branch.

A **vector processor**, or **array processor**, is a central processing unit (CPU) that implements an instruction set containing instructions that operate on one-dimensional arrays of data called *vectors*. This is in contrast to a scalar processor, whose instructions operate on single data items. Vector processors can greatly improve performance on certain workloads, notably numerical simulation and similar tasks.

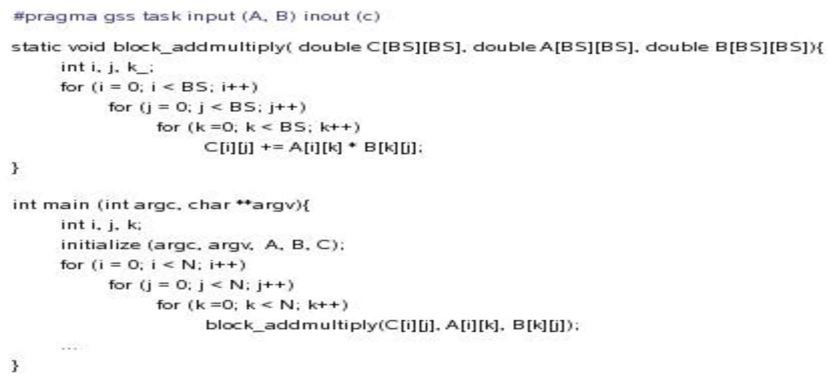
Vector machines appeared in the early 1970s and dominated supercomputer design through the 1970s into the 90s, notably the various Cray platforms. The rapid rise in the price-to-performance ratio of conventional microprocessor designs led to the vector supercomputer's demise in the later 1990s.

VECTOR OPERATIONS

Most commodity CPUs implement architectures that feature instructions for a form vector processing on multiple (vectorized) data sets, typically known as SIMD (**S**ingle **I**nstruction, **M**ultiple **D**ata). Common examples include VIS, MMX, SSE, AltiVec and AVX. Vector processing techniques are also found in video game console hardware and graphics accelerators. In 2000, IBM, Toshiba and Sony collaborated to create the Cell processor, consisting of one scalar processor and eight vector processors, which found use in the Sony PlayStation 3 among other applications. Other CPU designs may include some multiple instructions for vector processing on multiple (vectorised) data sets, typically known as MIMD (**M**ultiple **I**nstruction, **M**ultiple **D**ata) and realized with VLIW. Such designs are usually dedicated to a particular application and not commonly marketed for general purpose computing. In the Fujitsu FR-V VLIW/*vector processor* both technologies are combined.

MATRIX MULTIPLICATION

The functions submitted to the SPUs are block matrix multiplications. As can be seen, a simple annotation before the declaration of the function is enough to allow this behavior with Cell Superscalar. Following the link below you can download the whole file for this example. A simple vectorized version of the matmul can be downloaded at the bottom of the page. The example shows a hyper matrix multiplication calculated by blocks. The functions submitted to the SPUs are block matrix multiplications. An example of coding for Cell Superscalar is shown below.



```
#pragma gss task input (A, B) inout (c)
static void block_addmultiply( double C[BS][BS], double A[BS][BS], double B[BS][BS]){
        int i, j, k_;
        for (i = 0; i < BS; i++)
                for (j = 0; j < BS; j++)
                        for (k =0; k < BS; k++)
                                C[i][j] += A[i][k] * B[k][j];
}

int main (int argc, char **argv){
        int i, j, k;
        initialize (argc, argv,  A, B, C);
        for (i = 0; i < N; i++)
                for (j = 0; j < N; j++)
                        for (k =0; k < N; k++)
                                block_addmultiply(C[i][j], A[i][k], B[k][j]);
        ...
}
```

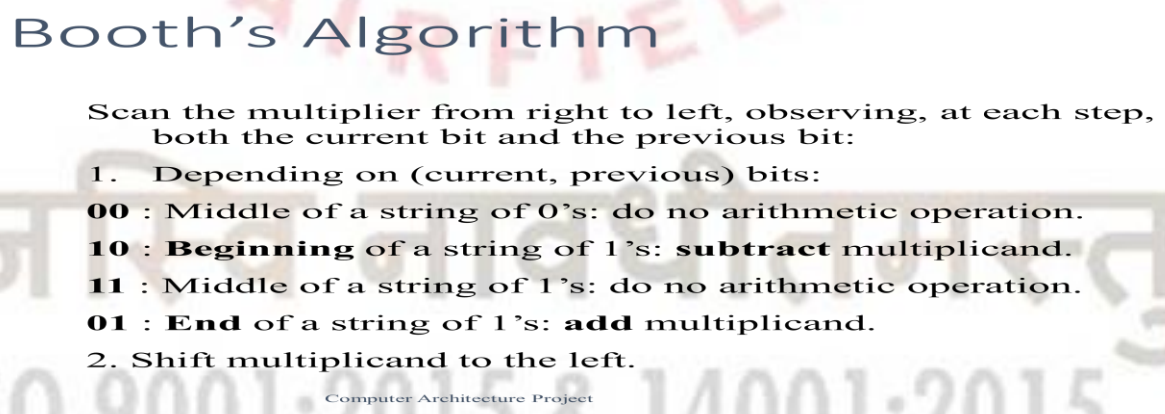
MEMORY INTERLEAVING

In computing, **interleaved memory** is a design made to compensate for the relatively slow speed of core memory or DRAM. With interleaved memory, memory addresses are allocated to each memory bank in turn such that in an interleaved system with 2 memory banks (assuming word-addressable memory) if address 32 belongs to bank 0, 33 would belong to bank 1, 34 would belong to bank 0 and so on. This means that contiguous reads (which are common both in multimedia and execution of programs) and contiguous writes (which are used frequently when filling storage or communication buffers) would actually use each memory bank in turn instead of using the same repeatedly. This results in significantly higher memory throughput as each bank has a minimum waiting time between reads and writes. An interleaved memory with "n" banks is said to be *n-way interleaved*. If there are "n" banks, memory location "i" would reside in bank number $i \bmod n$.

UNIT 3

COMPUTER ARITHMETIC:

INTRODUCTION: Anyone who has fiddled with a calculator long enough knows that it’s relatively easy to trick it into producing ridiculous answers. Indeed, in spite of the possibility of producing answers with a huge number of decimal places, it is important to understand right from the start that (non-trivial) numerical calculations are always wrong. And so unless one knows how wrong a numerical calculation might be, it t is hard to say how accurate the corresponding result might be. In order to determine the size of the error in a numerical calculation, it is crucial to first understand how computers calculate, ho w calculation errors are introduced, and how errors propagate.



Booth’s Algorithm

Scan the multiplier from right to left, observing, at each step, both the current bit and the previous bit:

1. Depending on (current, previous) bits:

00 : Middle of a string of 0’s: do no arithmetic operation.

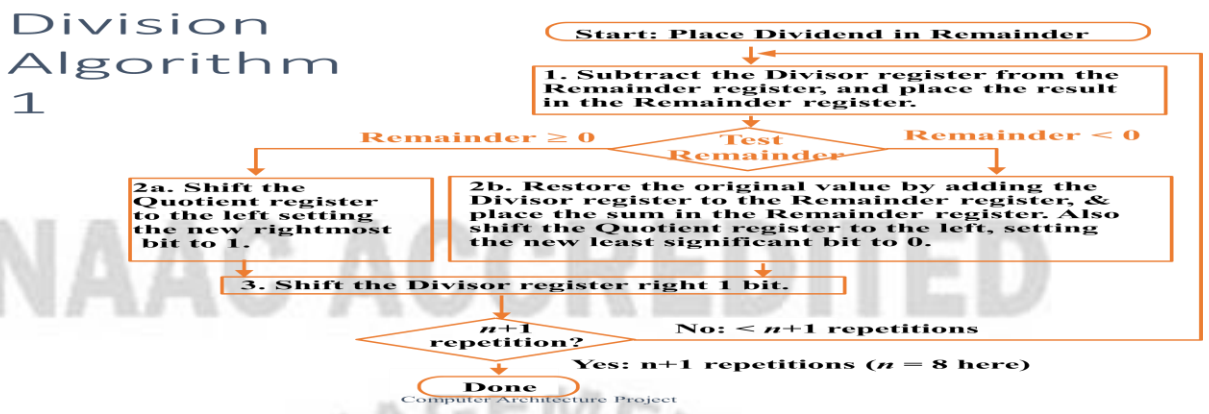
10 : **Beginning** of a string of 1’s: **subtract** multiplicand.

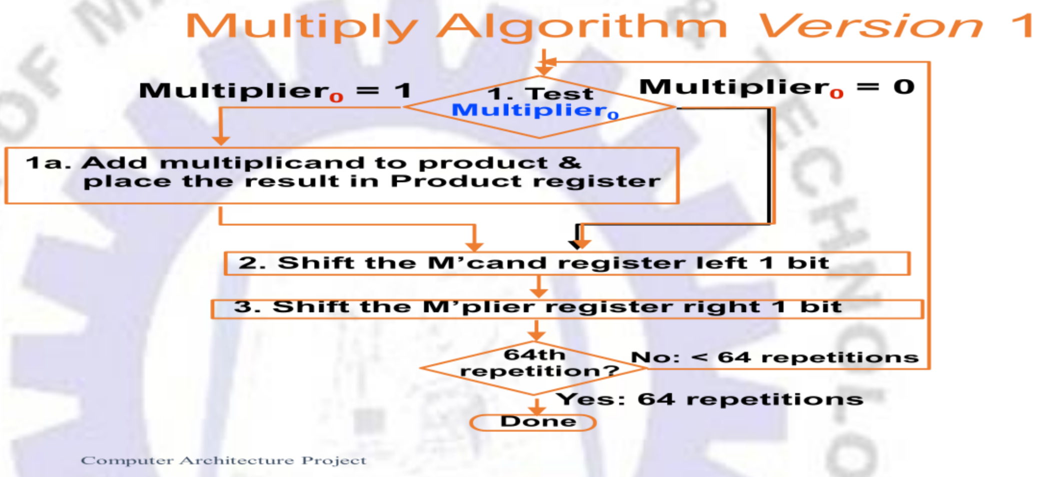
11 : Middle of a string of 1’s: do no arithmetic operation.

01 : **End** of a string of 1’s: **add** multiplicand.

2. Shift multiplicand to the left.

Computer Architecture Project


Division Algorithm 1
Start: Place Dividend in Remainder
1. Subtract the Divisor register from the Remainder register, and place the result in the Remainder register.
Remainder ≥ 0
Test Remainder
Remainder < 0
2a. Shift the Quotient register to the left setting the new rightmost bit to 1.
2b. Restore the original value by adding the Divisor register to the Remainder register, & place the sum in the Remainder register. Also shift the Quotient register to the left, setting the new least significant bit to 0.
3. Shift the Divisor register right 1 bit.
n+1 repetition?
No: < n+1 repetitions
Yes: n+1 repetitions (n = 8 here)
Done
Computer Architecture Project


Multiply Algorithm Version 1
Multiplier0 = 1
1. Test Multiplier0
Multiplier0 = 0
1a. Add multiplicand to product & place the result in Product register
2. Shift the M'cand register left 1 bit
3. Shift the M'plier register right 1 bit
64th repetition?
No: < 64 repetitions
Yes: 64 repetitions
Done
Computer Architecture Project

INPUT OUTPUT ORGANIZATION

PERIPHERAL DEVICES

A computer device, such as a CD-ROM drive or printer that is not part of the essential computer, i.e., the memory and microprocessor. Peripheral devices can be external -- such as a mouse, keyboard, printer, monitor, external Zip drive or scanner -- or internal, such as a CD-ROM drive, CD-R drive or internal modem. Internal peripheral devices are often referred to as *integrated peripherals.*

INPUT-OUTPUT INTERFACE

It would not be practical for every I/O device to be wired to the computer in a different way, so we must have a scheme where the hardware connections are fixed, and yet the communication with the device is flexible, so that the widely varying needs of devices can all be met.

An I/O device, from the viewpoint of the CPU, is a set of registers. The CPU communicates with and controls the I/O device by reading and writing these registers. For example, SPIM, the MIPS simulator, uses two registers to communicate with the keyboard.

The keyboard data register contains the ASCII code of the last key pressed.

The keyboard control register indicates when a new key has been pressed. If bit 0 is one, a key has been pressed since the last character was read. The keyboard controller sets this bit when a key is pressed. It clears this bit when the keyboard data register is read.

The CPU can find out whether a new character is available by reading the keyboard control register and testing bit 0. If bit 0 is 1, it then reads the keyboard data register to get the new key.

For another simple example, see the 10-bit analog to digital converter in the PIC 18f8520 spec sheet.

Accessing I/O devices at the hardware level is a lot like accessing memory. The registers in the I/O devices are connected to the CPU using buses. We need an address bus to specify which I/O device register is to be accessed. We need control lines to specify what kind of access is desired (read, write, reset, etc.) Finally, we need a data bus to transfer the data between the CPU and the device.

Each device has one or more control, status, and data registers at various I/O addresses. A hypothetical example:

Address	Register
ff00	keyboard status
ff01	keyboard data
ff02	display status
ff03	display data
ff04	disk status
ff05	disk block address
ff06	disk block size
ff07	disk data address
...	

I/O read and write operations can be more complex than memory read and write operations, but the basic idea is the same. I/O control generally involves more than just read and write control lines. In a sense, memory can be viewed as a very simple, fast I/O device. Whereas memory is just a large pool of slow, inexpensive registers for storing data, each I/O device register has a unique purpose in controlling a specific I/O device.

This does not affect how the CPU accesses them at the hardware level, but it does affect how they are used by software. Simple device control, such as stating whether an I/O register is to be read or written, can be done over the control lines. More complex devices are often controlled by sending special data blocks called *Peripheral Control Blocks (PCBs)* over the data lines. This is the primary method for communicating with disk drives, for example. Since I/O devices are of a very different nature than CPU circuits, there must be interface hardware to connect each device to the CPU.

ASYNCHRONOUS DATA TRANSFER

The internal operations of a CPU are synchronized by a common clock.

Generally, it is not possible, or at least not practical to synchronize I/O devices with the CPU's internal clock. I/O devices operate at different speeds than the CPU (note that the same I/O device may be connected to computers with vastly different CPU speeds). Data transfer links such as USB, Firewire, and Ethernet, operate on a clock that is separate from the CPU's internal clock, but common to devices at both ends of the link. The operations of an I/O interface must somehow be synchronized with the CPU's activities. This is often done using *handshaking* signals. The I/O device and CPU set wires or flip-flops to 0 or one to indicate their current state. (Recall the Mano FGI and FGO flags.)

The sender on an RS-232 serial interface sets a wire known as RTS (ready to send) to indicate that it has data to send. The receiver responds by by setting the CTS (clear to send) wire to indicate that it's OK to begin sending the next byte.

Newer interfaces such as USB, Ethernet, etc. use more complex setups, but basic 2-wire handshaking is still present in many modern interfaces.

MODE OF TRANSFER

Addressing modes are an aspect of the instruction set architecture in most central processing unit (CPU) designs. The various addressing modes that are defined in a given instruction set architecture define how machine language instructions in that architecture identify the operand (or operands) of each instruction. An addressing mode specifies how to calculate the effective memory address of an operand by using information held in registers and/or constants contained within a machine instruction or elsewhere. In computer programming, addressing modes are primarily of interest to compiler writers and to those who write code directly in assembly language.

Other modes of transfer for code or data

7.1 Absolute/Direct

7.2 Indexed absolute

7.3 Base plus index

7.4 Base plus index plus offset

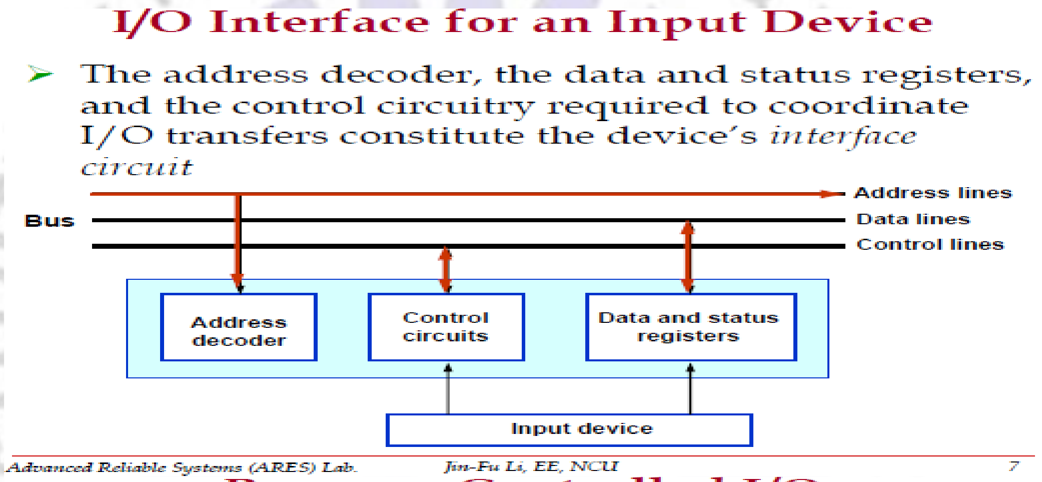
7.5 Scaled

7.6 Register indirect

7.7 Register auto increment indirect

7.8 Auto decrement register indirect

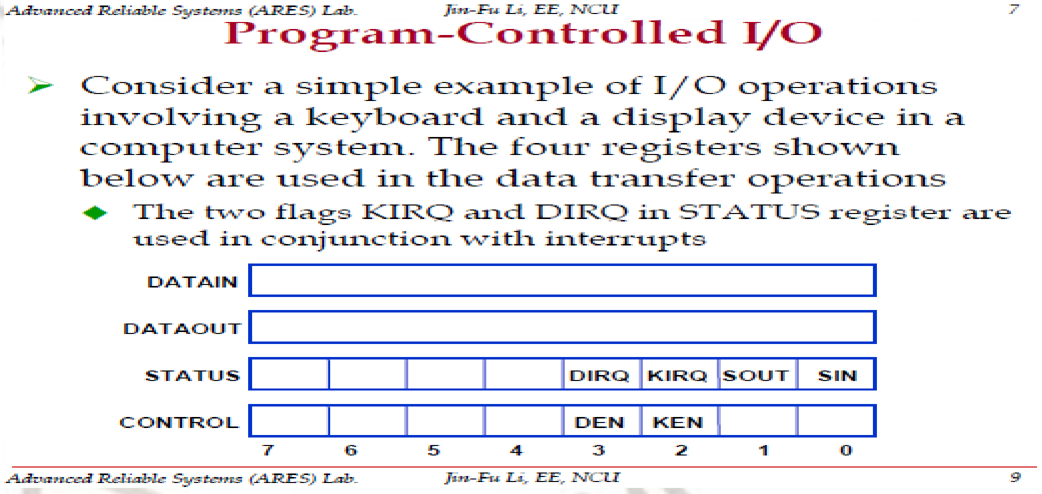
7.9 Memory indirect



I/O Interface for an Input Device

- The address decoder, the data and status registers, and the control circuitry required to coordinate I/O transfers constitute the device's *interface circuit*

Advanced Reliable Systems (ARES) Lab.   Jin-Fu Li, EE, NCU   7

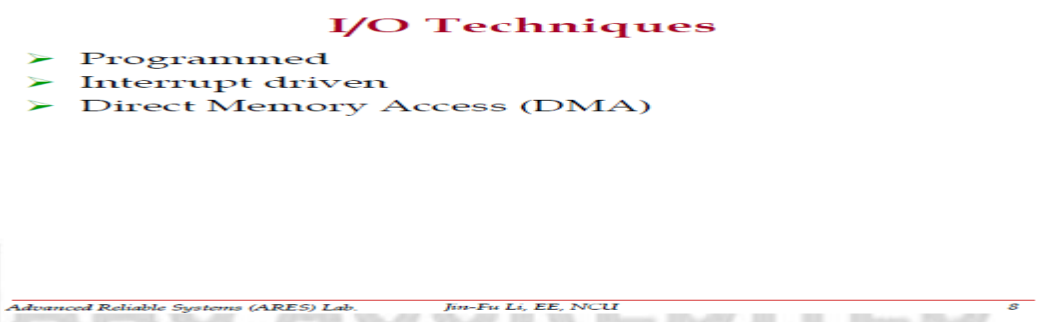


Program-Controlled I/O

- Consider a simple example of I/O operations involving a keyboard and a display device in a computer system. The four registers shown below are used in the data transfer operations
  - The two flags KIRQ and DIRQ in STATUS register are used in conjunction with interrupts

	7	6	5	4	3	2	1	0
DATAIN								
DATAOUT								
STATUS					DIRQ	KIRQ	SOUT	SIN
CONTROL					DEN	KEN		

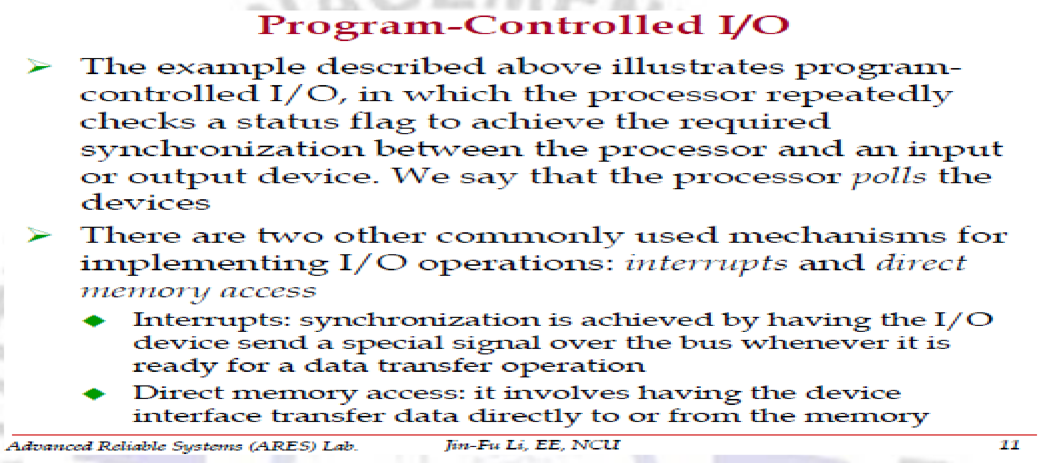
Advanced Reliable Systems (ARES) Lab.   Jin-Fu Li, EE, NCU   9



I/O Techniques

- Programmed
- Interrupt driven
- Direct Memory Access (DMA)

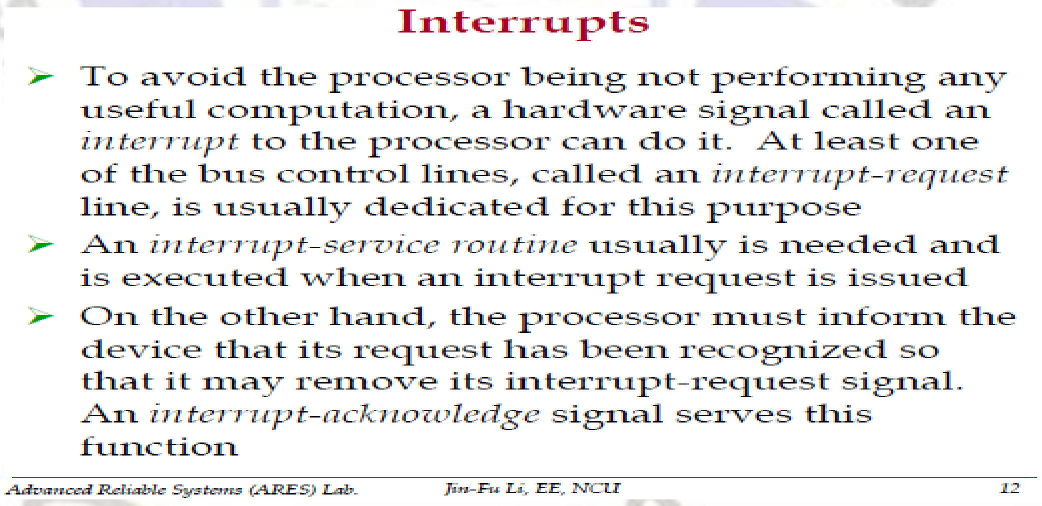
Advanced Reliable Systems (ARES) Lab. Jin-Fu Li, EE, NCU 8



Program-Controlled I/O

- The example described above illustrates program-controlled I/O, in which the processor repeatedly checks a status flag to achieve the required synchronization between the processor and an input or output device. We say that the processor *polls* the devices
- There are two other commonly used mechanisms for implementing I/O operations: *interrupts* and *direct memory access*
  - Interrupts: synchronization is achieved by having the I/O device send a special signal over the bus whenever it is ready for a data transfer operation
  - Direct memory access: it involves having the device interface transfer data directly to or from the memory

Advanced Reliable Systems (ARES) Lab. Jin-Fu Li, EE, NCU 11



Interrupts

- To avoid the processor being not performing any useful computation, a hardware signal called an *interrupt* to the processor can do it. At least one of the bus control lines, called an *interrupt-request* line, is usually dedicated for this purpose
- An *interrupt-service routine* usually is needed and is executed when an interrupt request is issued
- On the other hand, the processor must inform the device that its request has been recognized so that it may remove its interrupt-request signal. An *interrupt-acknowledge* signal serves this function

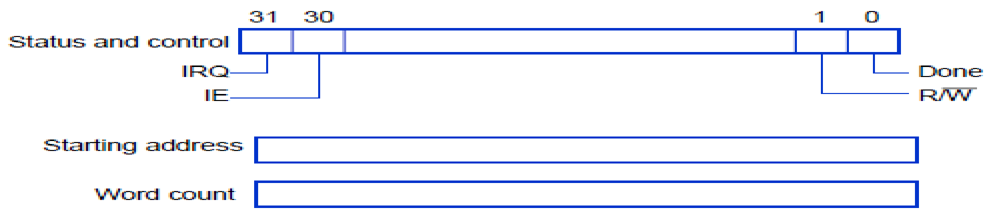
Advanced Reliable Systems (ARES) Lab. Jin-Fu Li, EE, NCU 12

Direct Memory Access

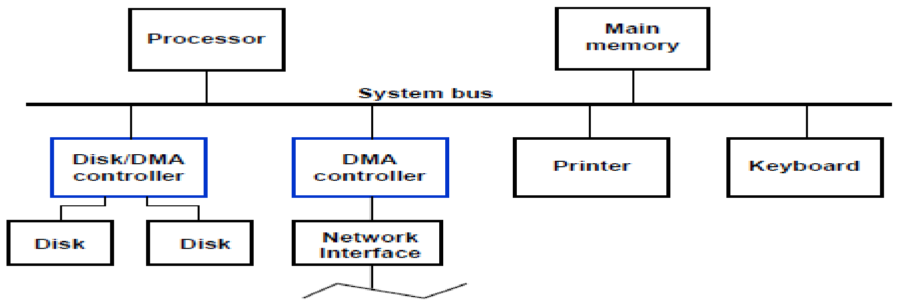
- To transfer large blocks of data at high speed, a special control unit may be provided between an external device and the main memory, without continuous intervention by the processor. This approach is called *direct memory access* (DMA)
- DMA transfers are performed by a control circuit that is part of the I/O device interface. We refer to this circuit as a DMA controller.
- Since it has to transfer blocks of data, the DMA controller must increment the memory address for successive words and keep track of the number of transfers

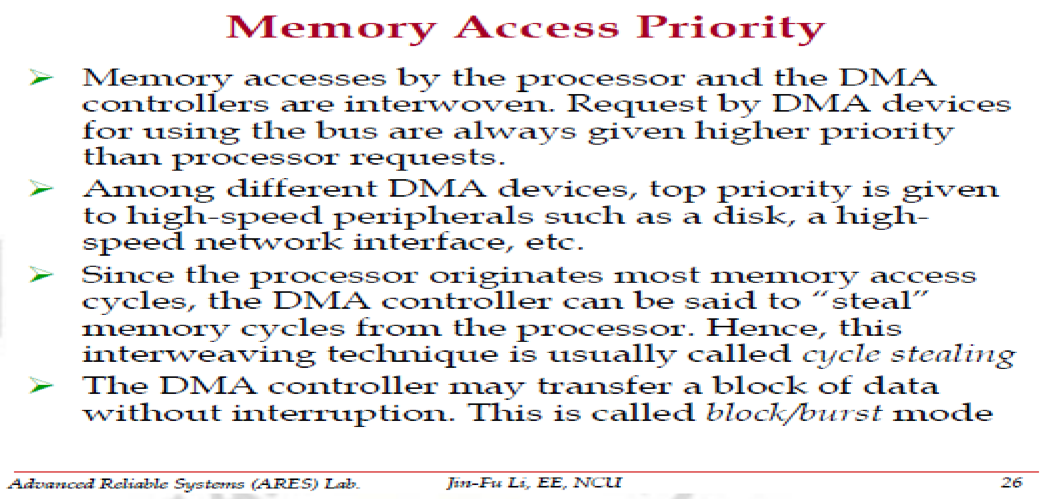
DMA Controller

- Although a DMA controller can transfer data without intervention by the processor, its operation must be under the control of a program executed by the processor
- An example



DMA Controller in a Computer System





Memory Access Priority

- Memory accesses by the processor and the DMA controllers are interwoven. Request by DMA devices for using the bus are always given higher priority than processor requests.
- Among different DMA devices, top priority is given to high-speed peripherals such as a disk, a high-speed network interface, etc.
- Since the processor originates most memory access cycles, the DMA controller can be said to "steal" memory cycles from the processor. Hence, this interweaving technique is usually called *cycle stealing*
- The DMA controller may transfer a block of data without interruption. This is called *block/burst* mode

Advanced Reliable Systems (ARES) Lab. Jin-Fu Li, EE, NCU 26

UNIT 4

MEMORY ORGANIZATION:

MEMORY HIERARCHY

Primary Memory

RAM

ROM

Secondary Memory

Punching Devices

Magnetic Tape

Floppy Disk

Optical Discs (CD/DVD)

Hard Disk Drives

Flash Drives

Through a random web browsing session, when you right click on an image and select 'save image as', have you ever wondered where exactly is it getting saved? How is it getting saved? And why does it not disappear when you turn off your computer? Computer memory is responsible for storing data and applications on a temporary or a permanent basis. It enables a person to retain the information stored on the computer. Without a memory device/arrangement in place, the processor would not be able to find a place which is needed to store the calculations and processes. There are various types of computer memory that can be installed, depending upon the actual need for functioning and specifications of the system.

Computer memory can be primarily classified into two types: Primary Memory and Secondary Memory.

PRIMARY MEMORY/ MAIN MEMORY Primary Memory (also called main memory), is used for immediate access of data by the processor. While primary memory storage demonstrates faster processing ability, it is costly and hence is not largely used for data storage. Most computer systems around the world use primary memory only for bootstrapping and related purposes, and use secondary memory devices for personal data storage purpose. Primary Memory can be divided into two types - Random Access Memory (RAM) and Read Only Memory (ROM). RAM retains its contents as long as the power supply is on. A RAM chip is used as primary memory in most computers today. However, older computers (in the '80s) used ROM devices (floppy disks, magnetic tapes, paper clips or punches; but more prominently floppy disks) as primary memory mechanism.

Random Access Memory (RAM) RAM is a memory scheme within the computer system responsible for storing data on a temporary basis, so that it can be promptly accessed by the processor as and when needed. It is volatile in nature, which means that data will be erased once supply to the storage device is turned off. RAM stores data randomly and the processor accesses these data randomly from the RAM storage. The information stored in the RAM is typically loaded from the computer's hard disk, and includes data related to the operating system and certain applications. When the system is switched off, the RAM loses all stored information. The data remains stored on secondary storage though, and can be retained when the system is running again. Some of the earliest computers used delay-line format of computer storage. Most modern computers use an embedded RAM circuitry on the motherboard which reads data in bursts. So, modern RAM devices are not random memory devices as such; they are burst memory access devices, but the term RAM has stuck in everyday usage.

There are primarily two forms of RAM: Static RAM (SRAM) and Dynamic RAM (DRAM). **Static RAM:** The most expensive of the lot, SRAM uses bistable latching circuitry to store one bit each, and hence is faster than its counterpart. Its high price prevents it from being widely used in everyday computing machines, but many modern machines use SRAM as the processor's cache register.

Dynamic RAM: Widely used in modern computers as primary memory, DRAM is slower than SRAM, but is inexpensive due to its one transistor-one capacitor paired assembly of memory storage.

Back to Index

Read Only Memory (ROM)

Unlike RAM, ROM is a permanent form of storage. ROM stays active regardless of whether powersupply to it is turned on or off. In spite of this, ROM was used (in rare cases is still used) as the primary device for most computers back in the '80s. This was because ROM devices do not allow data stored on them to be modified. As the name itself suggests, data can only be accessed and read by the user, not overwritten, upgraded, or modified. This made it an ideal choice as bootable devices for old computers, programmable interpreters, and portable OS files carrier. The system programs stored on a ROM device could never be altered and hence, stayed secure for use. The ROM memory used in modern computers is pre-programmed by the circuit manufacturer and cannot be altered by the user. The main reason why ROMs are not widely used in modern computer systems is because of the masking and error-retrieval costs. These processes are very expensive, and virtually negate the inexpensive manufacturing involved. Back to Index Cache is a type of RAM which was originally implemented as a temporary storage mechanism to assist redirection to previously manipulated data by the user or the machine. Eventually, the concept of cache has evolved to become a temporary as well as permanent form of storage for the computer, as well as for individual applications. Most individual applications these days maintain their own cache which can be accessed by the processor, as well as the user, and can be maintained for as long as required without any risk of losing the data.

SECONDARY MEMORY/ AUXILIARY MEMORY

Secondary memory is available on mass storage devices for permanent data storage. Data stored on a secondary device is retained even when it is not supplied any power. This data can be transported in most cases, and looks and appears the same on any machine, irrespective of where the data was first copied onto the secondary storage device. Unlike primary memory, secondary memory is not directly accessible by the computer. When a computer needs to run or execute an application stored in secondary memory, it first brings it to primary memory storage for a while, to control and carry out its execution. Once execution of the application is done, the processor releases the application and restores its control and memory data with the secondary memory device. Popular secondary memory devices include hard disk drives, flash drives (pen drives, memory cards etc.), and zip drives.

A couple of decades ago, as the 'personal computer' (PC) revolution was gathering storm, especially in America, floppy disks had acquired almost a cult status amongst PC users. Eventually, floppy disks were phased out by a better technology - a contemporary form of the optical drive called the Compact Disc or CD. CDs came with better speed and larger storage alternatives as compared to floppies. DVDs eventually took over the mantle from CDs, courtesy their ability to store almost 4 times more data. Although DVDs are still widely used, the preferred devices of secondary storage nowadays are portable hard disk drives or flash drives.

Back to Index

Punch Devices

The essential data storage techniques of the '50s and '60s, punch tapes and punch cards have become passé since the advent of newer data storage formats.

Punch Tapes: A 0.1 mm thick paper strip was used to store data in the form of punched holes. A keyboard was used to punch the desired alphabet onto the tape. This alphabet was represented on the tape by a certain number and a select pattern of holes. A separate tape machine was used to send and receive these tapes for distance communication purposes. For computing purposes, stored data on the tapes would be read by the processing unit's inbuilt decoding machine.

Punch Cards: Primarily used in textile and handloom industries, punch cards stored instructions of operation for machines. Early digital computers made punch cards popular as data storage assemblies. Their working is pretty much similar to that of punch tapes, except for the fact that instead of paper strips, this technique uses cards about 3¼ inches × 7⅜ inches in size. Around the 1920s and 1930s, IBM hit upon a series of card innovations which enabled pre-punched data verification cards and cards with the ability to read alphabets, numbers, and signs (symbols) on a single multipurpose card.

Magnetic Tape

Magnetic tape as a recording technique was invented in 1928. This formed the basis for magnetic digital information storage. This form of data storage gained immense popularity in the '70s, when magnetic tapes were wound around 10.6-inch reels. The device used for the read-write operations on these tapes is called a tape drive. Until the early 1980s, magnetic tape drives were huge external devices. With the introduction of IBM's 3480 family of magnetic tape cartridges, most magnetic tape storage assembly went inside the central processing unit.

Transferring data at around 7,200 characters/second, magnetic tapes store data in sequential order, which also can be accessed only in a sequential order. The magnetic tape's storage density and feasibility offered it a ready-made advantage against punched storage techniques. Even through the '90s, as floppy disks and compact discs were taking over the market, magnetic tapes held a fan-like following among large corporations for large-scale data storage. By the turn of the century, as solid state data storage took over, magnetic tapes lost hold with them too.

Floppy Disk

The floppy disk memory technique uses a thin plastic-coated film covered with magnetic material. It is covered with a protective plastic cover. Initially developed by IBM as inexpensive microcode feeders in 1967, floppy disks were made commercially available in 1971 to the public. Floppy disks began as giant 8-inch diskettes, and eventually evolved into 5¼-inch diskettes, and later 3½-inch diskettes. Floppy disks can easily be termed the most popular data storage forms ever, considering they were launched in 1971 and were broadly used right up to the late 2000s. The fact that their availability coincided with the rise of popularity of personal computers among the general public can be attributed as one of the main reasons behind its immense popularity, the others being portability, feasibility, cost-effectiveness, and the lack of better options. As of today, no modern machines are integrated with a floppy disk drive, though their popularity amongst low cost data management companies remains intact.

Optical Drives (CD/DVD)

Philips and Sony collaborated in the '70s on a project to create a new digital audio disc. This collaboration brought together the optical disc drive technologies both the companies were earlier separately working on. Launched in 1982-83, the Compact Disc (CD) eventually went on from being an audio disc to a data storage device. The DVD, (originally Digital Video Disc, but later amended to Digital Versatile Disc) format was based on the CD format, and was developed together by Philips, Sony, Toshiba, and Panasonic around the early '90s. It was launched in 1995 and became an instant success by the virtue of being same size as a CD yet offering almost 4 times its memory space. While data storage isn't forfended, DVDs are mostly used for audio and video recording/storage/playback purposes. In the late '90s, the popularity of CDs took a major three-way hit. While the launch of DVD had already put it out of favor with video enthusiasts, its audio and data storage purposes also waned in lieu of advancing technology. Affordable portable hard disk drives and flash drives drove CDs out as a preferred form of data storage.

On the other hand, the easy availability of MP3 players and the legendary rise of Apple's iPod, practically drove audio CDs out of the market. The DVD too has found successors in the form of HD DVD and Blu-ray discs, and is in the gradual process of being phased out from regular use.

Hard Disk Drives

The dominant technique for storing data in current times, a hard disk consists of rapidly rotating discs with a magnetic head to read and write data. Data can be accessed randomly. HDDs were introduced by IBM (Yes, again IBM!) around the late 1950s for real-time transaction processing machines. A few years later, IBM commercially launched the IBM 1311 model, which was almost as big as a dishwasher, and had the capacity to store about 2 million characters. Eventually, hard disk drives began to shrink in size and increase in storage capacity. Hard disk drives were sold to PC and Mac users in the '80s as an external device with a SCSI port on the back of the machines. A series of innovation on part of industry leaders through the '80s and early '90s led to the hard disk being integrated inside the CPU. A typical desktop hard disk is 3.5 inches in size, while for laptops it is 2.5 inches. As portability and convenience became keywords in modern times, hard disks have again become portable. External hard disks typically use the USB plug-and-play mechanism and are very cost-effective. Modern external hard disks can hold up to 2 terabytes of data. Hard disks seem to be in for the long haul. No better option is in sight for now, and one can easily predict that hard disks should continue to rule as the preferred form of data storage in the years to come.

Flash Drives

A flash drive is a data storage device that uses flash memory for storage purposes. Typical in design, flash drives are light-weight and small in design; and are hence easily portable. Flash drives operate from the power supplied by a computer's USB port (the port in which they are plugged in). The data on it can be erased and re-programmed as per the user's requirements. It only has a specific number of erase and write cycles that it can withstand, after which it creates a tendency to lose out on the stored information. Memory cards and USB flash drives are some
modes of this type of memory storage. Low cost, minimal power consumption, and portable features make flash drives extremely desirable and popular in modern times. The concept of computer memory has evolved since the first electronic computer (ENIAC) was set up in 1946 with a primitive read only pre-stored programming mechanism. ENIAC used function tables for storing instructions. Its maximum storage capacity was 600 two-hundred digit decimal instructions.

The way data is stored today and the volumes in which it can be stored today is like a million miles ahead of that. Memory management has become an important concept in every computer programmer's textbook. Corporations and computer scientists keep researching for newer, simpler, easier, and cost-effective methods of memory storage that can hold larger and larger capacity of data than what is currently possible. Computer memory and its evolution is a constant process, much like the rest of technology. It has changed multifold over the last few decades; expect it to change multifold in the decades to come.

CACHE MEMORY

Cache is a small high-speed memory. Stores data from some frequently used addresses (of main memory). Cache hit Data found in cache. Results in data transfer at maximum speed. Cache miss Data not found in cache. Processor loads data from M and copies into cache. This results in extra delay, called miss penalty. Hit ratio = percentage of memory accesses satisfied by the cache. Miss ratio = 1-hit ratio

Cache is partitioned into lines (also called blocks). Each line has 4-64 bytes in it. During data transfer, a whole line is read or written. Each line has a tag that indicates the address in M from which the line has been copied. Cache hit is detected through an associative search of all the tags. Associative search provides a fast response to the query: “Does this key match with any of the tags?” Data is read only if a match is found.

Types of Cache

Fully Associative

Direct Mapped

Set Associative

Fully Associative Cache tag data M-address

Associative search of tags is expensive. Feasible for very small size caches only.

The secret of success Program locality.

Cache line replacement

To fetch a new line after a miss, an existing line must be replaced. Two common policies for identifying the victim block are

LRU (Least Recently Used)

Random

Estimating Average Memory Access Time

Average memory access time = Hit time + Miss rate x Miss penalty

Assume that

Hit time = 5 ns

Miss rate = 10%

Miss penalty = 100 ns.

The average memory access time = 15 ns.

Better performance at a cheaper price.

Direct-Mapped Cache

A given memory block can be mapped into one and only cache line.

Advantage

No need of expensive associative search!

Disadvantage

Miss rate may go up due to possible increase of mapping conflicts.

ASSCIATIVE MEMORY

The replacement policy decides where in the cache a copy of a particular entry of main memory will go. If the replacement policy is free to choose any entry in the cache to hold the copy, the cache is called **fully associative**. At the other extreme, if each entry in main memory can go in just one place in the cache, the cache is **direct mapped**. Many caches implement a compromise in which each entry in main memory can go to any one of N places in the cache, and are described as N-way set associative. For example, the level-1 data cache in an AMD Athlon is 2-way set associative, which means that any particular location in main memory can be cached in either of 2 locations in the level-1 data cache.

Associativity is a trade-off. If there are ten places to which the replacement policy could have mapped a memory location, then to check if that location is in the cache, ten cache entries must be searched. Checking more places takes more power, chip area, and potentially time. On the other hand, caches with more associativity suffer fewer misses (see conflict misses, below), so that the CPU wastes less time reading from the slow main memory. The rule of thumb is that doubling the associativity, from direct mapped to 2-way, or from 2-way to 4-way, has about the same effect on hit rate as doubling the cache size. Associativity increases beyond 4-way have much less effect on the hit rate and are generally done for other reasons (see virtual aliasing, below).

In order of worse but simple to better but complex:
direct mapped cache — The best (fastest) hit times, and so the best tradeoff for "large" caches
2-way set associative cache
2-way skewed associative cache – In 1993, this was the best tradeoff for caches whose sizes were in the range 4K-8K bytes.
4-way set associative cache
fully associative cache – the best (lowest) miss rates, and so the best tradeoff when the miss penalty is very high

Direct-mapped cache

Here each location in main memory can only go in one entry in the cache. It doesn't have a replacement policy as such, since there is no choice of which cache entry's contents to evict. This means that if two locations map to the same entry, they may continually knock each other out. Although simpler, a direct-mapped cache needs to be much larger than an associative one to give comparable performance, and is more unpredictable. Let 'x' be block number in cache, 'y' be block number of memory, and 'n' be number of blocks in cache, then mapping is done with the help of the equation x=y mod n.

2-way set associative cache

If each location in main memory can be cached in either of two locations in the cache, one logical question is: *which one of the two?* The simplest and most commonly used scheme, shown in the right-hand diagram above, is to use the least significant bits of the memory location's index as the index for the cache memory, and to have two entries for each index. One benefit of this scheme is that the tags stored in the cache do not have to include that part of the main memory address which is implied by the cache memory's index. Since the cache tags have fewer bits, they take less area on the microprocessor chip and can be read and compared faster. Also LRU is especially simple since only one bit needs to be stored for each pair.

Speculative execution

One of the advantages of a direct mapped cache is that it allows simple and fast speculation. Once the address has been computed, the one cache index which might have a copy of that location in memory is known. That cache entry can be read, and the processor can continue to work with that data before it finishes checking that the tag actually matches the requested address. The idea of having the processor use the cached data before the tag match completes can be applied to associative caches as well. A subset of the tag, called a *hint*, can be used to pick just one of the possible cache entries mapping to the requested address.

The entry selected by the hint can then be used in parallel with checking the full tag. The hint technique works best when used in the context of address translation, as explained below.

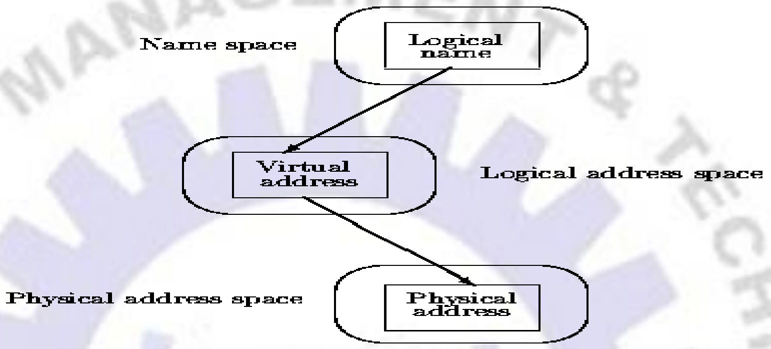
2-way skewed associative cache

Other schemes have been suggested, such as the *skewed cache*, where the index for way 0 is direct, as above, but the index for way 1 is formed with a hash function. A good hash function has the property that addresses which conflict with the direct mapping tend not to conflict when mapped with the hash function, and so it is less likely that a program will suffer from an unexpectedly large number of conflict misses due to a pathological access pattern. The downside is extra latency from computing the hash function. Additionally, when it comes time to load a new line and evict an old line, it may be difficult to determine which existing line was least recently used, because the new line conflicts with data at different indexes in each way; LRU tracking for non-skewed caches is usually done on a per-set basis. Nevertheless, skewed-associative caches have major advantages over conventional set-associative ones.

VIRTUAL MEMORY AND VIRTUAL MEMORY MANAGEMENT

Because main memory (*i.e.*, transistor memory) is much more expensive, per bit than disk memory (presently, approximately 10 to 50 times more expensive), it is usually economical to provide most of the memory requirements of a computer system as disk memory. Disk memory is also ``permanent'' and not (very) susceptible to such things as power failure. Data, and executable programs, are brought into memory, or *swapped* as they are needed by the CPU in much the same way as instructions and data are brought into the cache. Most large systems today implement this ``memory management'' using a hardware memory controller in combination with the operating system software. One of the most primitive forms of ``memory management'' is often implemented on systems with a small amount of main memory. This method leaves the responsibility of memory management entirely to the programmer; if a program requires more memory than is available, the program must be broken up into separate, independent sections and one ``overlaid'' on top of another when that particular section is to be executed. This type of memory management, which is completely under the control of the programmer, is sometimes the only type of memory management available for small microcomputer systems. Modern memory management schemes, usually implemented in mini - to mainframe computers, employ an automatic, user transparent scheme, usually called ``virtual memory''. In a computer system which supports virtual memory management, the computer appears to the programmer to have its address space limited only by the addressing range of the computer, not by the amount of memory which is physically connected to the computer as main memory.

In fact, *each process* appears to have available the full memory resources of the system. Processes can occupy the same virtual memory but be mapped into completely different physical memory locations. Of course, the parts of a program and data which are actually being executed must lie in main memory and there must be some way in which the ``virtual address'' is translated into the actual physical address in which the instructions and data are placed in main memory. The process of translating, or mapping, a virtual address into a physical address is called *virtual address translation*. Figure shows the relationship between a named variable and its physical location in the system.



This mapping can be accomplished in ways similar to those discussed for mapping main memory into the cache memory. In the case of virtual address mapping, however, the relative speed of main memory to disk memory (a factor of approximately 10,000 to 100,000) means that the cost of a ``miss'' in main memory is very high compared to a cache miss, so more elaborate replacement algorithms may be worthwhile.) In fact, in most processors, a direct mapping scheme is supported by the system hardware, in which a *page map* is maintained in physical memory. This means that each physical memory reference requires *both* an access to the page table *and* and an operand fetch. In effect, all memory references are indirect. Figure shows a typical virtual-to-physical address mapping.

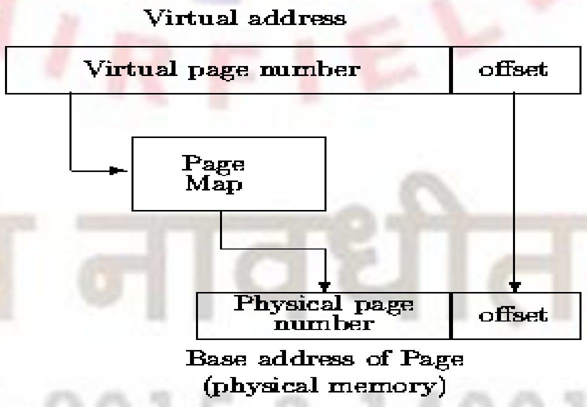


Figure: A direct mapped virtual to physical address translation

This requirement would be a considerable performance penalty, so most systems which support virtual addressing have a small associative memory (called a*translation lookaside buffer*, or TLB) which contains the last few virtual addresses and their corresponding physical addresses, so in most cases the virtual to physical mapping does not require an additional memory access. Figure shows a typical virtual-to-physical address mapping in a system containing a TLB.

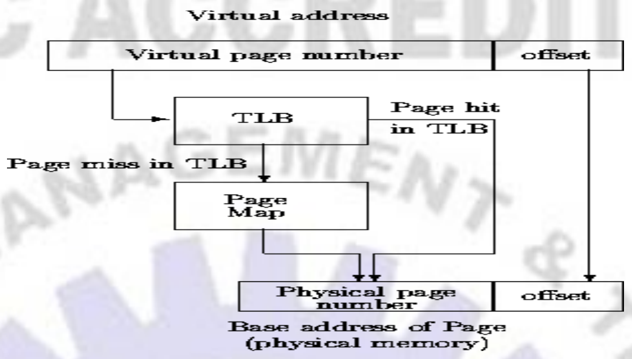


Figure: A virtual to physical address translation mechanism with a TLB

For many current architectures, including the VAX, INTEL 80486, and MIPS, addresses are 32 bits, so the virtual address space is 2^{32} bytes, or 4 G bytes. A physical memory of about 16-64 M bytes is typical for these machines, so the virtual address translation must map the 32 bits of the virtual memory address into a corresponding area of physical memory. Sections of programs and data not currently being executed normally are stored in disk, and are brought into main memory as necessary. If a virtual memory reference occurs to a location not currently in physical memory, the execution of that instruction is aborted, and can be restored again when the required information is placed in main memory from the disk by the memory controller. (Note that, when the instruction is aborted, the processor must be left in the same state it would have been had the instruction not been executed at all). While the memory controller is fetching the required information from disk, the processor can be executing another program, so the actual time required to find the information on the disk (the disk seek time) is not wasted by the processor. In this sense, the disk seek time usually imposes little (time) overhead on the computation, but the time required to actually place the information in memory may impact the time the user must wait for a result. If many disk seeks are required in a short time, however, the processor may have to wait for information from the disk.

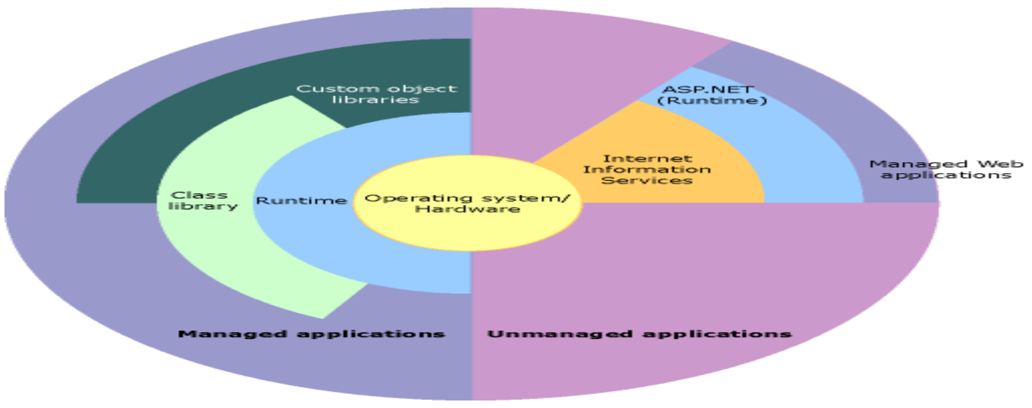
Normally, blocks of information are taken from the disk and placed in the memory of the processor. The two most common ways of determining the sizes of the blocks to be moved into and out of memory are called *segmentation* and *paging*, and the term *segmented memory management* or *paged memory management*refer to memory management systems in which the blocks in memory are *segments* or *pages*.

Overview of the .NET Framework

The .NET Framework is a technology that supports building and running the next generation of applications and XML Web services. The .NET Framework is designed to fulfill the following objectives:

- To provide a consistent object-oriented programming environment whether object code is stored and executed locally, executed locally but Internet-distributed, or executed remotely.
- To provide a code-execution environment that minimizes software deployment and versioning conflicts.
- To provide a code-execution environment that promotes safe execution of code, including code created by an unknown or semi-trusted third party.
- To provide a code-execution environment that eliminates the performance problems of scripted or interpreted environments.
- To make the developer experience consistent across widely varying types of applications, such as Windows-based applications and Web-based applications.
- To build all communication on industry standards to ensure that code based on the .NET Framework can integrate with any other code.
- The .NET Framework consists of the common language runtime and the .NET Framework class library. The common language runtime is the foundation of the .NET Framework. You can think of the runtime as an agent that manages code at execution time, providing core services such as memory management, thread management, and remoting, while also enforcing strict type safety and other forms of code accuracy that promote security and robustness. In fact, the concept of code management is a fundamental principle of the runtime. Code that targets the runtime is known as managed code, while code that does not target the runtime is known as unmanaged code. The class library is a comprehensive, object-oriented collection of reusable types that you can use to develop applications ranging from traditional command-line or graphical user interface (GUI) applications to applications based on the latest innovations provided by ASP.NET, such as Web Forms and XML Web services.
- The .NET Framework can be hosted by unmanaged components that load the common language runtime into their processes and initiate the execution of managed code, thereby creating a software environment that can exploit both managed and unmanaged features. The .NET Framework not only provides several runtime hosts, but also supports the development of third-party runtime hosts.

- For example, ASP.NET hosts the runtime to provide a scalable, server-side environment for managed code. ASP.NET works directly with the runtime to enable ASP.NET applications and XML Web services, both of which are discussed later in this topic.
- Internet Explorer is an example of an unmanaged application that hosts the runtime (in the form of a MIME type extension). Using Internet Explorer to host the runtime enables you to embed managed components or Windows Forms controls in HTML documents. Hosting the runtime in this way makes managed mobile code possible, but with significant improvements that only managed code can offer, such as semi-trusted execution and isolated file storage.
- The following illustration shows the relationship of the common language runtime and the class library to your applications and to the overall system. The illustration also shows how managed code operates within a larger architecture.



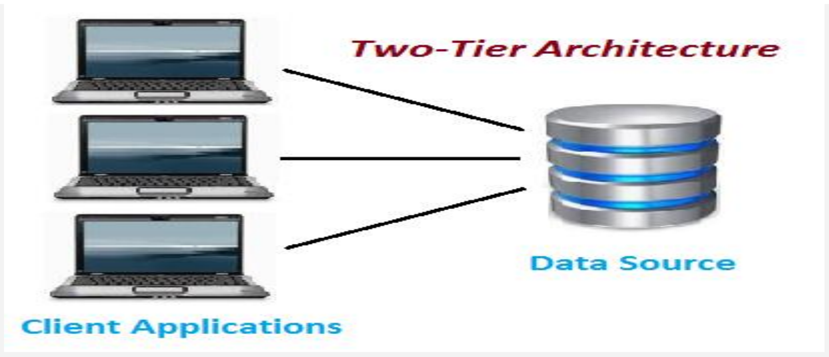
.NET Framework in context.

Two-Tier and Three-Tier Architecture

All projects are broadly divided into two types of applications **2 tier and 3 tier architecture**. Basically high level we can say that *2-tier architecture* is Client server application and *3-tier architecture* is Web based application. Below I am concentrating on the difference between Two-Tier and Three-Tier Architecture, what all advantages, disadvantages and practical examples.

Two-Tier Architecture:

The two-tier is based on Client Server architecture. The two-tier architecture is like client server application. The direct communication takes place between client and server. There is no intermediate between client and server. Because of tight coupling a 2 tiered application will run faster.



Two-Tier Architecture

The above figure shows the architecture of two-tier. Here the direct communication between client and server, there is no intermediate between client and server.

Let’s take a look of real life example of Railway Reservation two-tier architecture:

Let’s consider that first Person is making Railway Reservation for Mumbai to Delhi by Mumbai Express at Counter No. 1 and at same time second Person is also try to make Railway reservation of Mumbai to Delhi from Counter No. 2

If staff from Counter No. 1 is searching for availability into system & at the same staff from Counter No. 2 is also looking for availability of ticket for same day then in this case there is might be good change of confusion and chaos occurs. There might be chance of lock the Railway reservation that reserves the first.

But reservations can be making anywhere from the India, then how it is handled?

So here if there is difference of micro seconds for making reservation by staff from Counter No. 1 & 2 then second request is added into queue. So in this case the Staff is entering data to Client Application and reservation request is sent to the database. The database sends back the information/data to the client.

In this application the Staff user is an end user who is using Railway reservation application software. He gives inputs to the application software and it sends requests to Server. So here both Database and Server are incorporated with each other, so this technology is called as

“***Client-Server Technology***“.

The Two-tier architecture is divided into two parts:

1) Client Application (Client Tier)

2) Database (Data Tier)

On client application side the code is written for saving the data in the SQL server database. Client sends the request to server and it process the request & send back with data. The main problem of two tier architecture is the server cannot respond multiple request same time, as a result it cause a data integrity issue.

Advantages:

1. Easy to maintain and modification is bit easy
2. Communication is faster

Disadvantages:

1. In two tier architecture application performance will be degrade upon increasing the users.
2. Cost-ineffective

Three-Tier Architecture:

Three-tier architecture typically comprise a presentation tier, a business or data access tier, and a data tier. Three layers in the three tier architecture are as follows:

1) Client layer

2) Business layer

3) Data layer

1) Client layer:

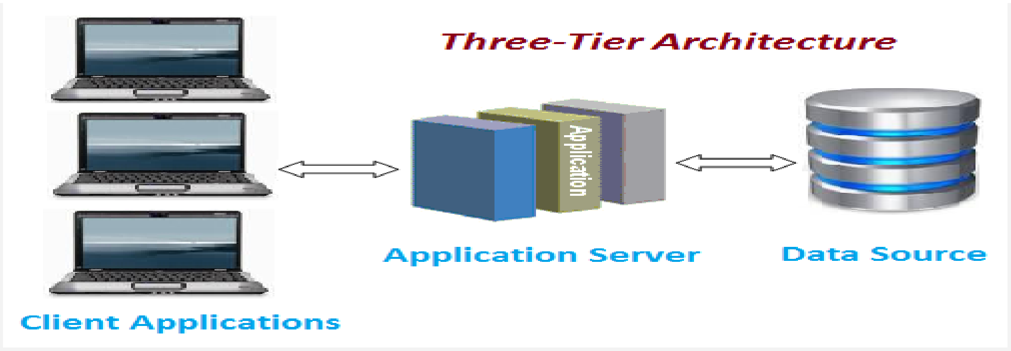
It is also called as *Presentation layer* which contains UI part of our application. This layer is used for the design purpose where data is presented to the user or input is taken from the user. For example designing registration form which contains text box, label, button etc.

2) Business layer:

In this layer all business logic written like validation of data, calculations, data insertion etc. This acts as a interface between Client layer and Data Access Layer. This layer is also called the intermediary layer helps to make communication faster between client and data layer.

3) Data layer:

In this layer actual database is comes in the picture. Data Access Layer contains methods to connect with database and to perform insert, update, delete, get data from database based on our input data.



Three-tier Architecture

Advantages

1. High performance, lightweight persistent objects
2. Scalability – Each tier can scale horizontally
3. Performance – Because the Presentation tier can cache requests, network utilization is minimized, and the load is reduced on the Application and Data tiers.
4. High degree of flexibility in deployment platform and configuration
5. Better Re-use
6. Improve Data Integrity
7. Improved Security – Client is not direct access to database.
8. Easy to maintain and modification is bit easy, won't affect other modules
9. In three tier architecture application performance is good.

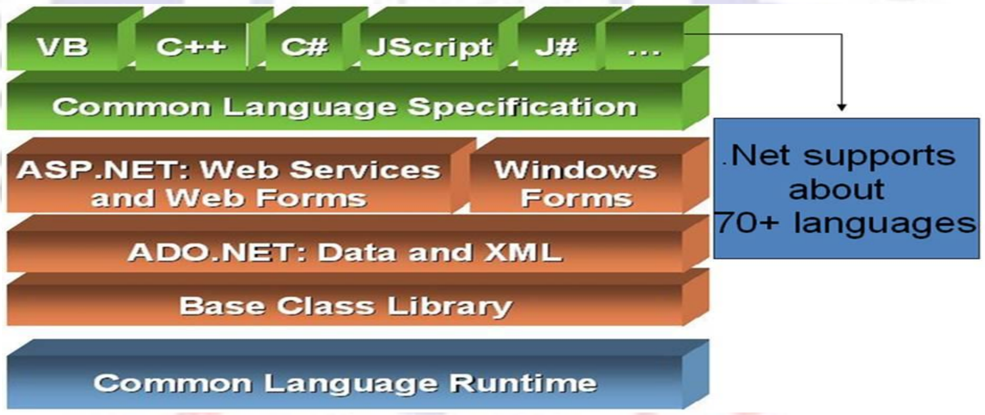
Disadvantages

1. Increase Complexity/Effort

.Net Architecture and .Net Framework basics:

Components of .Net Framework

Components of .Net Framework


VB
C++
C#
JScript
J#
...
Common Language Specification
ASP.NET: Web Services and Web Forms
Windows Forms
ADO.NET: Data and XML
Base Class Library
Common Language Runtime
.Net supports about 70+ languages

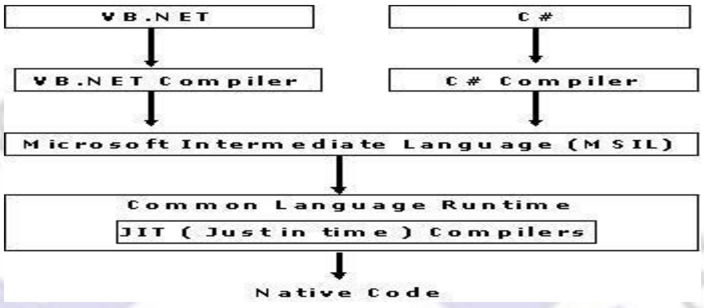
Net Framework is a platform that provides tools and technologies to develop Windows, Web and Enterprise applications. It mainly contains two components,

1. Common Language Runtime (CLR)
2. .Net Framework Class Library.

1. Common Language Runtime (CLR)

.Net Framework provides runtime environment called Common Language Runtime (CLR).It provides an environment to run all the .Net Programs. The code which runs under the CLR is called as Managed Code.

Programmers need not to worry on managing the memory if the programs are running under the CLR as it provides memory management and thread management. Programmatically, when our program needs memory, CLR allocates the memory for scope and de-allocates the memory if the scope is completed.
Language Compilers (e.g. C#, VB.Net, J#) will convert the Code/Program to Microsoft Intermediate Language (MSIL) intern this will be converted to Native Code by CLR. See the below Fig.


VB.NET
C#
VB.NET Compiler
C# Compiler
Microsoft Intermediate Language (MSIL)
Common Language Runtime
JIT (Just in time) Compilers
Native Code

There are currently over 15 language compilers being built by Microsoft and other companies also producing the code that will execute under CLR.

2. .Net Framework Class Library (FCL)

This is also called as Base Class Library and it is common for all types of applications i.e. the way you access the Library Classes and Methods in VB.NET will be the same in C#, and it is common for all other languages in .NET.

The following are different types of applications that can make use of .net class library.

1. Windows Application.
2. Console Application
3. Web Application.
4. XML Web Services.
5. Windows Services.

In short, developers just need to import the BCL in their language code and use its predefined methods and properties to implement common and complex functions like reading and writing to file, graphic rendering, database interaction, and XML document manipulation.

Below are the few more concepts that we need to know and understand as part of this .Net framework.

3. Common Type System (CTS)

It describes set of data types that can be used in different .Net languages in common. (i.e), CTS ensures that objects written in different .Net languages can interact with each other.

For Communicating between programs written in any .NET complaint language, the types have to be compatible on the basic level.

The common type system supports two general categories of types:

Value types:

Value types directly contain their data, and instances of value types are either allocated on the stack or allocated inline in a structure. Value types can be built-in (implemented by the runtime), user-defined, or enumerations.

Reference types:

Reference types store a reference to the value's memory address, and are allocated on the heap. Reference types can be self-describing types, pointer types, or interface types. The type of a reference type can be determined from values of self-describing types. Self-describing types are further split into arrays and class types. The class types are user-defined classes, boxed value types, and delegates.

4. Common Language Specification (CLS)

It is a sub set of CTS and it specifies a set of rules that needs to be adhered or satisfied by all language compilers targeting CLR. It helps in cross language inheritance and cross language debugging.

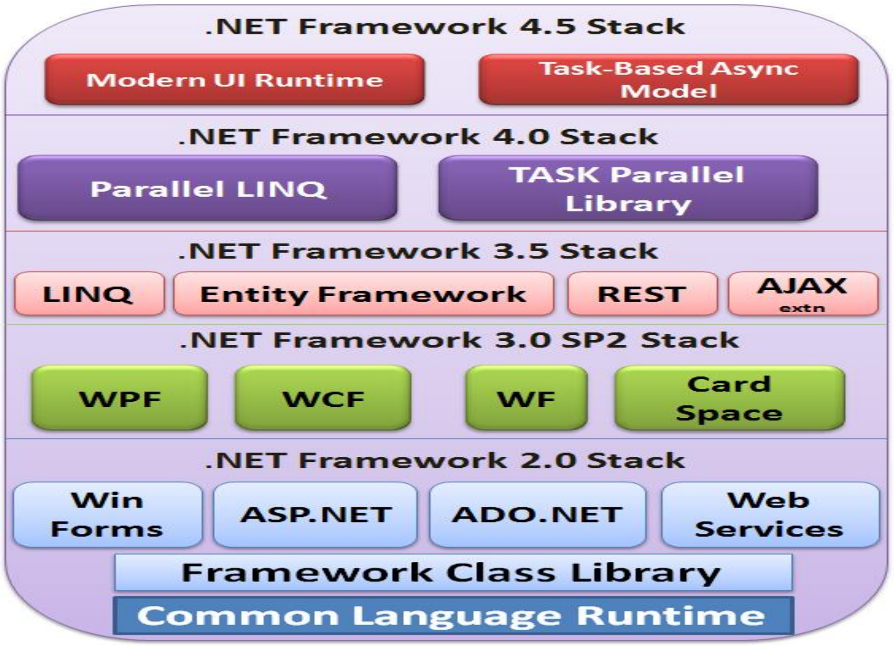
Common language specification Rules:

It describes the minimal and complete set of features to produce code that can be hosted by CLR. It ensures that products of compilers will work properly in .NET environment.

Sample Rules:

1. Representation of text strings
2. Internal representation of enumerations
3. Definition of static members and this is a subset of the CTS which all .NET languages are expected to support.
4. Microsoft has defined CLS which are nothing but guidelines that language to follow so that it can communicate with other .NET languages in a seamless manner.

Below mentioned the .Net Architecture stack for easy understanding.



Features of .Net

- Features of .NET Platform are :-
- **Common Language Runtime**
- Explains the features and benefits of the common language runtime, a run-time environment that manages the execution of code and provides services that simplify the development process.
- **Assemblies**
- Defines the concept of assemblies, which are collections of types and resources that form logical units of functionality. Assemblies are the fundamental units of deployment, version control, reuse, activation scoping, and security permissions.
- **Application Domains**
- Explains how to use application domains to provide isolation between applications.
- **Runtime Hosts**
- Describes the runtime hosts supported by the .NET Framework, including ASP.NET, Internet Explorer, and shell executables.
- **Common Type System**
- Identifies the types supported by the common language runtime.
- **Metadata and Self-Describing Components**
- Explains how the .NET Framework simplifies component interoperation by allowing compilers to emit additional declarative information, or metadata, into all modules and assemblies.
- **Cross-Language Interoperability**
- Explains how managed objects created in different programming languages can interact with one another.
- **.NET Framework Security**
- Describes mechanisms for protecting resources and code from unauthorized code and unauthorized users.
- **NET Framework Class Library**
- Introduces the library of types provided by the .NET Framework, which expedites and optimizes the development process and gives you access to system functionality.

.NET and Its Advantages / Disadvantages

Developers, for a long time, have been looking for a framework that can help them easily create an application. dotNET (also known as .NET) serves that purpose.

It is a framework, developed by Microsoft, which can be used to support the development as well as running / maintenance of modern day apps and XML web services. The framework strives to offer a highly consistent object oriented programming environment. It can be used to create apps that run on multiple platforms. In short, these are desktop, mobile, as well as web apps that run on Windows based devices, PCs, and servers. An interesting aspect of this framework is that it supports the creation of cross platform server apps that can smoothly run across server platforms as Linux, Windows, and even MAC. This framework can be used to design, develop, compile, build, and deploy an application with its broad range of support programs, code libraries, compilers, and APIs. These varied components assist in the development of a customized project/solution. .NET has been used by a large number of developers for the creation of apps and services on multiple devices and OS. One aspect of .NET that stands out is the range of features that it offers. In addition to that, programmers get access to a code execution environment to minimize conflicts during the deployment and versioning of the software. Another important characteristic that needs to be spoken about here is its consistent developer experience while building a range of different apps.

What is .NET used for?

Now we know that .NET is feature rich and it can be used for a wide variety of next generation apps. Let's now try to be more specific and delve into the range of apps that can be built with the use of this framework.

- **Business Functions**
- Those days are gone when businesses were conducted without the presence of a software tool in the background. Businesses, today, make constructive use of applications and software to streamline their business processes. Be it supply chain management, sales, or finance; business software can work wonders. .NET is regularly and extensively used in the creation of such software like CRM, supply management app, and more.
- **Re-Designing**
- An organization doesn't remain static. Your needs can alter with them, and your software needs to scale with it. .NET has been found to be a great platform for re-designing current applications in order to make it line up with the growing needs of an organization.

- **Interoperable Apps**
- If one aspires to have an interoperable app that seamlessly brings together user experience across multiple platforms, .NET can again play a pivotal role. With the range of features on offer, .NET can help programmers create apps that work in sync across platforms.
- **Gaming**
- .NET is extremely versatile, and that makes it suitable for the development of gaming applications. Its versatility can also help in the creation of web and enterprise-graded applications.
- **Communication**
- Businesses need emails and chat platforms for seamless communication between employees. .NET is one of the most popular platforms used for emails & chat. It can integrate with your Outlook for a more holistic experience. Moreover, there are various other techniques that make your communication more fool proof.
- **Multi-Tiered Software Architecture**
- .NET makes use of multi-tiered software architecture. It is known as multi-tiered because it physically separates functions for presentation, app processing, and data management. It helps developers build flexible applications. Moreover, developers can also add or edit a layer without being required to rework on the entire app.
- **Cross Platform**
- Most expert developers consider .NET to be a modular and swift platform that can be used to create server apps that run equally well across Linux, Windows, and MAC.
- **Mobile Apps**
- It lets you create apps that work smoothly across computer, mobile, and other devices.

Advantages of .NET

Here are some of the compelling advantages of .NET:

- **Object Oriented**
- Everything that you see in the .NET framework is an object. It is the same for what you write within the framework. This means that you get a powerful tool to not just access but also control your apps. This also makes it simpler for you to respond to recurring events.
- **Caching**
- The caching system that .NET includes is extremely robust and easy-to-use.

- **Easy Maintenance**
- Pages, with .NET, are extremely simple to write and maintain. This is because the source code and HTML are both together. In addition to that, the source code executes on the server. What does this mean? This makes your web pages more powerful and flexible.
- **Time-Saving**
- Time is money, and .NET helps you save a lot of that. The way it is developed, .NET removes a large part of the coding requirement. This means that the developers save time, and the app's time-to-market can be shortened considerably.
- **Simplicity**
- Performing common tasks with .NET is extremely simple and straight forward. Submission of forms is a breeze and so is site configuration, deployment, and client authentication.
- **Feature-Rich**
- There are a range of features that can be explored by the developers in order to create powerful apps. Consider the case of its rich toolbox as also the designer in the visual studio. They let you access such features as automatic deployment, WYSIWYG editing, and drag-and-drop controls.
- **Consistency**
- The management and monitoring of all the processes is performed by the framework. If one of the processes is dead, a new process can be created just as easily. This lets your app be consistently available for handling requests.
- **Monitoring**
- Finally, .NET also stands for its automatic monitoring. It will promptly notice any problems like infinite loops, memory leaks, etc. Not just this, it will also destroy these activities automatically and restart itself.
- In conclusion, there are tons of advantages that .NET offers. These features make it popular among clients as well as programmers.

Disadvantages of .NET

Along with advantages, you will always come across disadvantages of any platform. That's the case with .NET too.

Here's a compilation of some of the drawbacks of using .NET:

- **Limited Object Relational (OR) support**
- It is found to be limited at times, because such support is generally available with entity framework only.

- **Slower than Native Code**
- Managed code that you run with .NET can be slower than native code.
- **Vendor lock-in**
- The framework involves Vendor lock-in. This can mean that future development will be only dependent on Microsoft.
- **Expensive**
- In some cases, migration of apps to .NET can turn out to be expensive.

Garbage Collection: Memory Management in .Net

Memory management using the Garbage Collector

The Garbage Collector is an important process in the .Net CLR. The Microsoft .NET common language runtime requires that all resources be allocated from the managed heap. Objects are automatically freed when they are no longer needed by the application. The CLR performs Non-deterministic finalization: i.e. it cannot be predicted, when the Garbage Collector will be called to clean up unnecessary objects from memory. Non deterministic finalization is performed when the CLR foresees a memory deficiency. In all kinds of programs, resources are used. These resources could be in the form of 'memory buffers', 'screen space', 'network connections', 'database resources', and so on. Moreover, we should realize that in an object-oriented environment, every type is actually a resource available for our program to use. To use any of these resources requires that memory be allocated to represent the type. The bare minimum, steps required to access a resource are as follows:

1. Allocate memory for the resource. (Resource Allocation - explained below)
2. Initialize the memory to set the initial state of the resource.
3. Use the resource by accessing the instance members of the type.
4. Reset the state of the resource for cleanup.
5. Free the memory. (Garbage Collection - explained below)

Resource Allocation

There are two data types that the Garbage collector bifurcates into: Value types and Reference types

1. **<u>Value types</u>**

In C#, all the "things" declared with the following list of type declarations are Value types (because they are from System.ValueType):

- bool
- byte
- char

- decimal
- double
- enum
- float
- int
- long
- sbyte
- short
- struct
- uint
- ulong
- ushort

These Value Types reside on the stack. The CLR does not care about this type of data with reference to memory management. The Stack is self-maintaining, meaning that it basically takes care of its own memory management. When the top box is no longer used, it's thrown out.

2. **Reference Types**

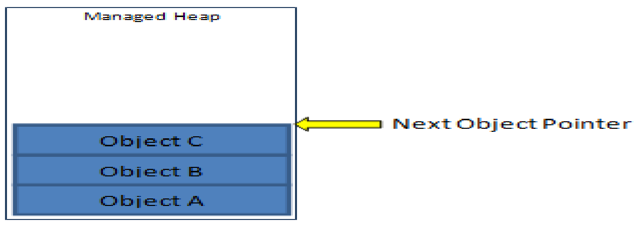
All the "things" declared with the types in this list are Reference types. They inherit from System.Object

- class
- interface
- delegate

These Reference Types exist on a managed heap. This is where the Garbage Collector comes into the picture.

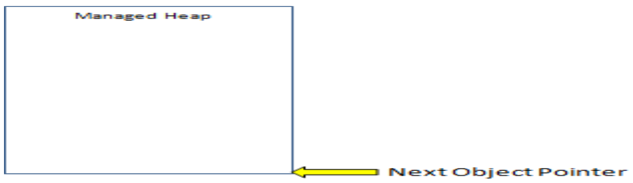
Managed Heap

A Managed Heap is a pool of memory where all the instances of Reference Types (classes and arrays etc.) are created and allocated space. When an object is initialized, the CLR simply allocates an address space in the Managed Heap. There is actually no storage space allocated at this point of time. However a pointer to this heap is maintained to keep track of the next object within this heap. The memory space on this heap is allocated only when an object is created using the new operator.

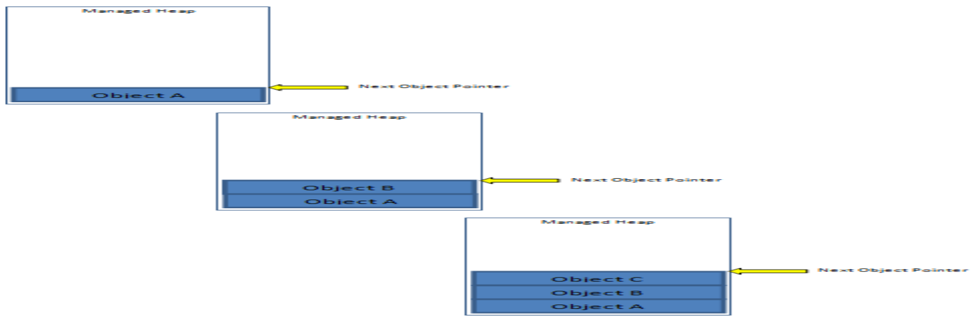


Mechanics of Resource Allocation - Object creation, initialization and instantiation.

Initially, the next object pointer is set to the base address of the reserved address space region.

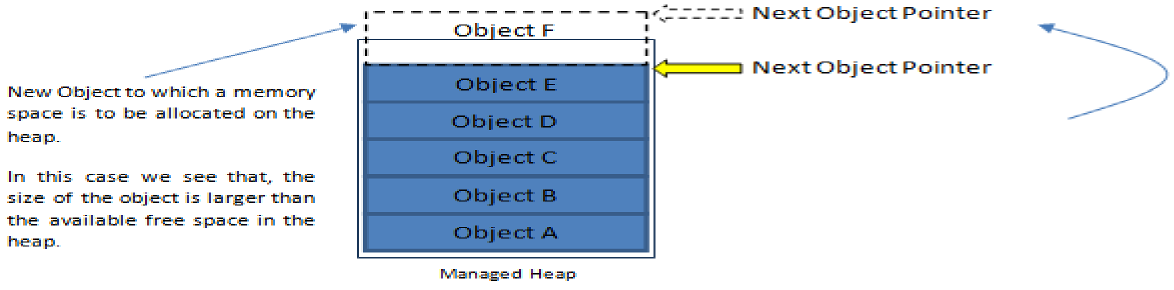


When we create an object i.e. declare a type, the runtime reserves a contiguous region of address space that initially has no storage allocated for it, when we create an object. The Jitter (Just-in-time Compiler) updates the manage heap and also maintains the addresses of all objects that are allocated within this managed heap. After declaring an object, we instantiate the object using the 'new' operator. This 'new' operator first makes sure that the bytes required by the new object fit in the reserved region (committing storage if necessary). If the object fits, then pointer points to the object in the heap, this object's constructor is called, and the new operator returns the address of the object. At this point, 'Next Object Pointer' is incremented past the object so that it points to where the next object will be placed in the heap.



Similarly, the objects keep adding to the heap, with the instantiation of each new object.

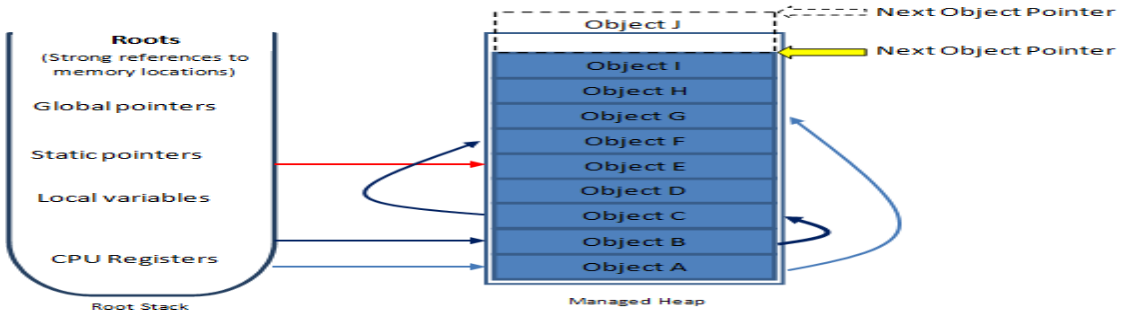
In this process, it might reach to a condition where there is no more space left in the heap to be allocated to a new object, or the available space is not enough to accommodate the new object that has been instantiated. This is to say, when the 'Next Object Pointer' reaches beyond the end of the address space, a memory deficiency occurs and therefore the CLR calls the Garbage Collector and gives it this memory heap.


Object F
Next Object Pointer
Next Object Pointer
New Object to which a memory space is to be allocated on the heap.
In this case we see that, the size of the object is larger than the available free space in the heap.
Object E
Object D
Object C
Object B
Object A
Managed Heap

Application Roots

Before we proceed to examine and understand the Garbage Collection process, let's take a final look at the Managed Heap. So, we see that the required memory space is allocated on the heap, to all the objects that have been initialized. An Object Pointer is set to the final location and stores the address, where the next object would be allocated the memory space on the heap. Is this all? No it is not. Many objects on a heap are linked one to the other forming a chain, depending on which object is referenced by whom. There might be multiple such chains in the heap. The reference to the first object, i.e. the memory location to any such chain is called a Root. Every application has a set of such roots. E.g. the global and static object pointers, any local variable/parameter object pointers and any CPU registers containing pointers to objects.

All these roots are stored in the Application stack and identify storage locations, which refer to objects on the managed heap or to objects that are set to null. This list of active roots is maintained by the just-in-time (JIT) compiler and common language runtime, and is made accessible to the garbage collector's algorithm.



In reality, a collection occurs when generation 0 is completely full. Briefly, a generation is a mechanism implemented by the garbage collector in order to improve performance. The idea is that newly created objects are part of a young generation, and objects created early in the application's lifecycle are in an old generation. Separating objects into generations can allow the garbage collector to collect specific generations instead of collecting all objects in the managed heap.

The Garbage Collector

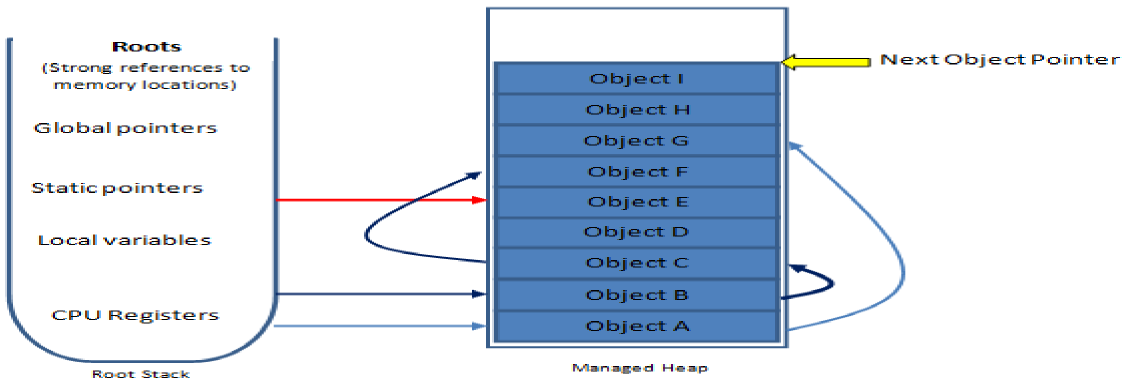
The basic algorithm used by the garbage collector is quite simple:

- Mark all managed memory as garbage
- Look for used memory blocks, and mark them as valid
- Discard all unused memory blocks
- Compact the heap

Once the GC receives this memory heap, The garbage collector checks to see if there are any objects in the heap that are no longer being used by the application.

If such objects exist, then the memory used by these objects can be reclaimed. (If no more memory is available for the heap, then the new operator throws an Out Of Memory Exception.)
Now there is a very valid question. How does the garbage collector know if the application is using an object or not?

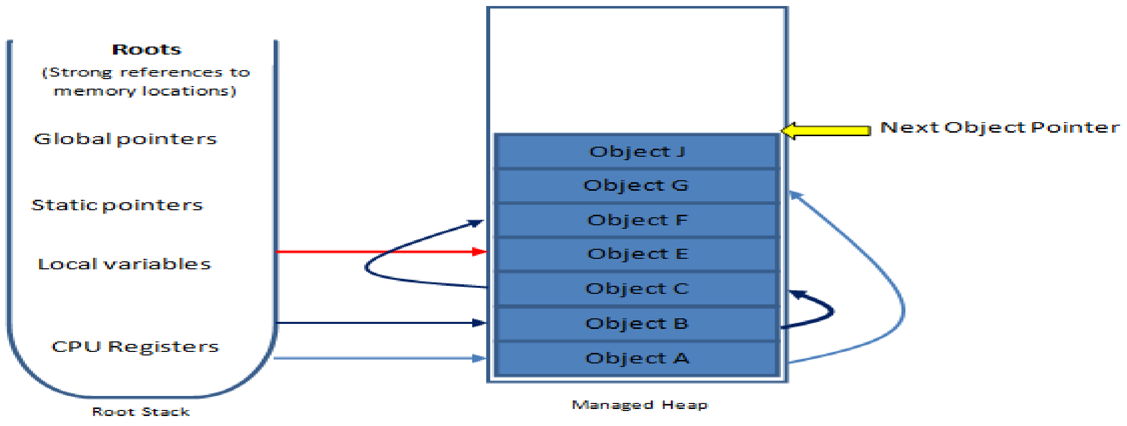
When the garbage collector starts running, it makes the assumption that all objects in the heap are garbage. In other words, it assumes that none of the application's roots refer to any objects in the heap. Now, the garbage collector starts checking the roots and building a graph of all objects reachable from the roots.



Managed Heap before Garbage Collection

In the above figure the Objects A, and E are directly referenced by the Application Roots. The Garbage collector starts from the Root, traversing through all the objects in the chain and stores them, making a graph. This action is repeated for all the chains one by one, and hence all the objects that are either referred directly by the Application Root or are indirectly referred by another object on the heap are added to the Graph. Here it is important to note that, if while traversing down a chain, the garbage collector reaches an object that it has already added to the Graph, then it stops going further down that path. This is because, it is obvious that the next objects further down the chain would already be there in the Graph. This increases the performance of the process as well as it also prevents infinite loops. In case there are any circular linked lists of objects. Once all the roots/chains have been checked and added to the Graph, the Garbage Collector assumes that are not in the Graph are not referenced from the application's roots and hence are not accessible by the application, and are therefore considered garbage. The garbage collector now checks the entire heap for contiguous blocks of garbage objects. It shifts the non-garbage objects down in the heap, removing all of the gaps in the heap. It then modifies the application's roots so that all the pointers now point to the objects' new locations. Also, if any object contains a pointer to another object, it corrects these pointers as well. Finally, the Next

Object Pointer is positioned just after the last non-garbage object. Now the heap is ready to allocate the memory space, required by the new object that was initialized and had triggered the Garbage Collection.



Managed Heap after Garbage Collection

However there might be a case where even after the Garbage Collection there is not enough space on the Heap to accommodate the new object. In such a scenario another very important feature of Garbage Collector called; 'Generations' comes into action.

Generations

Though this feature of the garbage collector resolves the issue where even after the garbage collection, there is not enough space on the heap to accommodate the new object. But, then it is absolutely wrong to say that the feature of Generations address and was developed only to address the memory accommodation issue. In fact this feature of the garbage collector exists purely to improve the performance.

The Garbage Collector with this feature is known as an 'Ephemeral Garbage Collector', and it makes the following assumptions:

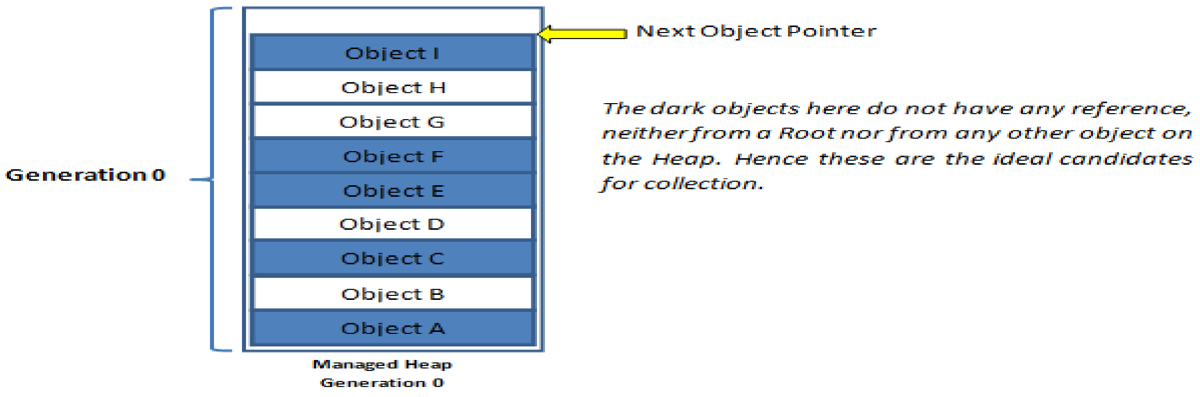
- The newer an object is, the shorter its lifetime will be.
- The older an object is, the longer its lifetime will be.
- Newer objects tend to have strong relationships to each other and are frequently accessed around the same time.
- Compacting a portion of the heap is faster than compacting the whole heap.

In order to understand, physically the Generations are nothing more than the identification of the Heap area into 3 categories. Once the Garbage Collector receives the memory heap, it divides/distributes the contents in the heap into three categories of objects and stores these objects into three tables:

- Generation 0 this stores short lived objects.
- Generation 1 this stores older objects.
- Generation 2 this stores the oldest objects.

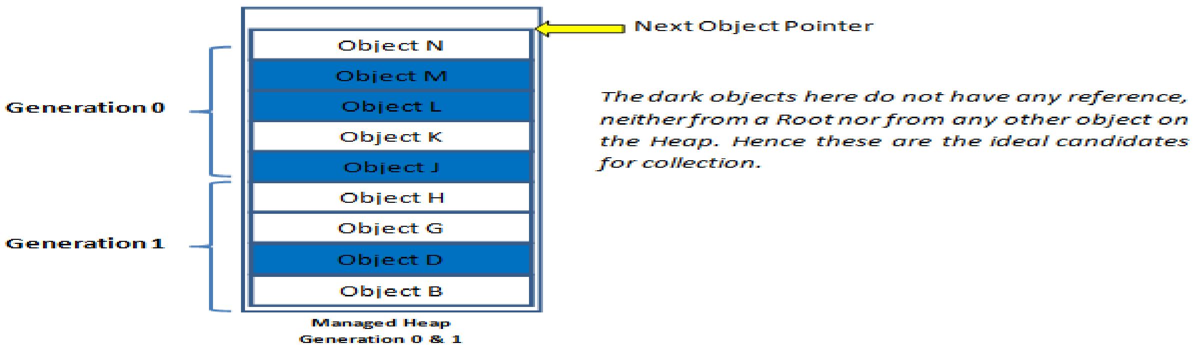
Generation 0

At first when there are no objects on the Heap, then every time a new object is added, it goes into Generation 0. , as discussed above. Hence the objects in the Generation 0 are all latest objects that have never been examined by the Garbage Collector.


Next Object Pointer
Object I
Object H
Object G
Object F
Object E
Object D
Object C
Object B
Object A
Generation 0
The dark objects here do not have any reference, neither from a Root nor from any other object on the Heap. Hence these are the ideal candidates for collection.
Managed Heap
Generation 0

Generation 1

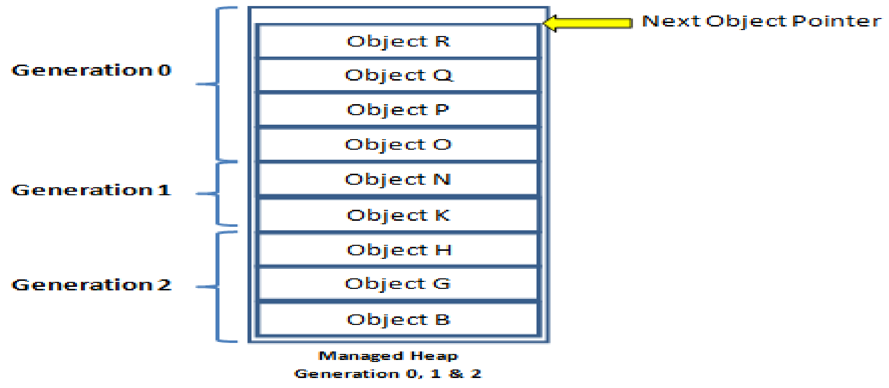
Here Objects A, C, E, F and I are free objects, i.e. they do not have any reference, neither from a Root nor from any other object on the heap. When this Generation 0 is completely full, i.e. there is no more space for a new Object then the garbage collection occurs. All the objects that survive the collection (Objects B, D, G and H) are compacted into the left most portion of the heap. These objects are the old objects and are now said to be in Generation1.


Next Object Pointer
Object N
Object M
Object L
Object K
Object J
Object H
Object G
Object D
Object B
Generation 0
Generation 1
The dark objects here do not have any reference, neither from a Root nor from any other object on the Heap. Hence these are the ideal candidates for collection.
Managed Heap
Generation 0 & 1

Now any new object added on the heap, goes to the Generation 0. So we see that in Generation 0 we have New Objects and in Generation 1 we have relative older objects.

Generation 2

Same as before, if again the Generation 0 is full, then again the heap is subjected to Garbage Collection. However, this time first all the objects in Generation 1, that survive the collection (Objects B, G & H) are compacted and are now considered to be in Generation 2. Then, all the objects in Generation 0 are compacted as before and are now considered to be in Generation 1. This clears the space in Generation 0 for new objects. Hence, we see that in Generation 2 we have the oldest objects (objects B, G & H), in generation we have relatively older objects (objects K & N) and in generation we have the most recently added i.e. the newest objects (objects O, P, Q & R).



Currently, generation 2 is the highest generation supported by the runtime's garbage collector. When future collections occur, any surviving objects currently in Generation 2 simply stays in generation 2.

But now the question is, how exactly does this feature of generations, help in improving the performance?

1. When the heap fills and a collection occurs, the garbage collector first examines and collects the objects in the Generation 0. The assumption is that the newer an object is, the shorter its lifetime is expected to be. Hence, we hope that, collecting and compacting the Generation 0 objects would free enough space on the heap to accommodate the new object. We can see that, examining and collecting objects from just the Generation 0 is much faster than if the collector had examined the objects in all the generations. However, even if collecting objects from Generation 0 doesn't provide the required amount of space, then the collector attempts to collect the objects from generations 1 and 0. Similarly, if the collection from both the generations 1 & 0, fails to create the required space, then the objects are collected from all the Generations-2, 1, and 0.
2. A generational collector can offer more optimizations by not traversing every object in the managed heap. If a root or object refers to an object in an old generation, the garbage collector can ignore any of the older objects' inner references, decreasing the time required to build the graph of reachable objects. Of course, it is possible that an old object refers to a new object. So that these objects are examined, the collector can take advantage of the system's write-watch support (provided by the Win32 GetWriteWatch function in Kernel32.dll). This support lets the collector know which old objects (if any) have been written to since the last collection. These specific old objects can have their references checked to see if they refer to any new objects.

Garbage Collection in Multithreaded Applications

While understanding the mechanism of Garbage Collection, as above, we made a big assumption that, only one thread is running. Though in fact under normal circumstances, there would be multiple threads accessing the managed heap or manipulating objects allocated space on the managed heap. When the space is requested by a new object from one thread and the Collection is triggered, then it becomes important that other threads must not access any objects (including object references on its own stack) since the collector is likely to move these objects, changing their memory locations. Hence, when the garbage collector starts a collection, all threads executing managed code are suspended. The runtime employs different mechanisms to safely suspend threads so as to keep the threads running as long as possible before the collection is performed. This is to reduce overhead as much as possible. The mechanisms that the garbage collector employs when applications have multiple threads are:

1. Fully Interruptible Code

When a collection starts, the collector suspends all application threads and determines -

i. Where the thread was suspended, i.e. where in a method the thread stopped,
ii. What object references the code is currently accessing, and
iii. Where those references are held in a variable or a CPU register, etc.

2. Hijacking

When a collection occurs, the collector modifies the thread's stack, such that when a method returns, a special function is executed, to suspend the thread. When the collection is complete, the thread resumes and returns to the method that originally called it. This is called Thread Hijacking. This thread hijacking allows threads that are executing unmanaged code to continue execution while a garbage collection is occurring. Please note, this does not cause any problem, since unmanaged code is not accessing objects on the managed heap unless the objects do not contain object references. If a thread that is currently executing unmanaged code returns to managed code, the thread is hijacked and is suspended until the Garbage Collection completes.

3. Safe Points

While compiling a method, if the just-in-time compiler finds that a Garbage Collection is pending -

i. It inserts a call to a function that suspends the thread,
ii. First the Garbage Collection is completed, and
iii. Then the thread is resumed.

This position where the compiler inserts these method calls is called a Safe Point.

4. Synchronization-free Allocations

On a multiprocessor system, generation 0 of the managed heap is split into multiple memory areas using one area per thread. This allows multiple threads to make allocations simultaneously so that exclusive access to the heap is not required.

5. Scalable Collections

On a multiprocessor system running the server version of the execution engine (MSCorSvr.dll), the managed heap is split into several sections, one per CPU. When a collection is initiated, the collector has one thread per CPU; all threads collect their own sections simultaneously. The workstation version of the execution engine (MSCorWks.dll) doesn't support this feature.

Finalisation

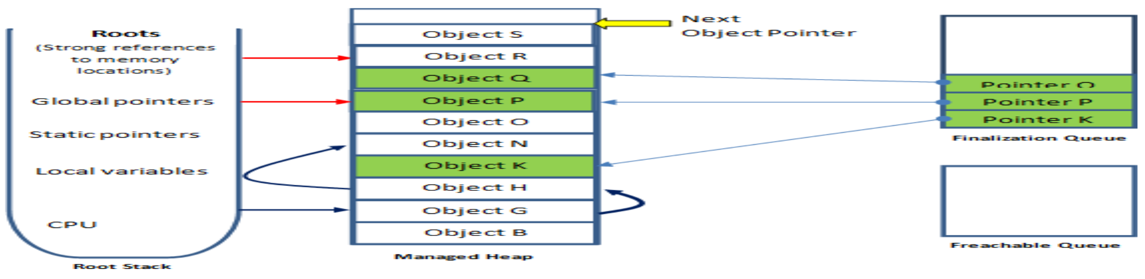
Say if you want to execute a piece of code for an object, just before it is subjected to Collection by the Garbage Collector. This has been made possible by a feature of the Garbage Collector called 'Finalization'.

Finalization allows a resource to be gracefully cleaned when it is being collected. By using finalization, a resource like a file or network connection, is able to clean itself up properly when the garbage collector decides to free the resource's memory. This is made possible by an override void Finalize () with clean up resource code in it. When the Garbage Collector recognizes that the class object is pretty much going to be a piece of garbage, it calls the Finalize() method.

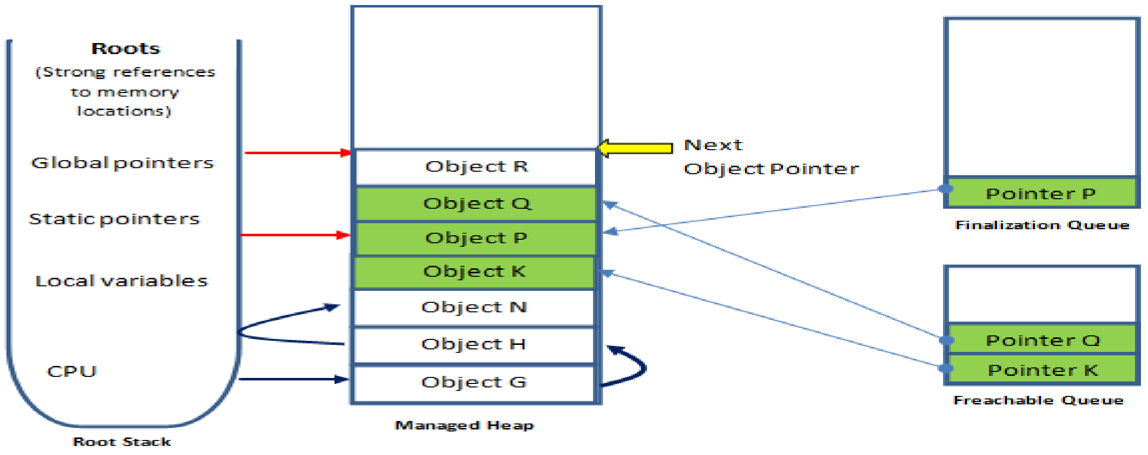
```
public class objBase
{
   public objBase()
   {
   }

   protected override void Finalize()
   {
      // Resource Cleanup Code
   }
}
```

Let's try and understand, how the Finalization works. When we create and then instantiate an object using the 'new' operator. This 'new' operator first makes sure that the bytes required by the new object are available on the heap, i.e. the new object could fit on the heap. It then not only puts the object on the heap, but also checks if the object's type contains a Finalize method. If yes then a pointer to the object is placed on the Finalization Queue.

The Finalization Queue is an internal data structure (Queue) controlled by the garbage collector. Each entry in the queue points to an object that should have its Finalize method called before the object's memory can be reclaimed. E.g. when the objects K, O and P were created, the system detected that these objects had Finalize methods; hence after adding these objects to the heap, the pointers to these objects were added to the Finalization Queue.



When the garbage collector finds the objects B, K, O, Q and S to be garbage, it scans the Finalization Queue looking for pointers to these objects. If a pointer is found, it is removed from the Finalization Queue and added to the Freachable Queue. The Freachable Queue is another internal data structure (Queue) controlled by the garbage collector. Each pointer in the Freachable Queue identifies an object that is ready to have its Finalize method called. Hence, we see that when the collection occurs, the memory occupied by objects B, O, and S is reclaimed because these objects did not have a Finalize method that needs to be called. However, the memory occupied by objects K and Q could not be reclaimed because their Finalize method has not been called yet and the pointers to both these objects are moved from the Finalization Queue to the Freachable Queue.

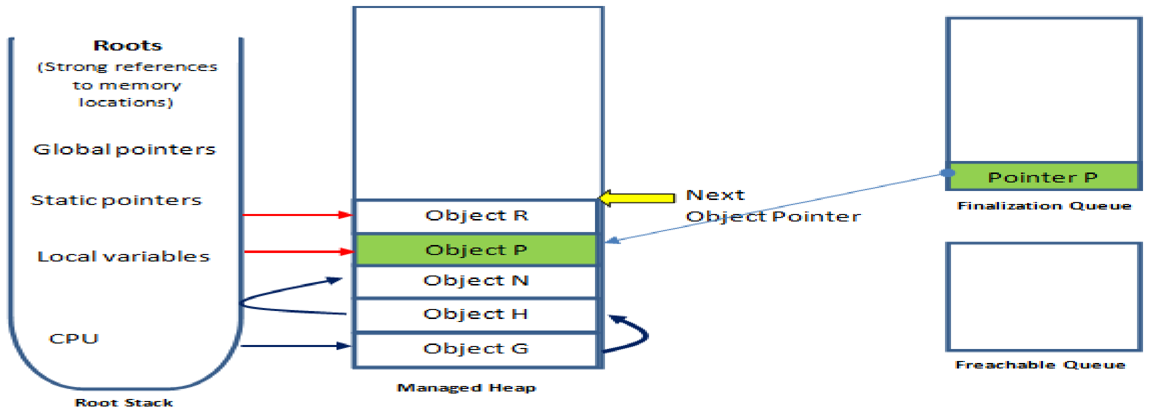


With this we reach to an understanding that, when an object is not reachable, the garbage collector considers the object garbage. Then, when the garbage collector moves an object's entry from the Finalization Queue to the Freachable Queue, the object is no longer considered garbage and its memory is not reclaimed. At this point, the garbage collector has finished identifying garbage. Some of the objects identified as garbage have been reclassified as not garbage.

The garbage collector compacts the reclaimable memory and the special runtime thread empties the Freachable Queue, executing each object's Finalize method.

There is a special runtime thread dedicated to calling Finalize methods. Normally when the Freachable Queue is empty, this thread sleeps. But when the pointers are added to this queue, this thread awakes, removes each entry from the queue, and calls each object's Finalize method. Because of this, you should not execute any code in a Finalize method that makes any assumption about the thread that's executing the code. For example, avoid accessing thread local storage in the Finalize method. In our example, we see that the pointers to objects K and Q have been moved to the Freachable Queue and their Finalize() method has been called by the thread.

Now the next time the garbage collection occurs, the Garbage Collector finds the pointers to objects K and Q in the Freacheble Queue and realizes that these finalized objects are truly garbage, and hence it goes ahead and reclaims the memory held by the objects K and Q. Please note that two iterations of the Garbage Collection process are required to free the memory space occupied by the objects that require finalization. In reality, more than two collections may be necessary since the objects could get promoted to an older generation.



So, we see that the Finalization process is quite fascinating, but I personally feel, it is not advisable to use this feature --

- When a garbage collection occurs, the Garbage collector looks for and identifies the objects that contain a Finalize method. These objects are promoted to older generations, which for the time being prevent the object's memory from being collected. Moreover, all the objects referred to directly or indirectly by these finalized objects, also get promoted to older generations.

- Finalizable objects take longer to allocate.
- When a garbage collector is forced to execute the Finalize method of an object, it has a strong potential to significantly hurt the performance, as it would require each object to be finalized.
- Finalizable objects may refer to other non-finalizable objects, prolonging their lifetime unnecessarily. In fact, you might want to consider breaking a type into two different types: a lightweight type with a Finalize method that doesn't refer to any other objects, and a separate type without a Finalize method that does refer to other objects.
- We have no control over when a Finalize method would execute. The object may hold on to the resources until the next time the garbage collector runs.
- When an application terminates, some objects are still reachable and will not have their Finalize method called. This can happen if background threads are using the objects or if objects are created during application shutdown or AppDomain unloading. In addition, by default, Finalize methods are not called for unreachable objects when an application exits so that the application may terminate quickly. Of course, all operating system resources will be reclaimed, but any objects in the managed heap are not able to clean up gracefully. We can change this default behavior by calling the System.GC type's RequestFinalizeOnShutdown method. However, we should use this method with care since calling it means that our type is controlling a policy for the entire application.
- The runtime doesn't make any guarantees as to the order in which Finalize methods are called. For example, let's say there is an object that contains a pointer to an inner object. The garbage collector has detected that both objects are garbage. Furthermore, say that the inner object's Finalize method gets called first. Now, the outer object's Finalize method is allowed to access the inner object and call methods on it, but the inner object has been finalized and the results may be unpredictable. For this reason, it is strongly recommended that Finalize methods not access any inner, member objects.

What about External Resources?

The garbage collector efficiently handles freeing resources from the managed heap, but a collection is only initiated when memory pressure triggers a collection. What about classes that manage limited external resources such as database connections, Windows handles or External files etc.? Waiting until a garbage collection is triggered to clean up database connections or file handles can severely degrade system performance.

Dispose Method

Microsoft has provided another alternative whereby we could use the IDisposable interface. When we implement this interface, we get a Dispose() method which can be used to convey a message to the CLR to take care of objects that are no longer required by the application.

This method can be used to close or release unmanaged resources such as files, streams, and handles held by an instance of the class that implements this interface. This method is, by convention, used for all tasks associated with freeing resources held by an object, or preparing an object for reuse. When implementing this method, objects must seek to ensure that all held resources are freed by propagating the call through the containment hierarchy. For example, if an object A allocates an object B, and object B allocates an object C, then A's Dispose implementation must call Dispose on B, which must in turn call Dispose on C. Objects must also call the Dispose method of their base class if the base class implements IDisposable. If an object's Dispose method is called more than once, the object must ignore all calls after the first one. The object must not throw an exception if its Dispose method is called multiple times. Dispose can throw an exception if an error occurs because a resource has already been freed and Dispose had not been called previously. Because the Dispose method must be called explicitly, objects that implement IDisposable must also implement a finalizer to handle freeing resources when Dispose is not called. By default, the garbage collector will automatically call an object's finalizer prior to reclaiming its memory. However, once the Dispose method has been called, it is typically unnecessary for the garbage collector to call the disposed object's finalizer. To prevent automatic finalization, Dispose implementations can call the GC.SuppressFinalize method.

Using the System.GC Class

The System.GC type allows your application some direct control over the garbage collector. It is used to access the garbage collection mechanism exposed by the .NET framework. This class includes the following useful methods:

- **GC.MaxGeneration**, is used to query the maximum generation supported by the managed heap. By default, the GC.MaxGeneration property always returns 2.
- **GC.SuppressFinalize** was described earlier in the column; this method inhibits finalization for an object. Call this method if you have already released external resources owned by an object.
- **GC.Collect** comes in two versions. The version that has no parameter performs a full collection on all generations in the managed heap. Another version accepts an integer value representing the generation to be collected. You'll rarely need to call this method, as the garbage collector automatically runs when needed.
- *void GC.Collect(Int32 Generation)*
- *void GC.Collect()*

- The first method allows you to specify which generation to collect. You may pass any integer from 0 to GC.MaxGeneration, inclusive. Passing 0 causes generation 0 to be collected; passing 1 cause generation 1 and 0 to be collected; and passing 2 causes generation 2, 1, and 0 to be collected. The version of the Collect method that takes no parameters forces a full collection of all generations and is equivalent to calling:*GC.Collect(GC.MaxGeneration);*
- **GC.Get Generation** returns the generation number for an object passed as a parameter. This method is useful when debugging or tracing for performance reasons, but has limited value in most applications.
  Int32 Get Generation(Object obj)
- *Int32 GetGeneration(Weak Reference wr)*
  The first version of Get Generation takes an object reference as a parameter, and the second version takes a Weak Reference reference as a parameter. Of course, the value returned will be somewhere between 0 and GC.MaxGeneration, inclusive
- **GC.GetTotalMemory** returns the amount of memory allocated in the heap. This number is not exact due to the way the managed heap works, but a close approximation can be obtained if true is passed as a parameter. This causes a collection to be performed before the memory usage is calculated.
- **Wait For Pending Finalizers**, this method simply suspends the calling thread until the thread processing the Freachable Queue has emptied the queue, calling each object's Finalize method. In most applications, it is unlikely that you will ever have to call this method.

Exception Handling

1. Exception is a runtime error which arises because of abnormal condition in a code sequence.
2. The exceptions are defined as the anomalies that occur during the execution of a program. Which can be because of the user, logic or system failure/error.
3. If a user or programmer does not provide a proper procedure to handle these anomalies or exceptions, then the .NET run time environment provide a default procedure, which terminates the current execution of that program .
4. Exception handling is an in built mechanism in .NET framework to detect and handle run time errors.
5. try, catch and finally are block which are used to handle the exceptions in .Net.
6. The try encloses the statements that might throw an exception whereas catch handles an exception if one exists. The finally can be used for doing any cleanup process.

7. If during exception of program no exception is occurred inside the try block, then the control is directly transferred to finally block.

   Examples of Exceptions :

   System.OutOfMemoryException

   System.NullReferenceException

   Syste.InvalidCastException

   Syste.ArrayTypeMismatchException

   System.IndexOutOfRangeException

   System.ArithmeticException

   System.DevideByZeroException

   System.OverFlowException

   Below code will terminate the program after exception is thrown.

```
public class Exception_Handling
{
   public static void Main(string[] args)
   {
      int first = 0;
      int second = 100 / first;
      Console.WriteLine(second);
   }
}
```

Below code will run after the exception is raised or thrown by program.

```
public class Exception_Handling
{
   public static void Main(string[] args)
   {
      int first = 0;
      int second = 0;
      try
      {
         second = 100 / first;
         Console.WriteLine("This code line is not executed");
      }
      catch (DivideByZeroException de)
```

```
        {
           Console.WriteLine(de.ToString());
        }
        Console.WriteLine("Result of Division :- {0}", div);
    }
}
```

Code Access Security

Code Access Security (CAS), in the Microsoft .NET framework, is Microsoft's solution to prevent untrusted code from performing privileged actions. When the CLR loads an assembly it will obtain evidence for the assembly and use this to identify the code group that the assembly belongs to. A code group contains a permission set (one or more permissions). Code that performs a privileged action will perform a code access demand which will cause the CLR to walk up the call stack and examine the permission set granted to the assembly of each method in the call stack. The code groups and permission sets are determined by the administrator of the machine who defines the security policy.

Evidence

Evidence can be any information associated with an assembly. The default evidences that are used by .NET code access security are:

Application directory: the directory in which an assembly resides.

Publisher: the assembly's publisher's digital signature (requires the assembly to be signed via Authenticode).

URL: the complete URL where the assembly was launched from

Site: the hostname of the URL/Remote Domain/VPN.

Zone: the security zone where the assembly resides

Hash: a cryptographic hash of the assembly, which identifies a specific version.

Strong Name: a combination of the assembly name, version and public key of the signing key used to sign the assembly. The signing key is not an X509 certificate, but a custom key pair generated by the strong naming tool, SN.EXE or by Visual Studio.

A developer can use custom evidence (so-called assembly evidence) but this requires writing a security assembly and in version 1.1 of .NET this facility does not work.

Evidence based on a hash of the assembly is easily obtained in code. For example, in C#, evidence may be obtained by the following code clause:

this.GetType().Assembly.Evidence

Policy

A policy is a set of expressions that uses evidence to determine a code group membership. A code group gives a permission set for the assemblies within that group. There are four policies in .NET:

Enterprise: policy for a family of machines that are part of an Active Directory installation.

Machine: policy for the current machine.

User: policy for the logged on user.

AppDomain: policy for the executing application domain.

The first three policies are stored in XML files and are administered through the .NET Configuration Tool 1.1 (mscorcfg.msc). The final policy is administered through code for the current application domain.

Code access security will present an assembly's evidence to each policy and will then take the intersection (that is the permissions common to all the generated permission set) as the permissions granted to the assembly. By default, the Enterprise, User, and AppDomain policies give full trust (that is they allow all assemblies to have all permissions) and the Machine policy is more restrictive. Since the intersection is taken this means that the final permission set is determined by the Machine policy.

Note that the policy system has been eliminated in .NET Framework 4.0.

Code group

Code groups associate a piece of evidence with a named permission set. The administrator uses the .NET Configuration Tool to specify a particular type of evidence (for example, Site) and a particular value for that evidence (for example, www.mysite.com) and then identifies the permission set that the code group will be granted.

Demands

Code that performs some privileged action will make a demand for one or more permissions. The demand makes the CLR walk the call stack and for each method the CLR will ensure that the demanded permissions are in the method's assembly's granted permissions. If the permission is not granted then a security exception is thrown. This prevents downloaded code from performing privileged actions. For example, if an assembly is downloaded from an untrusted site the assembly will not have any file IO permissions and so if this assembly attempts to access a file code access security will throw an exception preventing the call.

VB.Net - Overview

Visual Basic .NET (VB.NET) is an object-oriented computer programming language implemented on the .NET Framework. Although it is an evolution of classic Visual Basic language, it is not backwards-compatible with VB6, and any code written in the old version does not compile under VB.NET. Like all other .NET languages, VB.NET has complete support for object-oriented concepts. Everything in VB.NET is an object, including all of the primitive types (Short, Integer, Long, String, Boolean, etc.) and user-defined types, events, and even assemblies. All objects inherits from the base class Object. VB.NET is implemented by Microsoft's .NET framework. Therefore, it has full access to all the libraries in the .Net Framework. It's also possible to run VB.NET programs on Mono, the open-source alternative to .NET, not only under Windows, but even Linux or Mac OSX.

The following reasons make VB.Net a widely used professional language:

- Modern, general purpose.
- Object oriented.
- Component oriented.
- Easy to learn.
- Structured language.
- It produces efficient programs.
- It can be compiled on a variety of computer platforms.
- Part of .Net Framework.

Strong Programming Features VB.Net

VB.Net has numerous strong programming features that make it endearing to multitude of programmers worldwide. Let us mention some of these features:

- Boolean Conditions
- Automatic Garbage Collection
- Standard Library
- Assembly Versioning
- Properties and Events
- Delegates and Events Management
- Easy-to-use Generics
- Indexers

- Conditional Compilation
- Simple Multithreading

VB.Net - Environment Setup

In this chapter, we will discuss the tools available for creating VB.Net applications.

We have already mentioned that VB.Net is part of .Net framework and used for writing .Net applications. Therefore before discussing the available tools for running a VB.Net program, let us understand how VB.Net relates to the .Net framework.

The .Net Framework

The .Net framework is a revolutionary platform that helps you to write the following types of applications:

- Windows applications
- Web applications
- Web services

The .Net framework applications are multi-platform applications. The framework has been designed in such a way that it can be used from any of the following languages: Visual Basic, C#, C++, Jscript, and COBOL, etc.

All these languages can access the framework as well as communicate with each other.

The .Net framework consists of an enormous library of codes used by the client languages like VB.Net. These languages use object-oriented methodology.

Following are some of the components of the .Net framework:

- Common Language Runtime (CLR)
- The .Net Framework Class Library
- Common Language Specification
- Common Type System
- Metadata and Assemblies
- Windows Forms
- ASP.Net and ASP.Net AJAX
- ADO.Net
- Windows Workflow Foundation (WF)
- Windows Presentation Foundation
- Windows Communication Foundation (WCF)
- LINQ

For the jobs each of these components perform, please see ASP.Net - Introduction, and for details of each component, please consult Microsoft's documentation.

Integrated Development Environment (IDE) For VB.Net

Microsoft provides the following development tools for VB.Net programming:

- Visual Studio 2010 (VS)
- Visual Basic 2010 Express (VBE)
- Visual Web Developer

The last two are free. Using these tools, you can write all kinds of VB.Net programs from simple command-line applications to more complex applications. Visual Basic Express and Visual Web Developer Express edition are trimmed down versions of Visual Studio and has the same look and feel. They retain most features of Visual Studio. In this tutorial, we have used Visual Basic 2010 Express and Visual Web Developer (for the web programming chapter).

You can download it from here. It gets automatically installed in your machine. Please note that you need an active internet connection for installing the express edition.

Writing VB.Net Programs on Linux or Mac OS

Although the.NET Framework runs on the Windows operating system, there are some alternative versions that work on other operating systems. Mono is an open-source version of the .NET Framework which includes a Visual Basic compiler and runs on several operating systems, including various flavors of Linux and Mac OS. The most recent version is VB 2012.

The stated purpose of Mono is not only to be able to run Microsoft .NET applications cross-platform, but also to bring better development tools to Linux developers. Mono can be run on many operating systems including Android, BSD, iOS, Linux, OS X, Windows, Solaris and UNIX.

VB.Net - Program Structure

Before we study basic building blocks of the VB.Net programming language, let us look a bare minimum VB.Net program structure so that we can take it as a reference in upcoming chapters.

VB.Net Hello World Example

A VB.Net program basically consists of the following parts:

- Namespace declaration
- A class or module
- One or more procedures
- Variables

- The Main procedure
- Statements & Expressions
- Comments

Let us look at a simple code that would print the words "Hello World":

```
Imports System
Module Module1
  'This program will display Hello World
  Sub Main()
    Console.WriteLine("Hello World")
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Hello, World!
```

Let us look various parts of the above program:

- The first line of the program **Imports System** is used to include the System namespace in the program.
- The next line has a **Module** declaration, the module *Module1*. VB.Net is completely object oriented, so every program must contain a module of a class that contains the data and procedures that your program uses.
- Classes or Modules generally would contain more than one procedure. Procedures contain the executable code, or in other words, they define the behavior of the class. A procedure could be any of the following:
  - Function
  - Sub
  - Operator
  - Get
  - Set
  - AddHandler
  - RemoveHandler
  - RaiseEvent

- The next line('This program) will be ignored by the compiler and it has been put to add additional comments in the program.
- The next line defines the Main procedure, which is the entry point for all VB.Net programs. The Main procedure states what the module or class will do when executed.
- The Main procedure specifies its behavior with the statement
  Console.WriteLine("Hello World")
  WriteLine is a method of the *Console* class defined in the *System* namespace. This statement causes the message "Hello, World!" to be displayed on the screen.
- The last line **Console.ReadKey()** is for the VS.NET Users. This will prevent the screen from running and closing quickly when the program is launched from Visual Studio .NET.
  Compile & Execute VB.Net Program:
  If you are using Visual Studio.Net IDE, take the following steps:
- Start Visual Studio.
- On the menu bar, choose File, New, Project.
- Choose Visual Basic from templates
- Choose Console Application.
- Specify a name and location for your project using the Browse button, and then choose the OK button.
- The new project appears in Solution Explorer.
- Write code in the Code Editor.
- Click the Run button or the F5 key to run the project. A Command Prompt window appears that contains the line Hello World.
  You can compile a VB.Net program by using the command line instead of the Visual Studio IDE:
- Open a text editor and add the above mentioned code.
- Save the file as **helloworld.vb**
- Open the command prompt tool and go to the directory where you saved the file.
- Type **vbc helloworld.vb** and press enter to compile your code.
- If there are no errors in your code the command prompt will take you to the next line and would generate **helloworld.exe** executable file.
- Next, type **helloworld** to execute your program.
- You will be able to see "Hello World" printed on the screen.

VB.Net - Basic Syntax

VB.Net is an object-oriented programming language. In Object-Oriented Programming methodology, a program consists of various objects that interact with each other by means of actions. The actions that an object may take are called methods. Objects of the same kind are said to have the same type or, more often, are said to be in the same class.

When we consider a VB.Net program, it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what do class, object, methods and instance variables mean.

- **Object** - Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, eating, etc. An object is an instance of a class.
- **Class** - A class can be defined as a template/blueprint that describes the behaviors/states that objects of its type support.
- **Methods** - A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.
- **Instance Variables** - Each object has its unique set of instance variables. An object's state is created by the values assigned to these instance variables.

A Rectangle Class in VB.Net

For example, let us consider a Rectangle object. It has attributes like length and width. Depending upon the design, it may need ways for accepting the values of these attributes, calculating area and displaying details.

Let us look at an implementation of a Rectangle class and discuss VB.Net basic syntax on the basis of our observations in it:

```
Imports System
Public Class Rectangle
   Private length As Double
   Private width As Double
   'Public methods
   Public Sub AcceptDetails()
      length = 4.5
      width = 3.5
   End Sub
   Public Function GetArea() As Double
      GetArea = length * width
```

```
End Function
    Public Sub Display()
        Console.WriteLine("Length: {0}", length)
        Console.WriteLine("Width: {0}", width)
        Console.WriteLine("Area: {0}", GetArea())
    End Sub
    Shared Sub Main()
        Dim r As New Rectangle()
        r.Acceptdetails()
        r.Display()
        Console.ReadLine()
    End Sub
End Class
```

When the above code is compiled and executed, it produces the following result:

```
Length: 4.5
Width: 3.5
Area: 15.75
```

In previous chapter, we created a Visual Basic module that held the code. Sub Main indicates the entry point of VB.Net program. Here, we are using Class that contains both code and data. You use classes to create objects. For example, in the code, r is a Rectangle object.

An object is an instance of a class:

```
Dim r As New Rectangle()
```

A class may have members that can be accessible from outside class, if so specified. Data members are called fields and procedure members are called methods.

Shared methods or **static** methods can be invoked without creating an object of the class. Instance methods are invoked through an object of the class:

```
Shared Sub Main()
   Dim r As New Rectangle()
   r.Acceptdetails()
   r.Display()
   Console.ReadLine()
End Sub
```

Identifiers

An identifier is a name used to identify a class, variable, function, or any other user-defined item. The basic rules for naming classes in VB.Net are as follows:

- A name must begin with a letter that could be followed by a sequence of letters, digits (0 - 9) or underscore. The first character in an identifier cannot be a digit.
- It must not contain any embedded space or symbol like? - +! @ # % ^ & * ( ) [ ] { } . ; : " ' / and \. However, an underscore (_) can be used.
- It should not be a reserved keyword.

VB.Net Keywords

The following table lists the VB.Net reserved keywords:

AddHandler	AddressOf	Alias	And	AndAlso	As	Boolean
ByRef	Byte	ByVal	Call	Case	Catch	CBool
CByte	CChar	CDate	CDec	CDbl	Char	CInt
Class	CLng	CObj	Const	Continue	CSByte	CShort
CSng	CStr	CType	CUInt	CULng	CUShort	Date
Decimal	Declare	Default	Delegate	Dim	DirectCast	Do
Double	Each	Else	ElseIf	End	End If	Enum
Erase	Error	Event	Exit	False	Finally	For
Friend	Function	Get	GetType	GetXML Namespace	Global	GoTo
Handles	If	Implements	Imports	In	Inherits	Integer
Interface	Is	IsNot	Let	Lib	Like	Long
Loop	Me	Mod	Module	MustInherit	MustOverride	MyBase
MyClass	Namespace	Narrowing	New	Next	Not	Nothing
Not Inheritable	Not Overridable	Object	Of	On	Operator	Option

Optional	Or	OrElse	Overloads	Overridable	Overrides	ParamArray
Partial	Private	Property	Protected	Public	RaiseEvent	ReadOnly
ReDim	REM	Remove Handler	Resume	Return	SByte	Select
Set	Shadows	Shared	Short	Single	Static	Step
Stop	String	Structure	Sub	SyncLock	Then	Throw
To	True	Try	TryCast	TypeOf	UInteger	While
Widening	With	WithEvents	WriteOnly	Xor		

VB.Net - Data Types

Data types refer to an extensive system used for declaring variables or functions of different types. The type of a variable determines how much space it occupies in storage and how the bit pattern stored is interpreted.

Data Types Available in VB.Net

VB.Net provides a wide range of data types. The following table shows all the data types available:

Data Type	Storage Allocation	Value Range
Boolean	Depends on implementing platform	**True** or **False**
Byte	1 byte	0 through 255 (unsigned)
Char	2 bytes	0 through 65535 (unsigned)
Date	8 bytes	0:00:00 (midnight) on January 1, 0001 through 11:59:59 PM on December 31, 9999
Decimal	16 bytes	0 through +/-79,228,162,514,264,337,593,543,950,335 (+/-7.9...E+28) with no decimal point; 0 through +/-7.9228162514264337593543950335 with 28 places to the right of the decimal
Double	8 bytes	-1.79769313486231570E+308 through -4.94065645841246544E-324, for negative values 4.94065645841246544E-324 through 1.79769313486231570E+308, for positive values
Integer	4 bytes	-2,147,483,648 through 2,147,483,647 (signed)
Long	8 bytes	-9,223,372,036,854,775,808 through 9,223,372,036,854,775,807(signed)
Object	4 bytes on 32-bit platform 8 bytes on 64-bit platform	Any type can be stored in a variable of type Object
SByte	1 byte	-128 through 127 (signed)
Short	2 bytes	-32,768 through 32,767 (signed)
Single	4 bytes	-3.4028235E+38 through -1.401298E-45 for negative values; 1.401298E-45 through 3.4028235E+38 for positive values

Example

The following example demonstrates use of some of the types:

```
Module DataTypes
  Sub Main()
    Dim b As Byte
    Dim n As Integer
    Dim si As Single
    Dim d As Double
    Dim da As Date
    Dim c As Char
    Dim s As String
    Dim bl As Boolean
    b = 1
    n = 1234567
    si = 0.12345678901234566
    d = 0.12345678901234566
    da = Today
    c = "U"c
    s = "Me"
    If ScriptEngine = "VB" Then
      bl = True
    Else
      bl = False
    End If
    If bl Then
      'the oath taking
       Console.Write(c & " and," & s & vbCrLf)
       Console.WriteLine("declaring on the day of: {0}", da)
       Console.WriteLine("We will learn VB.Net seriously")
       Console.WriteLine("Lets see what happens to the floating point variables:")
       Console.WriteLine("The Single: {0}, The Double: {1}", si, d)
    End If
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
U and, Me
declaring on the day of: 12/4/2012 12:00:00 PM
We will learn VB.Net seriously
Lets see what happens to the floating point variables:
The Single:0.1234568, The Double: 0.123456789012346
```

The Type Conversion Functions in VB.Net

VB.Net provides the following in-line type conversion functions:

S.N	Functions & Description
1	**CBool(expression)** Converts the expression to Boolean data type.
2	**CByte(expression)** Converts the expression to Byte data type.
3	**CChar(expression)** Converts the expression to Char data type.
4	**CDate(expression)** Converts the expression to Date data type
5	**CDbl(expression)** Converts the expression to Double data type.
6	**CDec(expression)** Converts the expression to Decimal data type.
7	**CInt(expression)** Converts the expression to Integer data type.
8	**CLng(expression)** Converts the expression to Long data type.
9	**CObj(expression)** Converts the expression to Object type.
10	**CSByte(expression)** Converts the expression to SByte data type.
11	**CShort(expression)** Converts the expression to Short data type.
12	**CSng(expression)** Converts the expression to Single data type.

Example:

The following example demonstrates some of these functions:

```
Module DataTypes
```

```
   Sub Main()
      Dim n As Integer
      Dim da As Date
      Dim bl As Boolean = True
      n = 1234567
      da = Today
      Console.WriteLine(bl)
      Console.WriteLine(CSByte(bl))
      Console.WriteLine(CStr(bl))
      Console.WriteLine(CStr(da))
      Console.WriteLine(CChar(CChar(CStr(n))))
      Console.WriteLine(CChar(CStr(da)))
      Console.ReadKey()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
True
-1
True
12/4/2012
1
1
```

VB.Net - Variables

A variable is nothing but a name given to a storage area that our programs can manipulate. Each variable in VB.Net has a specific type, which determines the size and layout of the variable's memory; the range of values that can be stored within that memory; and the set of operations that can be applied to the variable.

We have already discussed various data types. The basic value types provided in VB.Net can be categorized as:

Type	Example
Integral types	SByte, Byte, Short, UShort, Integer, UInteger, Long, ULong and Char

Floating point types	Single and Double
Decimal types	Decimal
Boolean types	True or False values, as assigned
Date types	Date

VB.Net also allows defining other value types of variable like **Enum** and reference types of variables like **Class**. We will discuss date types and Classes in subsequent chapters.

Variable Declaration in VB.Net **The** Dim **statement is used for variable declaration and storage allocation for one or more variables. The Dim statement is used at module, class, structure, procedure or block level.**

Syntax for variable declaration in VB.Net is:

```
[ < attributelist> ] [ accessmodifier ] [[ Shared ] [ Shadows ] | [ Static ]]
[ ReadOnly ] Dim [ WithEvents ] variablelist
```

Where,

- ***attributelist*** is a list of attributes that apply to the variable. Optional.
- ***accessmodifier*** defines the access levels of the variables, it has values as - Public, Protected, Friend, Protected Friend and Private. Optional.
- ***Shared*** declares a shared variable, which is not associated with any specific instance of a class or structure, rather available to all the instances of the class or structure. Optional.
- ***Shadows*** indicate that the variable re-declares and hides an identically named element, or set of overloaded elements, in a base class. Optional.
- ***Static*** indicates that the variable will retain its value, even when the after termination of the procedure in which it is declared. Optional.
- ***ReadOnly*** means the variable can be read, but not written. Optional.
- ***WithEvents*** specifies that the variable is used to respond to events raised by the instance assigned to the variable. Optional.
- ***Variablelist*** provides the list of variables declared.

Each variable in the variable list has the following syntax and parts:

```
variablename[ ( [ boundslist ] ) ] [ As [ New ] datatype ] [ = initializer ]
```

Where,

- ***variablename***: is the name of the variable
- ***boundslist***: optional. It provides list of bounds of each dimension of an array variable.
- ***New***: optional. It creates a new instance of the class when the Dim statement runs.
- ***datatype***: Required if Option Strict is On. It specifies the data type of the variable.
- ***initializer***: Optional if New is not specified. Expression that is evaluated and assigned to the variable when it is created.

Some valid variable declarations along with their definition are shown here:

```
Dim StudentID As Integer
Dim StudentName As String
Dim Salary As Double
Dim count1, count2 As Integer
Dim status As Boolean
Dim exitButton As New System.Windows.Forms.Button
Dim lastTime, nextTime As Date
```

Variable Initialization in VB.Net

Variables are initialized (assigned a value) with an equal sign followed by a constant expression. The general form of initialization is:

```
variable_name = value;
```

for example,

```
Dim pi As Double
pi = 3.14159
```

You can initialize a variable at the time of declaration as follows:

```
Dim StudentID As Integer = 100
Dim StudentName As String = "Bill Smith"
```

Example

Try the following example which makes use of various types of variables:

```
Module variablesNdataypes
  Sub Main()
    Dim a As Short
```

```
      Dim b As Integer
      Dim c As Double
      a = 10
      b = 20
      c = a + b
      Console.WriteLine("a = {0}, b = {1}, c = {2}", a, b, c)
      Console.ReadLine()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
a = 10, b = 20, c = 30
```

Accepting Values from User

The Console class in the System namespace provides a function **ReadLine** for accepting input from the user and store it into a variable. For example,

```
Dim message As String
message = Console.ReadLine
```

The following example demonstrates it:

```
Module variablesNdataypes
   Sub Main()
      Dim message As String
      Console.Write("Enter message: ")
      message = Console.ReadLine
      Console.WriteLine()
      Console.WriteLine("Your Message: {0}", message)
      Console.ReadLine()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result (assume the user inputs Hello World):

```
Enter message: Hello World
Your Message: Hello World
```

Lvalues and Rvalues

There are two kinds of expressions:

- **lvalue :** An expression that is an lvalue may appear as either the left-hand or right-hand side of an assignment.
- **rvalue :** An expression that is an rvalue may appear on the right- but not left-hand side of an assignment.

  Variables are lvalues and so may appear on the left-hand side of an assignment. Numeric literals are rvalues and so may not be assigned and can not appear on the left-hand side. Following is a valid statement:

```
Dim g As Integer = 20
```

But following is not a valid statement and would generate compile-time error:

```
20 = g
```

VB.Net - Constants and Enumerations

The **constants** refer to fixed values that the program may not alter during its execution. These fixed values are also called literals.

Constants can be of any of the basic data types like an integer constant, a floating constant, a character constant, or a string literal. There are also enumeration constants as well.

The constants are treated just like regular variables except that their values cannot be modified after their definition.

An **enumeration** is a set of named integer constants.

Declaring Constants

In VB.Net, constants are declared using the **Const** statement. The Const statement is used at module, class, structure, procedure, or block level for use in place of literal values.

The syntax for the Const statement is:

```
[ < attributelist> ] [ accessmodifier ] [ Shadows ]
Const constantlist
```

Where,

- ***attributelist***: specifies the list of attributes applied to the constants; you can provide multiple attributes separated by commas. Optional.
- ***accessmodifier***: specifies which code can access these constants. Optional. Values can be either of the: Public, Protected, Friend, Protected Friend, or Private.

- ***Shadows***: this makes the constant hide a programming element of identical name in a base class. Optional.
- ***Constantlist***: gives the list of names of constants declared. Required.
  Where, each constant name has the following syntax and parts:

```
constantname [ As datatype ] = initializer
```

- ***constantname***: specifies the name of the constant
- ***datatype***: specifies the data type of the constant
- ***initializer***: specifies the value assigned to the constant

For example,

```
'The following statements declare constants.'
Const maxval As Long = 4999
Public Const message As String = "HELLO"
Private Const piValue As Double = 3.1415
```

Example

The following example demonstrates declaration and use of a constant value:

```
Module constantsNenum
  Sub Main()
    Const PI = 3.14149
    Dim radius, area As Single
    radius = 7
    area = PI * radius * radius
    Console.WriteLine("Area = " & Str(area))
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Area = 153.933
```

Print and Display Constants in VB.Net

VB.Net provides the following print and display constants:

Constant	Description
vbCrLf	Carriage return/linefeed character combination.

vbCr	Carriage return character.
vbLf	Linefeed character.
vbNewLine	Newline character.
vbNullChar	Null character.
vbNullString	Not the same as a zero-length string (""); used for calling external procedures.
vbObjectError	Error number. User-defined error numbers should be greater than this value. For example: Err.Raise(Number) = vbObjectError + 1000
vbTab	Tab character.
vbBack	Backspace character.

Declaring Enumerations

An enumerated type is declared using the **Enum** statement. The Enum statement declares an enumeration and defines the values of its members. The Enum statement can be used at the module, class, structure, procedure, or block level.

The syntax for the Enum statement is as follows:

```
[ < attributelist > ] [ accessmodifier ]  [ Shadows ]
Enum enumerationname [ As datatype ]
   memberlist
End Enum
```

Where,

- ***attributelist***: refers to the list of attributes applied to the variable. Optional.
- ***asscessmodifier***: specifies which code can access these enumerations. Optional. Values can be either of the: Public, Protected, Friend or Private.
- ***Shadows***: this makes the enumeration hide a programming element of identical name in a base class. Optional.
- ***enumerationname***: name of the enumeration. Required
- ***datatype***: specifies the data type of the enumeration and all its members.
- ***memberlist***: specifies the list of member constants being declared in this statement. Required.

Each member in the memberlist has the following syntax and parts:

```
[< attribute list>] member name [ = initializer ]
```

Where,

- ***name***: specifies the name of the member. Required.
- ***initializer***: value assigned to the enumeration member. Optional.

For example,

```
Enum Colors
   red = 1
   orange = 2
   yellow = 3
   green = 4
   azure = 5
   blue = 6
   violet = 7
End Enum
```

Example

```
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
The Color Red is: 1
The Color Yellow is: 3
The Color Blue is: 6
The Color Green is: 4
```

VB.Net - Modifiers

The modifiers are keywords added with any programming element to give some especial emphasis on how the programming element will behave or will be accessed in the program

For example, the access modifiers: Public, Private, Protected, Friend, Protected Friend, etc., indicate the access level of a programming element like a variable, constant, enumeration or a class.

List of Available Modifiers in VB.Net

The following table provides the complete list of VB.Net modifiers:

S.N	Modifier	Description

1	Ansi	Specifies that Visual Basic should marshal all strings to American National Standards Institute (ANSI) values regardless of the name of the external procedure being declared.
2	Assembly	Specifies that an attribute at the beginning of a source file applies to the entire assembly.
3	Async	Indicates that the method or lambda expression that it modifies is asynchronous. Such methods are referred to as async methods. The caller of an async method can resume its work without waiting for the async method to finish.
4	Auto	The *charsetmodifier* part in the **Declare** statement supplies the character set information for marshaling strings during a call to the external procedure. It also affects how Visual Basic searches the external file for the external procedure name. The Auto modifier specifies that Visual Basic should marshal strings according to .NET Framework rules.
5	ByRef	Specifies that an argument is passed by reference, i.e., the called procedure can change the value of a variable underlying the argument in the calling code. It is used under the contexts of: Declare Statement Function Statement Sub Statement
6	ByVal	Specifies that an argument is passed in such a way that the called procedure or property cannot change the value of a variable underlying the argument in the calling code. It is used under the contexts of: Declare Statement Function Statement Operator Statement Property Statement Sub Statement
7	Default	Identifies a property as the default property of its class, structure, or interface.
8	Friend	Specifies that one or more declared programming elements are accessible from within the assembly that contains their declaration, not only by the component that declares them. **Friend** access is often the preferred level for an application's programming elements, and Friend

		is the default access level of an interface, a module, a class, or a structure.
9	In	It is used in generic interfaces and delegates.
10	Iterator	Specifies that a function or Get accessor is an iterator. An **iterator** performs a custom iteration over a collection.
11	Key	The Key keyword enables you to specify behavior for properties of anonymous types.
12	Module	Specifies that an attribute at the beginning of a source file applies to the current assembly module. It is not same as the Module statement.
13	MustInherit	Specifies that a class can be used only as a base class and that you cannot create an object directly from it.
14	MustOverride	Specifies that a property or procedure is not implemented in this class and must be overridden in a derived class before it can be used.
15	Narrowing	Indicates that a conversion operator (CType) converts a class or structure to a type that might not be able to hold some of the possible values of the original class or structure.
16	NotInheritable	Specifies that a class cannot be used as a base class.
17	NotOverridable	Specifies that a property or procedure cannot be overridden in a derived class.
18	Optional	Specifies that a procedure argument can be omitted when the procedure is called.
19	Out	For generic type parameters, the Out keyword specifies that the type is covariant.
20	Overloads	Specifies that a property or procedure redeclares one or more existing properties or procedures with the same name.
21	Overridable	Specifies that a property or procedure can be overridden by an identically named property or procedure in a derived class.
22	Overrides	Specifies that a property or procedure overrides an identically named property or procedure inherited from a base class.

23	ParamArray	ParamArray allows you to pass an arbitrary number of arguments to the procedure. A ParamArray parameter is always declared using ByVal.
24	Partial	Indicates that a class or structure declaration is a partial definition of the class or structure.
25	Private	Specifies that one or more declared programming elements are accessible only from within their declaration context, including from within any contained types.
26	Protected	Specifies that one or more declared programming elements are accessible only from within their own class or from a derived class.
27	Public	Specifies that one or more declared programming elements have no access restrictions.
28	ReadOnly	Specifies that a variable or property can be read but not written.
29	Shadows	Specifies that a declared programming element redeclares and hides an identically named element, or set of overloaded elements, in a base class.
30	Shared	Specifies that one or more declared programming elements are associated with a class or structure at large, and not with a specific instance of the class or structure.
31	Static	Specifies that one or more declared local variables are to continue to exist and retain their latest values after termination of the procedure in which they are declared.
32	Unicode	Specifies that Visual Basic should marshal all strings to Unicode values regardless of the name of the external procedure being declared.
33	Widening	Indicates that a conversion operator (CType) converts a class or structure to a type that can hold all possible values of the original class or structure.
34	WithEvents	Specifies that one or more declared member variables refer to an instance of a class that can raise events.
35	WriteOnly	Specifies that a property can be written but not read.

VB.Net - Statements

A **statement** is a complete instruction in Visual Basic programs. It may contain keywords, operators, variables, literal values, constants and expressions.

Statements could be categorized as:

- **Declaration statements** - these are the statements where you name a variable, constant, or procedure, and can also specify a data type.
- **Executable statements** - these are the statements, which initiate actions. These statements can call a method or function, loop or branch through blocks of code or assign values or expression to a variable or constant. In the last case, it is called an Assignment statement.

Declaration Statements

The declaration statements are used to name and define procedures, variables, properties, arrays, and constants. When you declare a programming element, you can also define its data type, access level, and scope.

The programming elements you may declare include variables, constants, enumerations, classes, structures, modules, interfaces, procedures, procedure parameters, function returns, external procedure references, operators, properties, events, and delegates.

Following are the declaration statements in VB.Net:

S.N	Statements and Description	Example
1	**Dim Statement** Declares and allocates storage space for one or more variables.	Dim number As Integer Dim quantity As Integer = 100 Dim message As String = "Hello!"
2	**Const Statement** Declares and defines one or more constants.	Const maximum As Long = 1000 Const naturalLogBase As Object = CDec(2.7182818284)
3	**Enum Statement** Declares an enumeration and defines the values of its members.	Enum CoffeeMugSize Jumbo ExtraLarge Large Medium Small

		End Enum
4	**Class Statement** Declares the name of a class and introduces the definition of the variables, properties, events, and procedures that the class comprises.	Class Box Public length As Double Public breadth As Double Public height As Double End Class
5	**Structure Statement** Declares the name of a structure and introduces the definition of the variables, properties, events, and procedures that the structure comprises.	Structure Box Public length As Double Public breadth As Double Public height As Double End Structure
6	**Module Statement** Declares the name of a module and introduces the definition of the variables, properties, events, and procedures that the module comprises.	Public Module myModule Sub Main() Dim user As String = InputBox("What is your name?") MsgBox("User name is" & user) End Sub End Module
7	**Interface Statement** Declares the name of an interface and introduces the definitions of the members that the interface comprises.	Public Interface MyInterface Sub doSomething() End Interface
8	**Function Statement** Declares the name, parameters, and code that define a Function procedure.	Function myFunction (ByVal n As Integer) As Double Return 5.87 * n End Function
9	**Sub Statement** Declares the name, parameters, and code that define a Sub procedure.	Sub mySub(ByVal s As String) Return End Sub
10	**Declare Statement**	Declare Function

	Declares a reference to a procedure implemented in an external file.	getUserName Lib "advapi32.dll" Alias "GetUserNameA" (ByVal lpBuffer As String, ByRef nSize As Integer) As Integer
11	**Operator Statement** Declares the operator symbol, operands, and code that define an operator procedure on a class or structure.	Public Shared Operator + (ByVal x As obj, ByVal y As obj) As obj Dim r As New obj ' implemention code for r = x + y Return r End Operator
12	**Property Statement** Declares the name of a property, and the property procedures used to store and retrieve the value of the property.	ReadOnly Property quote() As String Get Return quoteString End Get End Property
13	**Event Statement** Declares a user-defined event.	Public Event Finished()
14	**Delegate Statement** Used to declare a delegate.	Delegate Function MathOperator(ByVal x As Double, ByVal y As Double) As Double

Executable Statements

An executable statement performs an action. Statements calling a procedure, branching to another place in the code, looping through several statements, or evaluating an expression are executable statements. An assignment statement is a special case of an executable statement.

Example

The following example demonstrates a decision making statement:

```
Module decisions
```

```
  Sub Main()
    'local variable definition '
    Dim a As Integer = 10

    ' check the boolean condition using if statement '
    If (a < 20) Then
      ' if condition is true then print the following '
      Console.WriteLine("a is less than 20")
    End If
    Console.WriteLine("value of a is : {0}", a)
    Console.ReadLine()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
a is less than 20;
value of a is : 10
```

VB.Net - Directives

The VB.Net compiler directives give instructions to the compiler to preprocess the information before actual compilation starts.

All these directives begin with #, and only white-space characters may appear before a directive on a line. These directives are not statements.

VB.Net compiler does not have a separate preprocessor; however, the directives are processed as if there was one. In VB.Net, the compiler directives are used to help in conditional compilation. Unlike C and C++ directives, they are not used to create macros.

Compiler Directives in VB.Net

VB.Net provides the following set of compiler directives:

- The #Const Directive
- The #ExternalSource Directive
- The #If...Then...#Else Directives
- The #Region Directive

The #Const Directive

This directive defines conditional compiler constants. Syntax for this directive is:

```
#Const constname = expression
```

Where,

- ***constname***: specifies the name of the constant. Required.
- ***expression***: it is either a literal, or other conditional compiler constant, or a combination including any or all arithmetic or logical operators except **Is**.

For example,

```
#Const state = "WEST BENGAL"
```

Example

The following code demonstrates a hypothetical use of the directive:

```
Module mydirectives
#Const age = True
Sub Main()
  #If age Then
     Console.WriteLine("You are welcome to the Robotics Club")
  #End If
  Console.ReadKey()
End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
You are welcome to the Robotics Club
```

The #ExternalSource Directive

This directive is used for indicating a mapping between specific lines of source code and text external to the source. It is used only by the compiler and the debugger has no effect on code compilation.

This directive allows including external code from an external code file into a source code file.

Syntax for this directive is:

```
#ExternalSource( StringLiteral , IntLiteral )
    [ LogicalLine ]
#End ExternalSource
```

The parameters of #ExternalSource directive are the path of external file, line number of the first line, and the line where the error occurred.

Example

The following code demonstrates a hypothetical use of the directive:

```
Module mydirectives
   Public Class ExternalSourceTester


      Sub TestExternalSource()


      #ExternalSource("c:\vbprogs\directives.vb", 5)
         Console.WriteLine("This is External Code. ")
      #End ExternalSource


      End Sub
   End Class


   Sub Main()
      Dim t As New ExternalSourceTester()
      t.TestExternalSource()
      Console.WriteLine("In Main.")
      Console.ReadKey()


   End Sub
```

When the above code is compiled and executed, it produces the following result:

```
This is External Code.
In Main.
```

The #If...Then...#Else Directives

This directive conditionally compiles selected blocks of Visual Basic code.

Syntax for this directive is:

```
#If expression Then
   statements
[ #ElseIf expression Then
   [ statements ]
...
#ElseIf expression Then
   [ statements ] ]
[ #Else
   [ statements ] ]
#End If
```

For example,

```
#Const TargetOS = "Linux"
#If TargetOS = "Windows 7" Then
   ' Windows 7 specific code
#ElseIf TargetOS = "WinXP" Then
   ' Windows XP specific code
#Else
   ' Code for other OS
#End if
```

Example

The following code demonstrates a hypothetical use of the directive:

```
Module mydirectives
#Const classCode = 8

   Sub Main()
   #If classCode = 7 Then
        Console.WriteLine("Exam Questions for Class VII")
   #ElseIf classCode = 8 Then
        Console.WriteLine("Exam Questions for Class VIII")
   #Else
        Console.WriteLine("Exam Questions for Higher Classes")
   #End If
        Console.ReadKey()

    End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Exam Questions for Class VIII
```

The #Region Directive

This directive helps in collapsing and hiding sections of code in Visual Basic files.

Syntax for this directive is:

```
#Region "identifier_string"
#End Region
```

For example,

```
#Region "StatsFunctions"
    ' Insert code for the Statistical functions here.
#End Region
```

VB.Net - Operators

An operator is a symbol that tells the compiler to perform specific mathematical or logical manipulations. VB.Net is rich in built-in operators and provides following types of commonly used operators:

- Arithmetic Operators
- Comparison Operators
- Logical/Bitwise Operators
- Bit Shift Operators
- Assignment Operators
- Miscellaneous Operators

This tutorial will explain the most commonly used operators.

Arithmetic Operators

Following table shows all the arithmetic operators supported by VB.Net. Assume variable **A** holds 2 and variable **B** holds 7, then:

Operator	Description	Example
^	Raises one operand to the power of another	B^A will give 49
+	Adds two operands	A + B will give 9
-	Subtracts second operand from the first	A - B will give -5
*	Multiplies both operands	A * B will give 14

/	Divides one operand by another and returns a floating point result	B / A will give 3.5
\	Divides one operand by another and returns an integer result	B \ A will give 3
MOD	Modulus Operator and remainder of after an integer division	B MOD A will give 1

Comparison Operators

Following table shows all the comparison operators supported by VB.Net. Assume variable **A** holds 10 and variable **B** holds 20, then:

Operator	**Description**	**Example**
=	Checks if the values of two operands are equal or not; if yes, then condition becomes true.	(A = B) is not true.
<>	Checks if the values of two operands are equal or not; if values are not equal, then condition becomes true.	(A <> B) is true.
>	Checks if the value of left operand is greater than the value of right operand; if yes, then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand; if yes, then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand; if yes, then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand; if yes, then condition becomes true.	(A <= B) is true.

Apart from the above, VB.Net provides three more comparison operators, which we will be using in forthcoming chapters; however, we give a brief description here.

- **Is** Operator - It compares two object reference variables and determines if two object references refer to the same object without performing value comparisons. If object1 and object2 both refer to the exact same object instance, result is **True**; otherwise, result is False.
- **IsNot** Operator - It also compares two object reference variables and determines if two object references refer to different objects. If object1 and object2 both refer to the exact same object instance, result is **False**; otherwise, result is True.

- **Like** Operator - It compares a string against a pattern.

Logical/Bitwise Operators

Following table shows all the logical operators supported by VB.Net. Assume variable A holds Boolean value True and variable B holds Boolean value False, then:

Operator	Description	Example
And	It is the logical as well as bitwise AND operator. If both the operands are true, then condition becomes true. This operator does not perform short-circuiting, i.e., it evaluates both the expressions.	(A And B) is False.
Or	It is the logical as well as bitwise OR operator. If any of the two operands is true, then condition becomes true. This operator does not perform short-circuiting, i.e., it evaluates both the expressions.	(A Or B) is True.
Not	It is the logical as well as bitwise NOT operator. Use to reverses the logical state of its operand. If a condition is true, then Logical NOT operator will make false.	Not(A And B) is True.
Xor	It is the logical as well as bitwise Logical Exclusive OR operator. It returns True if both expressions are True or both expressions are False; otherwise it returns False. This operator does not perform short-circuiting, it always evaluates both expressions and there is no short-circuiting counterpart of this operator.	A Xor B is True.
AndAlso	It is the logical AND operator. It works only on Boolean data. It performs short-circuiting.	(A AndAlso B) is False.
OrElse	It is the logical OR operator. It works only on Boolean data. It performs short-circuiting.	(A OrElse B) is True.
IsFalse	It determines whether an expression is False.	
IsTrue	It determines whether an expression is True.	

Bit Shift Operators

We have already discussed the bitwise operators. The bit shift operators perform the shift operations on binary values. Before coming into the bit shift operators, let us understand the bit operations. Bitwise operators work on bits and perform bit-by-bit operations. The truth tables for &, |, and ^ are as follows:

p	**q**	**p & q**	**p | q**	**p ^ q**
0	0	0	0	0
0	1	0	1	1
1	1	1	1	0
1	0	0	1	1

Assume if A = 60; and B = 13; now in binary format they will be as follows:

A = 0011 1100

B = 0000 1101

A&B = 0000 1100

A|B = 0011 1101

A^B = 0011 0001

~A = 1100 0011

We have seen that the Bitwise operators supported by VB.Net are And, Or, Xor and Not. The Bit shift operators are >> and << for left shift and right shift, respectively.

Assume that the variable A holds 60 and variable B holds 13, then:

Operator	**Description**	**Example**
And	Bitwise AND Operator copies a bit to the result if it exists in both operands.	(A AND B) will give 12, which is 0000 1100
Or	Binary OR Operator copies a bit if it exists in either operand.	(A Or B) will give 61, which is 0011 1101
Xor	Binary XOR Operator copies the bit if it is set in one operand but not both.	(A Xor B) will give 49, which is 0011 0001
Not	Binary Ones Complement Operator is unary and has the effect of 'flipping' bits.	(Not A) will give -61, which is 1100 0011 in 2's complement form due to a signed binary number.
<<	Binary Left Shift Operator. The left operands value is moved left by the number of bits specified by the right operand.	A << 2 will give 240, which is 1111 0000
>>	Binary Right Shift Operator. The left operands value is moved right by the number of bits specified by the right operand.	A >> 2 will give 15, which is 0000 1111

Assignment Operators

There are following assignment operators supported by VB.Net:

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	C = A + B will assign value of A + B into C
+=	Add AND assignment operator, It adds right operand to the left operand and assigns the result to left operand	C += A is equivalent to C = C + A
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assigns the result to left operand	C -= A is equivalent to C = C - A
*=	Multiply AND assignment operator, It multiplies right operand with the left operand and assigns the result to left operand	C *= A is equivalent to C = C * A
/=	Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand (floating point division)	C /= A is equivalent to C = C / A
\=	Divide AND assignment operator, It divides left operand with the right operand and assigns the result to left operand (Integer division)	C \= A is equivalent to C = C \A
^=	Exponentiation and assignment operator. It raises the left operand to the power of the right operand and assigns the result to left operand.	C^=A is equivalent to C = C ^ A
<<=	Left shift AND assignment operator	C <<= 2 is same as C = C << 2
>>=	Right shift AND assignment operator	C >>= 2 is same as C = C >> 2
&=	Concatenates a String expression to a String variable or property and assigns the result to the variable or property.	Str1 &= Str2 is same as Str1 = Str1 & Str2

Miscellaneous Operators

There are few other important operators supported by VB.Net.

Operator	Description	Example
AddressOf	Returns the address of a procedure.	AddHandler Button1.Click, AddressOf Button1_Click
Await	It is applied to an operand in an asynchronous method or lambda expression to suspend execution of the method until the awaited task completes.	Dim result As res = Await AsyncMethodThatReturnsResult() Await AsyncMethod()

GetType	It returns a Type object for the specified type. The Type object provides information about the type such as its properties, methods, and events.	`MsgBox(GetType(Integer).ToString())`

Function Expression	It declares the parameters and code that define a function lambda expression.	`Dim add5 = Function(num As Integer) num + 5`
		`'prints 10`
		`Console.WriteLine(add5(5))`

If	It uses short-circuit evaluation to conditionally return one of two values. The If operator can be called with three arguments or with two arguments.	`Dim num = 5`
		`Console.WriteLine(If(num >= 0, "Positive", "Negative"))`

Operators Precedence in VB.Net

Operator precedence determines the grouping of terms in an expression. This affects how an expression is evaluated. Certain operators have higher precedence than others; for example, the multiplication operator has higher precedence than the addition operator:

For example, x = 7 + 3 * 2; here, x is assigned 13, not 20 because operator * has higher precedence than +, so it first gets multiplied with 3*2 and then adds into 7.

Here, operators with the highest precedence appear at the top of the table, those with the lowest appear at the bottom. Within an expression, higher precedence operators will be evaluated first.

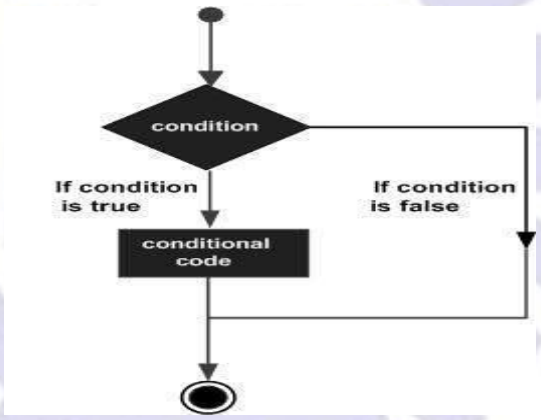
Operator	Precedence
Await	Highest
Exponentiation (^)	
Unary identity and negation (+, -)	
Multiplication and floating-point division (*, /)	
Integer division (\)	
Modulus arithmetic (Mod)	
Addition and subtraction (+, -)	
Arithmetic bit shift (<<, >>)	
All comparison operators (=, <>, <, <=, >, >=, Is, IsNot, Like, TypeOf...Is)	
Negation (Not)	
Conjunction (And, AndAlso)	

Inclusive disjunction (Or, OrElse)	
Exclusive disjunction (Xor)	Lowest

VB.Net - Decision Making

Decision making structures require that the programmer specify one or more conditions to be evaluated or tested by the program, along with a statement or statements to be executed if the condition is determined to be true, and optionally, other statements to be executed if the condition is determined to be false.

Following is the general form of a typical decision making structure found in most of the programming languages:



VB.Net provides the following types of decision making statements. Click the following links to check their details.

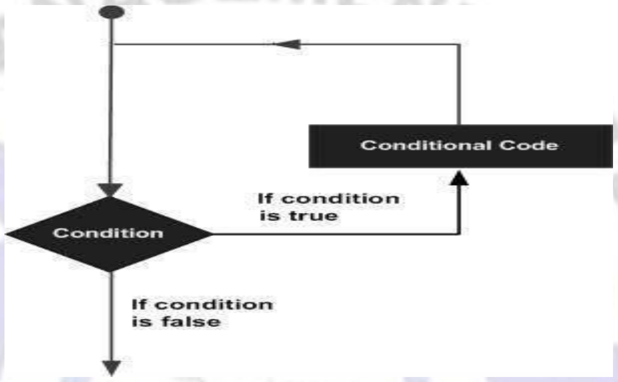
Statement	Description
If ... Then statement	An **If...Then statement** consists of a boolean expression followed by one or more statements.
If...Then...Else statement	An **If...Then statement** can be followed by an optional **Else statement**, which executes when the boolean expression is false.
nested If statements	You can use one **If** or **Else if** statement inside another **If** or **Else if** statement(s).
Select Case statement	A **Select Case** statement allows a variable to be tested for equality against a list of values.
nested Select Case statements	You can use one **select case** statement inside another **select case** statement(s).

VB.Net - Loops

There may be a situation when you need to execute a block of code several number of times. In general, statements are executed sequentially: The first statement in a function is executed first, followed by the second, and so on.

Programming languages provide various control structures that allow for more complicated execution paths.

A loop statement allows us to execute a statement or group of statements multiple times and following is the general form of a loop statement in most of the programming languages:



VB.Net provides following types of loops to handle looping requirements. Click the following links to check their details.

Loop Type	**Description**
Do Loop	It repeats the enclosed block of statements while a Boolean condition is True or until the condition becomes True. It could be terminated at any time with the Exit Do statement.
For...Next	It repeats a group of statements a specified number of times and a loop index counts the number of loop iterations as the loop executes.
For Each...Next	It repeats a group of statements for each element in a collection. This loop is used for accessing and manipulating all elements in an array or a VB.Net collection.
While... End While	It executes a series of statements as long as a given condition is True.
With... End With	It is not exactly a looping construct. It executes a series of statements that repeatedly refer to a single object or structure.
Nested loops	You can use one or more loops inside any another While, For or Do loop.

Loop Control Statements:

Loop control statements change execution from its normal sequence. When execution leaves a scope, all automatic objects that were created in that scope are destroyed.

VB.Net provides the following control statements. Click the following links to check their details.

Control Statement	Description
Exit statement	Terminates the **loop** or **select case** statement and transfers execution to the statement immediately following the loop or select case.
Continue statement	Causes the loop to skip the remainder of its body and immediately retest its condition prior to reiterating.
GoTo statement	Transfers control to the labeled statement. Though it is not advised to use GoTo statement in your program.

VB.Net - Strings

In VB.Net, you can use strings as array of characters, however, more common practice is to use the String keyword to declare a string variable. The string keyword is an alias for the **System.String** class.

Creating a String Object

You can create string object using one of the following methods:

- By assigning a string literal to a String variable
- By using a String class constructor
- By using the string concatenation operator (+)
- By retrieving a property or calling a method that returns a string
- By calling a formatting method to convert a value or object to its string representation

The following example demonstrates this:

```
Module strings
  Sub Main()
    Dim fname, lname, fullname, greetings As String
    fname = "Rowan"
    lname = "Atkinson"
    fullname = fname + " " + lname
    Console.WriteLine("Full Name: {0}", fullname)
   'by using string constructor
    Dim letters As Char() = {"H", "e", "l", "l", "o"}
    greetings = New String(letters)
    Console.WriteLine("Greetings: {0}", greetings)
    'methods returning String
    Dim sarray() As String = {"Hello", "From", "Tutorials", "Point"}
    Dim message As String = String.Join(" ", sarray)
```

```
      Console.WriteLine("Message: {0}", message)
      'formatting method to convert a value
      Dim waiting As DateTime = New DateTime(2012, 12, 12, 17, 58, 1)
      Dim chat As String = String.Format("Message sent at {0:t} on {0:D}", waiting)
      Console.WriteLine("Message: {0}", chat)
      Console.ReadLine()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Full Name: Rowan Atkinson
Greetings: Hello
Message: Hello From Tutorials Point
Message: Message sent at 5:58 PM on Wednesday, December 12, 2012
```

Properties of the String Class

The String class has the following two properties:

S.N	Property Name & Description
1	**Chars** Gets the *Char* object at a specified position in the current *String* object.
2	**Length** Gets the number of characters in the current String object.

Methods of the String Class

The String class has numerous methods that help you in working with the string objects. The following table provides some of the most commonly used methods:

S.N	Method Name & Description
1	**Public Shared Function Compare (strA As String, strB As String) As Integer** Compares two specified string objects and returns an integer that indicates their relative position in the sort order.
2	**Public Shared Function Compare (strA As String, strB As String, ignoreCase As Boolean) As Integer** Compares two specified string objects and returns an integer that indicates their relative position in the sort order. However, it ignores case if the Boolean parameter is true.
3	**Public Shared Function Concat (str0 As String, str1 As String) As String** Concatenates two string objects.

4	**Public Shared Function Concat (str0 As String, str1 As String, str2 As String) As String** Concatenates three string objects.
5	**Public Shared Function Concat (str0 As String, str1 As String, str2 As String, str3 As String) As String** Concatenates four string objects.
6	**Public Function Contains (value As String) As Boolean** Returns a value indicating whether the specified string object occurs within this string.
7	**Public Shared Function Copy (str As String) As String** Creates a new String object with the same value as the specified string.
8	**pPublic Sub CopyTo (sourceIndex As Integer, destination As Char(), destinationIndex As Integer, count As Integer)** Copies a specified number of characters from a specified position of the string object to a specified position in an array of Unicode characters.
9	**Public Function EndsWith (value As String) As Boolean** Determines whether the end of the string object matches the specified string.
10	**Public Function Equals (value As String) As Boolean** Determines whether the current string object and the specified string object have the same value.
11	**Public Shared Function Equals (a As String, b As String) As Boolean** Determines whether two specified string objects have the same value.
12	**Public Shared Function Format (format As String, arg0 As Object) As String** Replaces one or more format items in a specified string with the string representation of a specified object.
13	**Public Function IndexOf (value As Char) As Integer** Returns the zero-based index of the first occurrence of the specified Unicode character in the current string.
14	**Public Function IndexOf (value As String) As Integer** Returns the zero-based index of the first occurrence of the specified string in this instance.
15	**Public Function IndexOf (value As Char, startIndex As Integer) As Integer** Returns the zero-based index of the first occurrence of the specified Unicode character in this string, starting search at the specified character position.
16	**Public Function IndexOf (value As String, startIndex As Integer) As Integer** Returns the zero-based index of the first occurrence of the specified string in this instance, starting search at the specified character position.
17	**Public Function IndexOfAny (anyOf As Char()) As Integer** Returns the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters.
18	**Public Function IndexOfAny (anyOf As Char(), startIndex As Integer) As**

	Integer Returns the zero-based index of the first occurrence in this instance of any character in a specified array of Unicode characters, starting search at the specified character position.
19	**Public Function Insert (startIndex As Integer, value As String) As String** Returns a new string in which a specified string is inserted at a specified index position in the current string object.
20	**Public Shared Function IsNullOrEmpty (value As String) As Boolean** Indicates whether the specified string is null or an Empty string.
21	**Public Shared Function Join (separator As String, ParamArray value As String()) As String** Concatenates all the elements of a string array, using the specified separator between each element.
22	**Public Shared Function Join (separator As String, value As String(), startIndex As Integer, count As Integer) As String** Concatenates the specified elements of a string array, using the specified separator between each element.
23	**Public Function LastIndexOf (value As Char) As Integer** Returns the zero-based index position of the last occurrence of the specified Unicode character within the current string object.
24	**Public Function LastIndexOf (value As String) As Integer** Returns the zero-based index position of the last occurrence of a specified string within the current string object.
25	**Public Function Remove (startIndex As Integer) As String** Removes all the characters in the current instance, beginning at a specified position and continuing through the last position, and returns the string.
26	**Public Function Remove (startIndex As Integer, count As Integer) As String** Removes the specified number of characters in the current string beginning at a specified position and returns the string.
27	**Public Function Replace (oldChar As Char, newChar As Char) As String** Replaces all occurrences of a specified Unicode character in the current string object with the specified Unicode character and returns the new string.
28	**Public Function Replace (oldValue As String, newValue As String) As String** Replaces all occurrences of a specified string in the current string object with the specified string and returns the new string.
29	**Public Function Split (ParamArray separator As Char()) As String()** Returns a string array that contains the substrings in the current string object, delimited by elements of a specified Unicode character array.
30	**Public Function Split (separator As Char(), count As Integer) As String()** Returns a string array that contains the substrings in the current string object, delimited by elements of a specified Unicode character array. The int parameter specifies the maximum number of substrings to return.

31	**Public Function StartsWith (value As String) As Boolean** Determines whether the beginning of this string instance matches the specified string.
32	**Public Function ToCharArray As Char()** Returns a Unicode character array with all the characters in the current string object.
33	**Public Function ToCharArray (startIndex As Integer, length As Integer) As Char()** Returns a Unicode character array with all the characters in the current string object, starting from the specified index and up to the specified length.
34	**Public Function ToLower As String** Returns a copy of this string converted to lowercase.
35	**Public Function ToUpper As String** Returns a copy of this string converted to uppercase.
36	**Public Function Trim As String** Removes all leading and trailing white-space characters from the current String object.

The above list of methods is not exhaustive, please visit MSDN library for the complete list of methods and String class constructors.

Examples:

The following example demonstrates some of the methods mentioned above:

Comparing Strings:

```
Module strings
  Sub Main()
    Dim str1, str2 As String
    str1 = "This is test"
    str2 = "This is text"
    If (String.Compare(str1, str2) = 0) Then
        Console.WriteLine(str1 + " and " + str2 +
                    " are equal.")
    Else
        Console.WriteLine(str1 + " and " + str2 +
                    " are not equal.")
    End If
    Console.ReadLine()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
This is test and This is text are not equal.
```

String Contains String:

```
Module strings
  Sub Main()
    Dim str1 As String
    str1 = "This is test"
    If (str1.Contains("test")) Then
       Console.WriteLine("The sequence 'test' was found.")
    End If
    Console.ReadLine()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
The sequence 'test' was found.
```

Getting a Substring:

```
Module strings
  Sub Main()
    Dim str As String
    str = "Last night I dreamt of San Pedro"
    Console.WriteLine(str)
    Dim substr As String = str.Substring(23)
    Console.WriteLine(substr)
    Console.ReadLine()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Last night I dreamt of San Pedro
San Pedro.
```

Joining Strings:

```
Module strings
  Sub Main()
    Dim strarray As String() = {"Down the way where the nights are gay",
                     "And the sun shines daily on the mountain top",
                      "I took a trip on a sailing ship",
                     "And when I reached Jamaica",
                     "I made a stop"}
    Dim str As String = String.Join(vbCrLf, strarray)
    Console.WriteLine(str)
    Console.ReadLine()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Down the way where the nights are gay
And the sun shines daily on the mountain top
I took a trip on a sailing ship
And when I reached Jamaica
I made a stop
```

VB.Net - Date & Time

Most of the softwares you write need implementing some form of date functions returning current date and time. Dates are so much part of everyday life that it becomes easy to work with them without thinking. VB.Net also provides powerful tools for date arithmetic that makes manipulating dates easy.

The **Date** data type contains date values, time values, or date and time values. The default value of Date is 0:00:00 (midnight) on January 1, 0001. The equivalent .NET data type is **System.DateTime**.

The **DateTime** structure represents an instant in time, typically expressed as a date and time of day

```
'Declaration
<SerializableAttribute> _
Public Structure DateTime _
          Implements IComparable, IFormattable, IConvertible, ISerializable,
          IComparable(Of DateTime), IEquatable(Of DateTime)
```

You can also get the current date and time from the DateAndTime class.

The **DateAndTime** module contains the procedures and properties used in date and time operations.

```
'Declaration
<StandardModuleAttribute> _
Public NotInheritable Class DateAndTime
```

Note:

Both the DateTime structure and the DateAndTime module contain properties like **Now** and **Today**, so often beginners find it confusing. The DateAndTime class belongs to the Microsoft.VisualBasic namespace and the DateTime structure belongs to the System namespace.

Therefore, using the later would help you in porting your code to another .Net language like C#. However, the DateAndTime class/module contains all the legacy date functions available in Visual Basic.

Properties and Methods of the DateTime Structure

The following table lists some of the commonly used **properties** of the **DateTime** Structure:

S.N	Property	Description
1	**Date**	Gets the date component of this instance.
2	**Day**	Gets the day of the month represented by this instance.
3	**DayOfWeek**	Gets the day of the week represented by this instance.
4	**DayOfYear**	Gets the day of the year represented by this instance.
5	**Hour**	Gets the hour component of the date represented by this instance.
6	**Kind**	Gets a value that indicates whether the time represented by this instance is based on local time, Coordinated Universal Time (UTC), or neither.
7	**Millisecond**	Gets the milliseconds component of the date represented by this instance.
8	**Minute**	Gets the minute component of the date represented by this instance.
9	**Month**	Gets the month component of the date represented by this instance.
10	**Now**	Gets a **DateTime** object that is set to the current date and time on this computer, expressed as the local time.
11	**Second**	Gets the seconds component of the date represented by this instance.
12	**Ticks**	Gets the number of ticks that represent the date and time of this instance.
13	**TimeOfDay**	Gets the time of day for this instance.
14	**Today**	Gets the current date.

15	**UtcNow**	Gets a **DateTime** object that is set to the current date and time on this computer, expressed as the Coordinated Universal Time (UTC).
16	**Year**	Gets the year component of the date represented by this instance.

The following table lists some of the commonly used **methods** of the **DateTime** structure:

S.N	Method Name & Description
1	**Public Function Add (value As TimeSpan) As DateTime** Returns a new DateTime that adds the value of the specified TimeSpan to the value of this instance.
2	**Public Function AddDays (value As Double) As DateTime** Returns a new DateTime that adds the specified number of days to the value of this instance.
3	**Public Function AddHours (value As Double) As DateTime** Returns a new DateTime that adds the specified number of hours to the value of this instance.
4	**Public Function AddMinutes (value As Double) As DateTime** Returns a new DateTime that adds the specified number of minutes to the value of this instance.
5	**Public Function AddMonths (months As Integer) As DateTime** Returns a new DateTime that adds the specified number of months to the value of this instance.
6	**Public Function AddSeconds (value As Double) As DateTime** Returns a new DateTime that adds the specified number of seconds to the value of this instance.
7	**Public Function AddYears (value As Integer) As DateTime** Returns a new DateTime that adds the specified number of years to the value of this instance.
8	**Public Shared Function Compare (t1 As DateTime,t2 As DateTime) As Integer** Compares two instances of DateTime and returns an integer that indicates whether the first instance is earlier than, the same as, or later than the second instance.
9	**Public Function CompareTo (value As DateTime) As Integer** Compares the value of this instance to a specified DateTime value and returns an integer that indicates whether this instance is earlier than, the same as, or later than the specified DateTime value.
10	**Public Function Equals (value As DateTime) As Boolean** Returns a value indicating whether the value of this instance is equal to the value of the specified DateTime instance.
11	**Public Shared Function Equals (t1 As DateTime, t2 As DateTime) As Boolean** Returns a value indicating whether two DateTime instances have the same date and time

	value.
12	**Public Overrides Function ToString As String** Converts the value of the current DateTime object to its equivalent string representation.

The above list of methods is not exhaustive, please visit Microsoft documentation for the complete list of methods and properties of the DateTime structure.

Creating a DateTime Object

You can create a DateTime object in one of the following ways:

- By calling a DateTime constructor from any of the overloaded DateTime constructors.
- By assigning the DateTime object a date and time value returned by a property or method.
- By parsing the string representation of a date and time value.
- By calling the DateTime structure's implicit default constructor.

The following example demonstrates this:

```
Module Module1
  Sub Main()
    'DateTime constructor: parameters year, month, day, hour, min, sec
    Dim date1 As New Date(2012, 12, 16, 12, 0, 0)
    'initializes a new DateTime value
    Dim date2 As Date = #12/16/2012 12:00:52 AM#
    'using properties
    Dim date3 As Date = Date.Now
    Dim date4 As Date = Date.UtcNow
    Dim date5 As Date = Date.Today
    Console.WriteLine(date1)
    Console.WriteLine(date2)
    Console.WriteLine(date3)
    Console.WriteLine(date4)
    Console.WriteLine(date5)
    Console.ReadKey()
  End Sub
End Module
```

When the above code was compiled and executed, it produces the following result:

```
12/16/2012 12:00:00 PM
```

```
12/16/2012 12:00:52 PM
12/12/2012 10:22:50 PM
12/12/2012 12:00:00 PM
```

Getting the Current Date and Time:

The following programs demonstrate how to get the current date and time in VB.Net:

Current Time:

```
Module dateNtime
   Sub Main()
      Console.Write("Current Time: ")
      Console.WriteLine(Now.ToLongTimeString)
      Console.ReadKey()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Current Time: 11 :05 :32 AM
```

Current Date:

```
Module dateNtime
   Sub Main()
      Console.WriteLine("Current Date: ")
      Dim dt As Date = Today
      Console.WriteLine("Today is: {0}", dt)
      Console.ReadKey()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Today is: 12/11/2012 12:00:00 AM
```

Formatting Date

A Date literal should be enclosed within hash signs (# #), and specified in the format M/d/yyyy, for example #12/16/2012#. Otherwise, your code may change depending on the locale in which your application is running.

For example, you specified Date literal of #2/6/2012# for the date February 6, 2012. It is alright for the locale that uses mm/dd/yyyy format. However, in a locale that uses dd/mm/yyyy format, your literal would compile to June 2, 2012. If a locale uses another format say, yyyy/mm/dd, the literal would be invalid and cause a compiler error.

To convert a Date literal to the format of your locale or to a custom format, use the **Format** function of String class, specifying either a predefined or user-defined date format.
The following example demonstrates this.

```
Module dateNtime
  Sub Main()
    Console.WriteLine("India Wins Freedom: ")
    Dim independenceDay As New Date(1947, 8, 15, 0, 0, 0)
    ' Use format specifiers to control the date display.
    Console.WriteLine(" Format 'd:' " & independenceDay.ToString("d"))
    Console.WriteLine(" Format 'D:' " & independenceDay.ToString("D"))
    Console.WriteLine(" Format 't:' " & independenceDay.ToString("t"))
    Console.WriteLine(" Format 'T:' " & independenceDay.ToString("T"))
    Console.WriteLine(" Format 'f:' " & independenceDay.ToString("f"))
    Console.WriteLine(" Format 'F:' " & independenceDay.ToString("F"))
    Console.WriteLine(" Format 'g:' " & independenceDay.ToString("g"))
    Console.WriteLine(" Format 'G:' " & independenceDay.ToString("G"))
    Console.WriteLine(" Format 'M:' " & independenceDay.ToString("M"))
    Console.WriteLine(" Format 'R:' " & independenceDay.ToString("R"))
    Console.WriteLine(" Format 'y:' " & independenceDay.ToString("y"))
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
India Wins Freedom:
Format 'd:' 8/15/1947
Format 'D:' Friday, August 15, 1947
Format 't:' 12:00 AM
Format 'T:' 12:00:00 AM
Format 'f:' Friday, August 15, 1947 12:00 AM
Format 'F:' Friday, August 15, 1947 12:00:00 AM
Format 'g:' 8/15/1947 12:00 AM
Format 'G:' 8/15/1947 12:00:00 AM
Format 'M:' 8/15/1947 August 15
Format 'R:' Fri, 15 August 1947 00:00:00 GMT
Format 'y:' August, 1947
```

Predefined Date/Time Formats

The following table identifies the predefined date and time format names. These may be used by name as the style argument for the **Format** function:

Format	**Description**
General Date, or G	Displays a date and/or time. For example, 1/12/2012 07:07:30 AM.
Long Date,Medium Date, or D	Displays a date according to your current culture's long date format. For example, Sunday, December 16, 2012.
Short Date, or d	Displays a date using your current culture's short date format. For example, 12/12/2012.
Long Time,Medium Time, orT	Displays a time using your current culture's long time format; typically includes hours, minutes, seconds. For example, 01:07:30 AM.
Short Time or t	Displays a time using your current culture's short time format. For example, 11:07 AM.
f	Displays the long date and short time according to your current culture's format. For example, Sunday, December 16, 2012 12:15 AM.
F	Displays the long date and long time according to your current culture's format. For example, Sunday, December 16, 2012 12:15:31 AM.
g	Displays the short date and short time according to your current culture's format. For example, 12/16/2012 12:15 AM.
M, m	Displays the month and the day of a date. For example, December 16.
R, r	Formats the date according to the RFC1123Pattern property.
s	Formats the date and time as a sortable index. For example, 2012-12-16T12:07:31.
u	Formats the date and time as a GMT sortable index. For example, 2012-12-16 12:15:31Z.
U	Formats the date and time with the long date and long time as GMT. For example, Sunday, December 16, 2012 6:07:31 PM.
Y, y	Formats the date as the year and month. For example, December, 2012.

For other formats like user-defined formats, please consult Microsoft Documentation.

Properties and Methods of the DateAndTime Class

The following table lists some of the commonly used **properties** of the **DateAndTime** Class:

S.N	Property	Description
1	**Date**	Returns or sets a String value representing the current date according to your system.
2	**Now**	Returns a Date value containing the current date and time according to your system.
3	**TimeOfDay**	Returns or sets a Date value containing the current time of day according to your system.
4	**Timer**	Returns a Double value representing the number of seconds elapsed since midnight.
5	**TimeString**	Returns or sets a String value representing the current time of day according to your system.
6	**Today**	Gets the current date.

The following table lists some of the commonly used **methods** of the **DateAndTime** class:

S.N	Method Name & Description
1	**Public Shared Function DateAdd (Interval As DateInterval, Number As Double, DateValue As DateTime) As DateTime** Returns a Date value containing a date and time value to which a specified time interval has been added.
2	**Public Shared Function DateAdd (Interval As String,Number As Double,DateValue As Object) As DateTime** Returns a Date value containing a date and time value to which a specified time interval has been added.
3	**Public Shared Function DateDiff (Interval As DateInterval, Date1 As DateTime, Date2 As DateTime, DayOfWeek As FirstDayOfWeek, WeekOfYear As FirstWeekOfYear) As Long** Returns a Long value specifying the number of time intervals between two Date values.
4	**Public Shared Function DatePart (Interval As DateInterval, DateValue As DateTime, FirstDayOfWeekValue As FirstDayOfWeek, FirstWeekOfYearValue As FirstWeekOfYear) As Integer** Returns an Integer value containing the specified component of a given Date value.
5	**Public Shared Function Day (DateValue As DateTime) As Integer** Returns an Integer value from 1 through 31 representing the day of the month.
6	**Public Shared Function Hour (TimeValue As DateTime) As Integer** Returns an Integer value from 0 through 23 representing the hour of the day.
7	**Public Shared Function Minute (TimeValue As DateTime) As Integer** Returns an Integer value from 0 through 59 representing the minute of the hour.

8	**Public Shared Function Month (DateValue As DateTime) As Integer** Returns an Integer value from 1 through 12 representing the month of the year.
9	**Public Shared Function MonthName (Month As Integer, Abbreviate As Boolean) As String** Returns a String value containing the name of the specified month.
10	**Public Shared Function Second (TimeValue As DateTime) As Integer** Returns an Integer value from 0 through 59 representing the second of the minute.
11	**Public Overridable Function ToString As String** Returns a string that represents the current object.
12	**Public Shared Function Weekday (DateValue As DateTime, DayOfWeek As FirstDayOfWeek) As Integer** Returns an Integer value containing a number representing the day of the week.
13	**Public Shared Function WeekdayName (Weekday As Integer, Abbreviate As Boolean, FirstDayOfWeekValue As FirstDayOfWeek) As String** Returns a String value containing the name of the specified weekday.
14	**Public Shared Function Year (DateValue As DateTime) As Integer** Returns an Integer value from 1 through 9999 representing the year.

The above list is not exhaustive. For complete list of properties and methods of the DateAndTime class, please consult Microsoft Documentation.

The following program demonstrates some of these and methods:

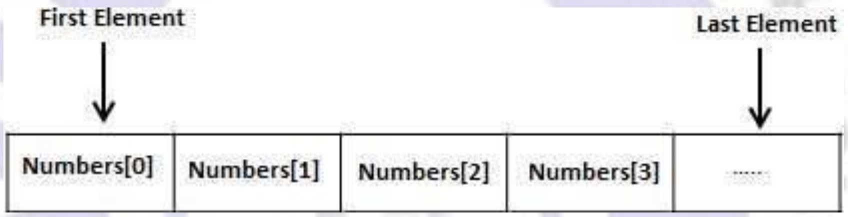
```
Module Module1
  Sub Main()
    Dim birthday As Date
    Dim bday As Integer
    Dim month As Integer
    Dim monthname As String
    ' Assign a date using standard short format.
    birthday = #7/27/1998#
    bday = Microsoft.VisualBasic.DateAndTime.Day(birthday)
    month = Microsoft.VisualBasic.DateAndTime.Month(birthday)
    monthname = Microsoft.VisualBasic.DateAndTime.MonthName(month)
    Console.WriteLine(birthday)
    Console.WriteLine(bday)
    Console.WriteLine(month)
    Console.WriteLine(monthname)
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
7/27/1998 12:00:00 AM
27
7
July
```

VB.Net - Arrays

An array stores a fixed-size sequential collection of elements of the same type. An array is used to store a collection of data, but it is often more useful to think of an array as a collection of variables of the same type. All arrays consist of contiguous memory locations. The lowest address corresponds to the first element and the highest address to the last element.



Creating Arrays in VB.Net

To declare an array in VB.Net, you use the Dim statement. For example,

```
Dim intData(30)      ' an array of 31 elements
Dim strData(20) As String   ' an array of 21 strings
Dim twoDarray(10, 20) As Integer     'a two dimensional array of integers
Dim ranges(10, 100)             'a two dimensional array
```

You can also initialize the array elements while declaring the array. For example,

```
Dim intData() As Integer = {12, 16, 20, 24, 28, 32}
Dim names() As String = {"Karthik", "Sandhya", _
"Shivangi", "Ashwitha", "Somnath"}
Dim miscData() As Object = {"Hello World", 12d, 16ui, "A"c}
```

The elements in an array can be stored and accessed by using the index of the array. The following program demonstrates this:

```
Module arrayApl
   Sub Main()
      Dim n(10) As Integer  ' n is an array of 11 integers '
      Dim i, j As Integer
      ' initialize elements of array n '
```

```
    For i = 0 To 10
        n(i) = i + 100 ' set element at location i to i + 100
    Next i
    ' output each array element's value '
    For j = 0 To 10
        Console.WriteLine("Element({0}) = {1}", j, n(j))
    Next j
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Element(0) = 100
Element(1) = 101
Element(2) = 102
Element(3) = 103
Element(4) = 104
Element(5) = 105
Element(6) = 106
Element(7) = 107
Element(8) = 108
Element(9) = 109
Element(10) = 110
```

Dynamic Arrays

Dynamic arrays are arrays that can be dimensioned and re-dimensioned as par the need of the program. You can declare a dynamic array using the **ReDim** statement.

Syntax for ReDim statement:

```
ReDim [Preserve] arrayname(subscripts)
```

Where,

- The **Preserve** keyword helps to preserve the data in an existing array, when you resize it.
- **arrayname** is the name of the array to re-dimension.
- **subscripts** specifies the new dimension.

```
Module arrayApl
  Sub Main()
    Dim marks() As Integer
    ReDim marks(2)
    marks(0) = 85
    marks(1) = 75
    marks(2) = 90
    ReDim Preserve marks(10)
    marks(3) = 80
    marks(4) = 76
    marks(5) = 92
    marks(6) = 99
    marks(7) = 79
    marks(8) = 75
    For i = 0 To 10
      Console.WriteLine(i & vbTab & marks(i))
    Next i
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
0       85
1       75
2       90
3       80
4       76
5       92
6       99
7       79
8       75
9       0
10      0
```

Multi-Dimensional Arrays

VB.Net allows multidimensional arrays. Multidimensional arrays are also called rectangular arrays. You can declare a 2-dimensional array of strings as:

```
Dim twoDStringArray(10, 20) As String
```

or, a 3-dimensional array of Integer variables:

```
Dim threeDIntArray(10, 10, 10) As Integer
```

The following program demonstrates creating and using a 2-dimensional array:

```
Module arrayApl
  Sub Main()
    ' an array with 5 rows and 2 columns
    Dim a(,) As Integer = {{0, 0}, {1, 2}, {2, 4}, {3, 6}, {4, 8}}
    Dim i, j As Integer
    ' output each array element's value '
    For i = 0 To 4
       For j = 0 To 1
          Console.WriteLine("a[{0},{1}] = {2}", i, j, a(i, j))
       Next j
    Next i
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
a[0,0]: 0
a[0,1]: 0
a[1,0]: 1
a[1,1]: 2
a[2,0]: 2
a[2,1]: 4
a[3,0]: 3
a[3,1]: 6
a[4,0]: 4
a[4,1]: 8
```

Jagged Array

A Jagged array is an array of arrays. The follwoing code shows declaring a jagged array named *scores* of Integers:

```
Dim scores As Integer()() = New Integer(5)(){}
```

The following example illustrates using a jagged array:

```
Module arrayApl
  Sub Main()
```

```
    'a jagged array of 5 array of integers
    Dim a As Integer()() = New Integer(4)() {}
    a(0) = New Integer() {0, 0}
    a(1) = New Integer() {1, 2}
    a(2) = New Integer() {2, 4}
    a(3) = New Integer() {3, 6}
    a(4) = New Integer() {4, 8}
    Dim i, j As Integer
    ' output each array element's value
    For i = 0 To 4
      For j = 0 To 1
        Console.WriteLine("a[{0},{1}] = {2}", i, j, a(i)(j))
      Next j
    Next i
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
a[0][0]: 0
a[0][1]: 0
a[1][0]: 1
a[1][1]: 2
a[2][0]: 2
a[2][1]: 4
a[3][0]: 3
a[3][1]: 6
a[4][0]: 4
a[4][1]: 8
```

The Array Class

The Array class is the base class for all the arrays in VB.Net. It is defined in the System namespace. The Array class provides various properties and methods to work with arrays.

Properties of the Array Class

The following table provides some of the most commonly used **properties** of the **Array** class:

S.N	Property Name & Description
1	**IsFixedSize** Gets a value indicating whether the Array has a fixed size.
2	**IsReadOnly** Gets a value indicating whether the Array is read-only.
3	**Length** Gets a 32-bit integer that represents the total number of elements in all the dimensions of the Array.
4	**LongLength** Gets a 64-bit integer that represents the total number of elements in all the dimensions of the Array.
5	**Rank** Gets the rank (number of dimensions) of the Array.

Methods of the Array Class

The following table provides some of the most commonly used **methods** of the **Array** class:

S.N	Method Name & Description
1	**Public Shared Sub Clear (array As Array, index As Integer, length As Integer)** Sets a range of elements in the Array to zero, to false, or to null, depending on the element type.
2	**Public Shared Sub Copy (sourceArray As Array, destinationArray As Array, length As Integer)** Copies a range of elements from an Array starting at the first element and pastes them into another Array starting at the first element. The length is specified as a 32-bit integer.
3	**Public Sub CopyTo (array As Array, index As Integer)** Copies all the elements of the current one-dimensional Array to the specified one-dimensional Array starting at the specified destination Array index. The index is specified as a 32-bit integer.
4	**Public Function GetLength (dimension As Integer) As Integer** Gets a 32-bit integer that represents the number of elements in the specified dimension of the Array.
5	**Public Function GetLongLength (dimension As Integer) As Long** Gets a 64-bit integer that represents the number of elements in the specified dimension of the Array.
6	**Public Function GetLowerBound (dimension As Integer) As Integer**

	Gets the lower bound of the specified dimension in the Array.
7	**Public Function GetType As Type** Gets the Type of the current instance (Inherited from Object).
8	**Public Function GetUpperBound (dimension As Integer) As Integer** Gets the upper bound of the specified dimension in the Array.
9	**Public Function GetValue (index As Integer) As Object** Gets the value at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.
10	**Public Shared Function IndexOf (array As Array,value As Object) As Integer** Searches for the specified object and returns the index of the first occurrence within the entire one-dimensional Array.
11	**Public Shared Sub Reverse (array As Array)** Reverses the sequence of the elements in the entire one-dimensional Array.
12	**Public Sub SetValue (value As Object, index As Integer)** Sets a value to the element at the specified position in the one-dimensional Array. The index is specified as a 32-bit integer.
13	**Public Shared Sub Sort (array As Array)** Sorts the elements in an entire one-dimensional Array using the IComparable implementation of each element of the Array.
14	**Public Overridable Function ToString As String** Returns a string that represents the current object (Inherited from Object).

For complete list of Array class properties and methods, please consult Microsoft documentation.

Example

The following program demonstrates use of some of the methods of the Array class:

```
Module arrayApl
  Sub Main()
    Dim list As Integer() = {34, 72, 13, 44, 25, 30, 10}
    Dim temp As Integer() = list
    Dim i As Integer
    Console.Write("Original Array: ")
    For Each i In list
       Console.Write("{0} ", i)
    Next i
    Console.WriteLine()
```

```
      ' reverse the array
      Array.Reverse(temp)
      Console.Write("Reversed Array: ")
      For Each i In temp
         Console.Write("{0} ", i)
      Next i
      Console.WriteLine()
      'sort the array
      Array.Sort(list)
      Console.Write("Sorted Array: ")
      For Each i In list
         Console.Write("{0} ", i)
      Next i
      Console.WriteLine()
      Console.ReadKey()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Original Array: 34 72 13 44 25 30 10
Reversed Array: 10 30 25 44 13 72 34
Sorted Array: 10 13 25 30 34 44 72
```

VB.Net - Collections

Collection classes are specialized classes for data storage and retrieval. These classes provide support for stacks, queues, lists, and hash tables. Most collection classes implement the same interfaces. Collection classes serve various purposes, such as allocating memory dynamically to elements and accessing a list of items on the basis of an index, etc. These classes create collections of objects of the Object class, which is the base class for all data types in VB.Net.

Various Collection Classes and Their Usage

The following are the various commonly used classes of the **System.Collection** namespace. Click the following links to check their details.

Class	**Description and Useage**
ArrayList	It represents ordered collection of an object that can be **indexed** individually.

	It is basically an alternative to an array. However, unlike array, you can add and remove items from a list at a specified position using an **index** and the array resizes itself automatically. It also allows dynamic memory allocation, add, search and sort items in the list.
Hashtable	It uses a **key** to access the elements in the collection. A hash table is used when you need to access elements by using key, and you can identify a useful key value. Each item in the hash table has a **key/value** pair. The key is used to access the items in the collection.
SortedList	It uses a **key** as well as an **index** to access the items in a list. A sorted list is a combination of an array and a hash table. It contains a list of items that can be accessed using a key or an index. If you access items using an index, it is an ArrayList, and if you access items using a key, it is a Hashtable. The collection of items is always sorted by the key value.
Stack	It represents a **last-in, first out** collection of object. It is used when you need a last-in, first-out access of items. When you add an item in the list, it is called **pushing** the item, and when you remove it, it is called **popping** the item.
Queue	It represents a **first-in, first out** collection of object. It is used when you need a first-in, first-out access of items. When you add an item in the list, it is called **enqueue**, and when you remove an item, it is called **deque**.
BitArray	It represents an array of the **binary representation** using the values 1 and 0. It is used when you need to store the bits but do not know the number of bits in advance. You can access items from the BitArray collection by using an **integer index**, which starts from zero.

VB.Net - Functions

A procedure is a group of statements that together perform a task when called. After the procedure is executed, the control returns to the statement calling the procedure. VB.Net has two types of procedures:

- Functions
- Sub procedures or Subs

Functions return a value, whereas Subs do not return a value.

Defining a Function

The Function statement is used to declare the name, parameter and the body of a function. The syntax for the Function statement is:

```
[Modifiers] Function FunctionName [(ParameterList)] As ReturnType
    [Statements]
End Function
```

Where,

- ***Modifiers***: specify the access level of the function; possible values are: Public, Private, Protected, Friend, Protected Friend and information regarding overloading, overriding, sharing, and shadowing.
- ***FunctionName***: indicates the name of the function
- ***ParameterList***: specifies the list of the parameters
- ***ReturnType***: specifies the data type of the variable the function returns

Example

Following code snippet shows a function *FindMax* that takes two integer values and returns the larger of the two.

```
Function FindMax(ByVal num1 As Integer, ByVal num2 As Integer) As Integer
   ' local variable declaration */
   Dim result As Integer
   If (num1 > num2) Then
       result = num1
   Else
       result = num2
   End If
   FindMax = result
End Function
```

Function Returning a Value

In VB.Net, a function can return a value to the calling code in two ways:

- By using the return statement
- By assigning the value to the function name

The following example demonstrates using the *FindMax* function:

```
Module myfunctions
   Function FindMax(ByVal num1 As Integer, ByVal num2 As Integer) As Integer
      ' local variable declaration */
      Dim result As Integer
      If (num1 > num2) Then
          result = num1
      Else
```

```
         result = num2
      End If
      FindMax = result
   End Function
   Sub Main()
      Dim a As Integer = 100
      Dim b As Integer = 200
      Dim res As Integer
      res = FindMax(a, b)
      Console.WriteLine("Max value is : {0}", res)
      Console.ReadLine()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Max value is : 200
```

Recursive Function

A function can call itself. This is known as recursion. Following is an example that calculates factorial for a given number using a recursive function:

```
Module myfunctions
   Function factorial(ByVal num As Integer) As Integer
      ' local variable declaration */
      Dim result As Integer
      If (num = 1) Then
         Return 1
      Else
         result = factorial(num - 1) * num
         Return result
      End If
   End Function
   Sub Main()
      'calling the factorial method
      Console.WriteLine("Factorial of 6 is : {0}", factorial(6))
      Console.WriteLine("Factorial of 7 is : {0}", factorial(7))
      Console.WriteLine("Factorial of 8 is : {0}", factorial(8))
      Console.ReadLine()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Factorial of 6 is: 720
Factorial of 7 is: 5040
Factorial of 8 is: 40320
```

Param Arrays

At times, while declaring a function or sub procedure, you are not sure of the number of arguments passed as a parameter. VB.Net param arrays (or parameter arrays) come into help at these times. The following example demonstrates this:

```
Module myparamfunc
   Function AddElements(ParamArray arr As Integer()) As Integer
      Dim sum As Integer = 0
      Dim i As Integer = 0
      For Each i In arr
          sum += i
      Next i
      Return sum
   End Function
   Sub Main()
      Dim sum As Integer
      sum = AddElements(512, 720, 250, 567, 889)
      Console.WriteLine("The sum is: {0}", sum)
      Console.ReadLine()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
The sum is: 2938
```

Passing Arrays as Function Arguments

You can pass an array as a function argument in VB.Net. The following example demonstrates this:

```
Module arrayParameter
   Function getAverage(ByVal arr As Integer(), ByVal size As Integer) As Double
      'local variables
       Dim i As Integer
```

```
        Dim avg As Double
        Dim sum As Integer = 0
        For i = 0 To size - 1
            sum += arr(i)
        Next i
        avg = sum / size
        Return avg
    End Function
    Sub Main()
        ' an int array with 5 elements '
        Dim balance As Integer() = {1000, 2, 3, 17, 50}
        Dim avg As Double
        'pass pointer to the array as an argument
        avg = getAverage(balance, 5)
        ' output the returned value '
        Console.WriteLine("Average value is: {0} ", avg)
        Console.ReadLine()
    End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Average value is: 214.4
```

VB.Net - Sub Procedures

As we mentioned in the previous chapter, Sub procedures are procedures that do not return any value. We have been using the Sub procedure Main in all our examples. We have been writing console applications so far in these tutorials. When these applications start, the control goes to the Main Sub procedure, and it in turn, runs any other statements constituting the body of the program.

Defining Sub Procedures

The **Sub** statement is used to declare the name, parameter and the body of a sub procedure. The syntax for the Sub statement is:

```
[Modifiers] Sub SubName [(ParameterList)]
    [Statements]
End Sub
```

Where,

- ***Modifiers***: specify the access level of the procedure; possible values are: Public, Private, Protected, Friend, Protected Friend and information regarding overloading, overriding, sharing, and shadowing.
- ***SubName***: indicates the name of the Sub
- ***ParameterList***: specifies the list of the parameters

Example

The following example demonstrates a Sub procedure *CalculatePay* that takes two parameters *hours* and *wages* and displays the total pay of an employee:

```
Module mysub
  Sub CalculatePay(ByRef hours As Double, ByRef wage As Decimal)
    'local variable declaration
    Dim pay As Double
    pay = hours * wage
    Console.WriteLine("Total Pay: {0:C}", pay)
  End Sub
  Sub Main()
    'calling the CalculatePay Sub Procedure
    CalculatePay(25, 10)
    CalculatePay(40, 20)
    CalculatePay(30, 27.5)
    Console.ReadLine()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Total Pay: $250.00
Total Pay: $800.00
Total Pay: $825.00
```

Passing Parameters by Value

This is the default mechanism for passing parameters to a method. In this mechanism, when a method is called, a new storage location is created for each value parameter. The values of the actual parameters are copied into them. So, the changes made to the parameter inside the method have no effect on the argument.

In VB.Net, you declare the reference parameters using the **ByVal** keyword. The following example demonstrates the concept:

```
Module paramByval
```

```
  Sub swap(ByVal x As Integer, ByVal y As Integer)
    Dim temp As Integer
    temp = x ' save the value of x
    x = y    ' put y into x
    y = temp 'put temp into y
  End Sub
  Sub Main()
    ' local variable definition
    Dim a As Integer = 100
    Dim b As Integer = 200
    Console.WriteLine("Before swap, value of a : {0}", a)
    Console.WriteLine("Before swap, value of b : {0}", b)
    ' calling a function to swap the values '
    swap(a, b)
    Console.WriteLine("After swap, value of a : {0}", a)
    Console.WriteLine("After swap, value of b : {0}", b)
    Console.ReadLine()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Before swap, value of a :100
Before swap, value of b :200
After swap, value of a :100
After swap, value of b :200
```

It shows that there is no change in the values though they had been changed inside the function.

Passing Parameters by Reference

A reference parameter is a reference to a memory location of a variable. When you pass parameters by reference, unlike value parameters, a new storage location is not created for these parameters. The reference parameters represent the same memory location as the actual parameters that are supplied to the method. In VB.Net, you declare the reference parameters using the **ByRef** keyword. The following example demonstrates this:

```
Module paramByref
  Sub swap(ByRef x As Integer, ByRef y As Integer)
    Dim temp As Integer
    temp = x ' save the value of x
```

```
      x = y    ' put y into x
      y = temp 'put temp into y
   End Sub
   Sub Main()
      ' local variable definition
      Dim a As Integer = 100
      Dim b As Integer = 200
      Console.WriteLine("Before swap, value of a : {0}", a)
      Console.WriteLine("Before swap, value of b : {0}", b)
      ' calling a function to swap the values '
      swap(a, b)
      Console.WriteLine("After swap, value of a : {0}", a)
      Console.WriteLine("After swap, value of b : {0}", b)
      Console.ReadLine()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Before swap, value of a : 100
Before swap, value of b : 200
After swap, value of a : 200
After swap, value of b : 100
```

VB.Net - Classes & Objects

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

Objects are instances of a class. The methods and variables that constitute a class are called members of the class.

Class Definition

A class definition starts with the keyword **Class** followed by the class name; and the class body, ended by the End Class statement. Following is the general form of a class definition:

```
[ <attributelist> ] [ accessmodifier ] [ Shadows ] [ MustInherit | NotInheritable ] [ Partial ] _
Class name [ ( Of typelist ) ]
    [ Inherits classname ]
    [ Implements interfacenames ]
```

```
   [ statements ]
End Class
```

Where,

- ***attributelist*** is a list of attributes that apply to the class. Optional.
- ***accessmodifier*** defines the access levels of the class, it has values as - Public, Protected, Friend, Protected Friend and Private. Optional.
- ***Shadows*** indicate that the variable re-declares and hides an identically named element, or set of overloaded elements, in a base class. Optional.
- ***MustInherit*** specifies that the class can be used only as a base class and that you cannot create an object directly from it, i.e., an abstract class. Optional.
- ***NotInheritable*** specifies that the class cannot be used as a base class.
- ***Partial*** indicates a partial definition of the class.
- ***Inherits*** specifies the base class it is inheriting from.
- ***Implements*** specifies the interfaces the class is inheriting from.

The following example demonstrates a Box class, with three data members, length, breadth and height:

```
Module mybox
   Class Box
      Public length As Double   ' Length of a box
      Public breadth As Double   ' Breadth of a box
      Public height As Double    ' Height of a box
   End Class
   Sub Main()
      Dim Box1 As Box = New Box()       ' Declare Box1 of type Box
      Dim Box2 As Box = New Box()       ' Declare Box2 of type Box
      Dim volume As Double = 0.0     ' Store the volume of a box here
      ' box 1 specification
      Box1.height = 5.0
      Box1.length = 6.0
      Box1.breadth = 7.0
       ' box 2 specification
      Box2.height = 10.0
      Box2.length = 12.0
      Box2.breadth = 13.0
      'volume of box 1
      volume = Box1.height * Box1.length * Box1.breadth
      Console.WriteLine("Volume of Box1 : {0}", volume)
      'volume of box 2
      volume = Box2.height * Box2.length * Box2.breadth
      Console.WriteLine("Volume of Box2 : {0}", volume)
      Console.ReadKey()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Volume of Box1 : 210
Volume of Box2 : 1560
```

Member Functions and Encapsulation

A member function of a class is a function that has its definition or its prototype within the class definition like any other variable. It operates on any object of the class of which it is a member and has access to all the members of a class for that object.

Member variables are attributes of an object (from design perspective) and they are kept private to implement encapsulation. These variables can only be accessed using the public member functions.

Let us put above concepts to set and get the value of different class members in a class:

```
Module mybox
   Class Box
      Public length As Double   ' Length of a box
      Public breadth As Double   ' Breadth of a box
      Public height As Double    ' Height of a box
      Public Sub setLength(ByVal len As Double)
          length = len
      End Sub
      Public Sub setBreadth(ByVal bre As Double)
          breadth = bre
      End Sub
      Public Sub setHeight(ByVal hei As Double)
          height = hei
      End Sub
      Public Function getVolume() As Double
          Return length * breadth * height
      End Function
```

```
  End Class
  Sub Main()
    Dim Box1 As Box = New Box()       ' Declare Box1 of type Box
    Dim Box2 As Box = New Box()       ' Declare Box2 of type Box
    Dim volume As Double = 0.0     ' Store the volume of a box here
   ' box 1 specification
    Box1.setLength(6.0)
    Box1.setBreadth(7.0)
    Box1.setHeight(5.0)
       'box 2 specification
    Box2.setLength(12.0)
    Box2.setBreadth(13.0)
    Box2.setHeight(10.0)
        ' volume of box 1
    volume = Box1.getVolume()
    Console.WriteLine("Volume of Box1 : {0}", volume)
    'volume of box 2
    volume = Box2.getVolume()
    Console.WriteLine("Volume of Box2 : {0}", volume)
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Volume of Box1 : 210
Volume of Box2 : 1560
```

Constructors and Destructors

A class **constructor** is a special member Sub of a class that is executed whenever we create new objects of that class. A constructor has the name **New** and it does not have any return type.

Following program explains the concept of constructor:

```
Class Line
  Private length As Double    ' Length of a line
  Public Sub New()   'constructor
    Console.WriteLine("Object is being created")
  End Sub
  Public Sub setLength(ByVal len As Double)
```

```
      length = len
   End Sub
        Public Function getLength() As Double
      Return length
   End Function
   Shared Sub Main()
      Dim line As Line = New Line()
      'set line length
      line.setLength(6.0)
      Console.WriteLine("Length of line : {0}", line.getLength())
      Console.ReadKey()
   End Sub
End Class
```

When the above code is compiled and executed, it produces the following result:

```
Object is being created
Length of line : 6
```

A default constructor does not have any parameter, but if you need, a constructor can have parameters. Such constructors are called **parameterized constructors**. This technique helps you to assign initial value to an object at the time of its creation as shown in the following example:

```
Class Line
   Private length As Double     ' Length of a line
   Public Sub New(ByVal len As Double)   'parameterised constructor
      Console.WriteLine("Object is being created, length = {0}", len)
      length = len
   End Sub
   Public Sub setLength(ByVal len As Double)
      length = len
   End Sub
        Public Function getLength() As Double
      Return length
   End Function
   Shared Sub Main()
      Dim line As Line = New Line(10.0)
      Console.WriteLine("Length of line set by constructor : {0}", line.getLength())
      'set line length
```

```
    line.setLength(6.0)
    Console.WriteLine("Length of line set by setLength : {0}", line.getLength())
    Console.ReadKey()
  End Sub
End Class
```

When the above code is compiled and executed, it produces the following result:

```
Object is being created, length = 10
Length of line set by constructor : 10
Length of line set by setLength : 6
```

A **destructor** is a special member Sub of a class that is executed whenever an object of its class goes out of scope. A **destructor** has the name **Finalize** and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories, etc.

Destructors cannot be inherited or overloaded.

Following example explains the concept of destructor:

```
Class Line
  Private length As Double    ' Length of a line
  Public Sub New()   'parameterised constructor
    Console.WriteLine("Object is being created")
  End Sub
  Protected Overrides Sub Finalize()  ' destructor
    Console.WriteLine("Object is being deleted")
  End Sub
  Public Sub setLength(ByVal len As Double)
    length = len
  End Sub
  Public Function getLength() As Double
    Return length
  End Function
  Shared Sub Main()
    Dim line As Line = New Line()
    'set line length
    line.setLength(6.0)
    Console.WriteLine("Length of line : {0}", line.getLength())
    Console.ReadKey()
```

```
   End Sub
End Class
```

When the above code is compiled and executed, it produces the following result:

```
Object is being created
Length of line : 6
Object is being deleted
```

Shared Members of a VB.Net Class

We can define class members as static using the Shared keyword. When we declare a member of a class as Shared, it means no matter how many objects of the class are created, there is only one copy of the member. The keyword **Shared** implies that only one instance of the member exists for a class. Shared variables are used for defining constants because their values can be retrieved by invoking the class without creating an instance of it. Shared variables can be initialized outside the member function or class definition. You can also initialize Shared variables inside the class definition. You can also declare a member function as Shared. Such functions can access only Shared variables. The Shared functions exist even before the object is created.

The following example demonstrates the use of shared members:

```
Class StaticVar
   Public Shared num As Integer
   Public Sub count()
      num = num + 1
   End Sub
   Public Shared Function getNum() As Integer
      Return num
   End Function
   Shared Sub Main()
      Dim s As StaticVar = New StaticVar()
      s.count()
      s.count()
      s.count()
      Console.WriteLine("Value of variable num: {0}", StaticVar.getNum())
      Console.ReadKey()
   End Sub
End Class
```

When the above code is compiled and executed, it produces the following result:

```
Value of variable num: 3
```

Inheritance

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time. When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class, and the new class is referred to as the **derived** class.

Base & Derived Classes:

A class can be derived from more than one class or interface, which means that it can inherit data and functions from multiple base classes or interfaces.

The syntax used in VB.Net for creating derived classes is as follows:

```
<access-specifier> Class <base_class>
...
End Class
Class <derived_class>: Inherits <base_class>
...
End Class
```

Consider a base class Shape and its derived class Rectangle:

```
' Base class
Class Shape
  Protected width As Integer
  Protected height As Integer
  Public Sub setWidth(ByVal w As Integer)
    width = w
  End Sub
  Public Sub setHeight(ByVal h As Integer)
    height = h
  End Sub
End Class
' Derived class
Class Rectangle : Inherits Shape
  Public Function getArea() As Integer
    Return (width * height)
  End Function
End Class
Class RectangleTester
  Shared Sub Main()
    Dim rect As Rectangle = New Rectangle()
    rect.setWidth(5)
    rect.setHeight(7)
    ' Print the area of the object.
    Console.WriteLine("Total area: {0}", rect.getArea())
    Console.ReadKey()
  End Sub
End Class
```

When the above code is compiled and executed, it produces the following result:

```
Total area: 35
```

Base Class Initialization

The derived class inherits the base class member variables and member methods. Therefore, the super class object should be created before the subclass is created. The super class or the base class is implicitly known as **MyBase** in VB.Net

The following program demonstrates this:

```
' Base class
Class Rectangle
  Protected width As Double
  Protected length As Double
  Public Sub New(ByVal l As Double, ByVal w As Double)
    length = l
    width = w
  End Sub
  Public Function GetArea() As Double
    Return (width * length)
  End Function
  Public Overridable Sub Display()
    Console.WriteLine("Length: {0}", length)
    Console.WriteLine("Width: {0}", width)
    Console.WriteLine("Area: {0}", GetArea())
  End Sub
```

```
   'end class Rectangle
End Class
'Derived class
Class Tabletop : Inherits Rectangle
   Private cost As Double
   Public Sub New(ByVal l As Double, ByVal w As Double)
      MyBase.New(l, w)
   End Sub
   Public Function GetCost() As Double
      Dim cost As Double
      cost = GetArea() * 70
      Return cost
   End Function
   Public Overrides Sub Display()
      MyBase.Display()
      Console.WriteLine("Cost: {0}", GetCost())
   End Sub
    'end class Tabletop
End Class
Class RectangleTester
   Shared Sub Main()
      Dim t As Tabletop = New Tabletop(4.5, 7.5)
      t.Display()
      Console.ReadKey()
   End Sub
End Class
```

When the above code is compiled and executed, it produces the following result:

```
Length: 4.5
Width: 7.5
Area: 33.75
Cost: 2362.5
```

VB.Net supports multiple inheritance.

VB.Net - Exception Handling

An exception is a problem that arises during the execution of a program. An exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. VB.Net exception handling is built upon four keywords: **Try**, **Catch**, **Finally** and **Throw**.

- **Try**: A Try block identifies a block of code for which particular exceptions will be activated. It's followed by one or more Catch blocks.
- **Catch**: A program catches an exception with an exception handler at the place in a program where you want to handle the problem. The Catch keyword indicates the catching of an exception.
- **Finally**: The Finally block is used to execute a given set of statements, whether an exception is thrown or not thrown. For example, if you open a file, it must be closed whether an exception is raised or not.
- **Throw**: A program throws an exception when a problem shows up. This is done using a Throw keyword.

Syntax

Assuming a block will raise an exception, a method catches an exception using a combination of the Try and Catch keywords. A Try/Catch block is placed around the code that might generate an exception. Code within a Try/Catch block is referred to as protected code, and the syntax for using Try/Catch looks like the following:

```
Try
   [ tryStatements ]
   [ Exit Try ]
[ Catch [ exception [ As type ] ] [ When expression ]
   [ catchStatements ]
   [ Exit Try ] ]
[ Catch ... ]
[ Finally
   [ finallyStatements ] ]
End Try
```

You can list down multiple catch statements to catch different type of exceptions in case your try block raises more than one exception in different situations.

Exception Classes in .Net Framework

In the .Net Framework, exceptions are represented by classes. The exception classes in .Net Framework are mainly directly or indirectly derived from the **System. Exception** class. Some of the exception classes derived from the System. Exception class are the **System.ApplicationException** and **System.SystemException** classes.

The **System.ApplicationException** class supports exceptions generated by application programs. So the exceptions defined by the programmers should derive from this class.

The **System.SystemException** class is the base class for all predefined system exception.

The following table provides some of the predefined exception classes derived from the Sytem.SystemException class:

Exception Class	**Description**
System.IO.IOException	Handles I/O errors.
System.IndexOutOfRangeException	Handles errors generated when a method refers to an array index out of range.
System.ArrayTypeMismatchException	Handles errors generated when type is mismatched with the array type.
System.NullReferenceException	Handles errors generated from deferencing a null object.
System.DivideByZeroException	Handles errors generated from dividing a dividend with zero.
System.InvalidCastException	Handles errors generated during typecasting.
System.OutOfMemoryException	Handles errors generated from insufficient free memory.
System.StackOverflowException	Handles errors generated from stack overflow.

Handling Exceptions

VB.Net provides a structured solution to the exception handling problems in the form of try and catch blocks. Using these blocks the core program statements are separated from the error-handling statements.

These error handling blocks are implemented using the **Try**, **Catch** and **Finally** keywords. Following is an example of throwing an exception when dividing by zero condition occurs:

```
Module exceptionProg
  Sub division(ByVal num1 As Integer, ByVal num2 As Integer)
    Dim result As Integer
    Try
       result = num1 \ num2
    Catch e As DivideByZeroException
       Console.WriteLine("Exception caught: {0}", e)
    Finally
       Console.WriteLine("Result: {0}", result)
    End Try
  End Sub
  Sub Main()
    division(25, 0)
    Console.ReadKey()
 End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Exception caught: System.DivideByZeroException: Attempted to divide by zero.
at ...
Result: 0
```

Creating User-Defined Exceptions

You can also define your own exception. User-defined exception classes are derived from the **ApplicationException** class. The following example demonstrates this:

```
Module exceptionProg
  Public Class TempIsZeroException : Inherits ApplicationException
    Public Sub New(ByVal message As String)
       MyBase.New(message)
    End Sub
  End Class
  Public Class Temperature
    Dim temperature As Integer = 0
    Sub showTemp()
       If (temperature = 0) Then
          Throw (New TempIsZeroException("Zero Temperature found"))
       Else
          Console.WriteLine("Temperature: {0}", temperature)
       End If
    End Sub
  End Class
  Sub Main()
```

```
      Dim temp As Temperature = New Temperature()
      Try
         temp.showTemp()
      Catch e As TempIsZeroException
         Console.WriteLine("TempIsZeroException: {0}", e.Message)
      End Try
      Console.ReadKey()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
TempIsZeroException: Zero Temperature found
```

Throwing Objects

You can throw an object if it is either directly or indirectly derived from the System.Exception class.

You can use a throw statement in the catch block to throw the present object as:

```
Throw [ expression ]
```

The following program demonstrates this:

```
Module exceptionProg
   Sub Main()
      Try
         Throw New ApplicationException("A custom exception _
                     is being thrown here...")
      Catch e As Exception
         Console.WriteLine(e.Message)
      Finally
         Console.WriteLine("Now inside the Finally Block")
      End Try
      Console.ReadKey()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
A custom exception is being thrown here...
Now inside the Finally Block
```

VB.Net - File Handling

A **file** is a collection of data stored in a disk with a specific name and a directory path. When a file is opened for reading or writing, it becomes a **stream**. The stream is basically the sequence of bytes passing through the communication path. There are two main streams: the **input stream** and the **output stream**. The **input stream** is used for reading data from file (read operation) and the **output stream** is used for writing into the file (write operation).

VB.Net I/O Classes

The System.IO namespace has various classes that are used for performing various operations with files, like creating and deleting files, reading from or writing to a file, closing a file, etc.

The following table shows some commonly used non-abstract classes in the System.IO namespace:

I/O Class	**Description**
BinaryReader	Reads primitive data from a binary stream.
BinaryWriter	Writes primitive data in binary format.
BufferedStream	A temporary storage for a stream of bytes.
Directory	Helps in manipulating a directory structure.
DirectoryInfo	Used for performing operations on directories.
DriveInfo	Provides information for the drives.
File	Helps in manipulating files.
FileInfo	Used for performing operations on files.
FileStream	Used to read from and write to any location in a file.
MemoryStream	Used for random access of streamed data stored in memory.
Path	Performs operations on path information.
StreamReader	Used for reading characters from a byte stream.
StreamWriter	Is used for writing characters to a stream.
StringReader	Is used for reading from a string buffer.
StringWriter	Is used for writing into a string buffer.

The FileStream Class

The **FileStream** class in the System.IO namespace helps in reading from, writing to and closing files. This class derives from the abstract class Stream.

You need to create a **FileStream** object to create a new file or open an existing file. The syntax for creating a **FileStream** object is as follows:

```
Dim <object_name> As FileStream = New FileStream(<file_name>, <FileMode Enumerator>,
<FileAccess Enumerator>, <FileShare Enumerator>)
```

For example, for creating a FileStream object **F** for reading a file named **sample.txt**:

```
Dim f1 As FileStream = New FileStream("sample.txt", FileMode.OpenOrCreate,
FileAccess.ReadWrite)
```

Parameter	Description
FileMode	The **FileMode** enumerator defines various methods for opening files. The members of the FileMode enumerator are: **Append**: It opens an existing file and puts cursor at the end of file, or creates the file, if the file does not exist. **Create**: It creates a new file. **CreateNew**: It specifies to the operating system that it should create a new file. **Open**: It opens an existing file. **OpenOrCreate**: It specifies to the operating system that it should open a file if it exists, otherwise it should create a new file. **Truncate**: It opens an existing file and truncates its size to zero bytes.
FileAccess	**FileAccess** enumerators have members: **Read**, **ReadWrite** and **Write**.
FileShare	**FileShare** enumerators have the following members: **Inheritable**: It allows a file handle to pass inheritance to the child processes **None**: It declines sharing of the current file **Read**: It allows opening the file for reading **ReadWrite**: It allows opening the file for reading and writing **Write**: It allows opening the file for writing

Example:

The following program demonstrates use of the **FileStream** class:

```
Imports System.IO
Module fileProg
  Sub Main()
    Dim f1 As FileStream = New FileStream("sample.txt", _
         FileMode.OpenOrCreate, FileAccess.ReadWrite)
    Dim i As Integer
    For i = 0 To 20
       f1.WriteByte(CByte(i))
    Next i
    f1.Position = 0
```

```
      For i = 0 To 20
         Console.Write("{0} ", f1.ReadByte())
      Next i
      f1.Close()
      Console.ReadKey()
   End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 -1
```

Advanced File Operations in VB.Net

The preceding example provides simple file operations in VB.Net. However, to utilize the immense powers of System.IO classes, you need to know the commonly used properties and methods of these classes. We will discuss these classes and the operations they perform in the following sections. Please click the links provided to get to the individual sections:

Topic and Description
Reading from and Writing into Text files It involves reading from and writing into text files. The **StreamReader** and **StreamWriter** classes help to accomplish it.
Reading from and Writing into Binary files It involves reading from and writing into binary files. The **BinaryReader** and **BinaryWriter** classes help to accomplish this.
Manipulating the Windows file system It gives a VB.Net programmer the ability to browse and locate Windows files and directories.

VB.Net - Basic Controls

An object is a type of user interface element you create on a Visual Basic form by using a toolbox control. In fact, in Visual Basic, the form itself is an object. Every Visual Basic control consists of three important elements:

- **Properties** which describe the object,
- **Methods** cause an object to do something and
- **Events** are what happens when an object does something.

Control Properties

All the Visual Basic Objects can be moved, resized or customized by setting their properties. A property is a value or characteristic held by a Visual Basic object, such as Caption or Fore Color.

Properties can be set at design time by using the Properties window or at run time by using statements in the program code.

```
Object. Property = Value
```

Where

- **Object** is the name of the object you're customizing.
- **Property** is the characteristic you want to change.
- **Value** is the new property setting.

For example,

```
Form1.Caption = "Hello"
```

You can set any of the form properties using Properties Window. Most of the properties can be set or read during application execution. You can refer to Microsoft documentation for a complete list of properties associated with different controls and restrictions applied to them.

Control Methods

A method is a procedure created as a member of a class and they cause an object to do something. Methods are used to access or manipulate the characteristics of an object or a variable. There are mainly two categories of methods you will use in your classes:

- If you are using a control such as one of those provided by the Toolbox, you can call any of its public methods. The requirements of such a method depend on the class being used.
- If none of the existing methods can perform your desired task, you can add a method to a class.

For example, the *MessageBox* control has a method named *Show, which is called in the code snippet below:*

```
Public Class Form1
   Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As System.EventArgs)
            Handles Button1.Click
      MessageBox.Show("Hello, World")
   End Sub
End Class
```

Control Events

An event is a signal that informs an application that something important has occurred. For example, when a user clicks a control on a form, the form can raise a **Click** event and call a procedure that handles the event. There are various types of events associated with a Form like click, double click, close, load, resize, etc.

Following is the default structure of a form **Load** event handler subroutine. You can see this code by double clicking the code which will give you a complete list of the all events associated with Form control:

```
Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
 'event handler code goes here
End Sub
```

Here, **Handles MyBase.Load** indicates that **Form1_Load()** subroutine handles **Load** event. Similar way, you can check stub code for click, double click. If you want to initialize some variables like properties, etc., then you will keep such code inside Form1_Load() subroutine. Here, important point to note is the name of the event handler, which is by default Form1_Load, but you can change this name based on your naming convention you use in your application programming.

Basic Controls

VB.Net provides a huge variety of controls that help you to create rich user interface. Functionalities of all these controls are defined in the respective control classes. The control classes are defined in the **System.Windows.Forms** namespace.

The following table lists some of the commonly used controls:

S.N.	Widget & Description
1	**Forms** The container for all the controls that make up the user interface.
2	**TextBox** It represents a Windows text box control.
3	**Label** It represents a standard Windows label.
4	**Button** It represents a Windows button control.
5	**ListBox** It represents a Windows control to display a list of items.
6	**ComboBox** It represents a Windows combo box control.
7	**RadioButton** It enables the user to select a single option from a group of choices when paired with other RadioButton controls.
8	**CheckBox** It represents a Windows CheckBox.
9	**PictureBox** It represents a Windows picture box control for displaying an

	image.
10	**ProgressBar** It represents a Windows progress bar control.
11	**ScrollBar** It Implements the basic functionality of a scroll bar control.
12	**DateTimePicker** It represents a Windows control that allows the user to select a date and a time and to display the date and time with a specified format.
13	**TreeView** It displays a hierarchical collection of labeled items, each represented by a TreeNode.
14	**ListView** It represents a Windows list view control, which displays a collection of items that can be displayed using one of four different views.

VB.Net - Dialog Boxes

There are many built-in dialog boxes to be used in Windows forms for various tasks like opening and saving files, printing a page, providing choices for colors, fonts, page setup, etc., to the user of an application. These built-in dialog boxes reduce the developer's time and workload.

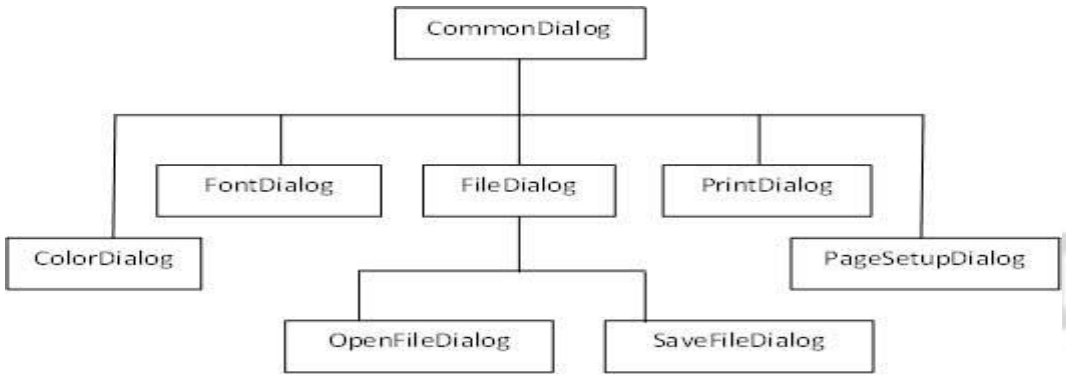
All of these dialog box control classes inherit from the **CommonDialog** class and override the *RunDialog()* function of the base class to create the specific dialog box.

The RunDialog() function is automatically invoked when a user of a dialog box calls its *ShowDialog()* function.

The **ShowDialog** method is used to display all the dialog box controls at run-time. It returns a value of the type of **DialogResult** enumeration. The values of DialogResult enumeration are:

- **Abort** - returns DialogResult.Abort value, when user clicks an Abort button.
- **Cancel**- returns DialogResult.Cancel, when user clicks a Cancel button.
- **Ignore** - returns DialogResult.Ignore, when user clicks an Ignore button.
- **No** - returns DialogResult.No, when user clicks a No button.
- **None** - returns nothing and the dialog box continues running.
- **OK** - returns DialogResult.OK, when user clicks an OK button
- **Retry** - returns DialogResult.Retry , when user clicks an Retry button
- **Yes** - returns DialogResult.Yes, when user clicks an Yes button

The following diagram shows the common dialog class inheritance:



All these above-mentioned classes have corresponding controls that could be added from the Toolbox during design time. You can include relevant functionality of these classes to your application, either by instantiating the class programmatically or by using relevant controls. When you double click any of the dialog controls in the toolbox or drag the control onto the form, it appears in the Component tray at the bottom of the Windows Forms Designer, they do not directly show up on the form. The following table lists the commonly used dialog box controls. Click the following links to check their detail:

S.N.	**Control & Description**
1	**ColorDialog** It represents a common dialog box that displays available colors along with controls that enable the user to define custom colors.
2	**FontDialog** It prompts the user to choose a font from among those installed on the local computer and lets the user select the font, font size, and color.
3	**OpenFileDialog** It prompts the user to open a file and allows the user to select a file to open.
4	**SaveFileDialog** It prompts the user to select a location for saving a file and allows the user to specify the name of the file to save data.
5	**PrintDialog** It lets the user to print documents by selecting a printer and choosing which sections of the document to print from a Windows Forms application.

VB.Net - Advanced Form

In this chapter, let us study the following concepts:

- Adding menus and sub menus in an application
- Adding the cut, copy and paste functionalities in a form
- Anchoring and docking controls in a form

- Modal forms

Adding Menus and Sub Menus in an Application

Traditionally, the *Menu*, *MainMenu*, *ContextMenu*, and *MenuItem* classes were used for adding menus, sub-menus and context menus in a Windows application.

Now,the **MenuStrip**,the **ToolStripMenuItem**, **ToolStripDropDown** and **ToolStripDropDownMenu** controls replace and add functionality to the Menu-related controls of previous versions. However, the old control classes are retained for both backward compatibility and future use.

Let us create a typical windows main menu bar and sub menus using the old version controls first since these controls are still much used in old applications.

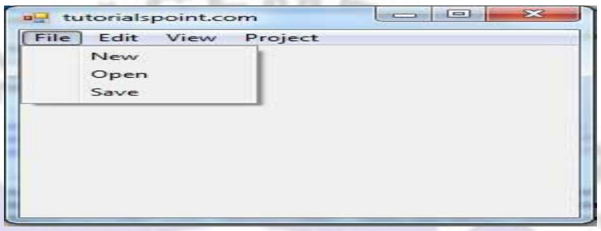
Following is an example, which shows how we create a menu bar with menu items: File, Edit, View and Project. The File menu has the sub menus New, Open and Save.

Let's double click on the Form and put the following code in the opened window.

```
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    'defining the main menu bar
    Dim mnuBar As New MainMenu()
    'defining the menu items for the main menu bar
    Dim myMenuItemFile As New MenuItem("&File")
    Dim myMenuItemEdit As New MenuItem("&Edit")
    Dim myMenuItemView As New MenuItem("&View")
    Dim myMenuItemProject As New MenuItem("&Project")
    'adding the menu items to the main menu bar
    mnuBar.MenuItems.Add(myMenuItemFile)
    mnuBar.MenuItems.Add(myMenuItemEdit)
    mnuBar.MenuItems.Add(myMenuItemView)
    mnuBar.MenuItems.Add(myMenuItemProject)
   ' defining some sub menus
    Dim myMenuItemNew As New MenuItem("&New")
    Dim myMenuItemOpen As New MenuItem("&Open")
    Dim myMenuItemSave As New MenuItem("&Save")
    'add sub menus to the File menu
    myMenuItemFile.MenuItems.Add(myMenuItemNew)
    myMenuItemFile.MenuItems.Add(myMenuItemOpen)
    myMenuItemFile.MenuItems.Add(myMenuItemSave)
    'add the main menu to the form
```

```
        Me.Menu = mnuBar
          ' Set the caption bar text of the form.
        Me.Text = "tutorialspoint.com"
    End Sub
End Class
```

When the above code is executed and run using Start button available at the Microsoft Visual Studio tool bar, it will show the following window:



Windows Forms contain a rich set of classes for creating your own custom menus with modern appearance, look and feel. The **MenuStrip**, **ToolStripMenuItem**, **ContextMenuStrip** controls are used to create menu bars and context menus efficiently.

Click the following links to check their details:

S.N.	Control & Description
1	**MenuStrip** It provides a menu system for a form.
2	**ToolStripMenuItem** It represents a selectable option displayed on a **MenuStrip** or **ContextMenuStrip**. The ToolStripMenuItem control replaces and adds functionality to the MenuItem control of previous versions.
2	**ContextMenuStrip** It represents a shortcut menu.

Adding the Cut, Copy and Paste Functionalities in a Form

The methods exposed by the **ClipBoard** class are used for adding the cut, copy and paste functionalities in an application. The ClipBoard class provides methods to place data on and retrieve data from the system Clipboard.

It has the following commonly used methods:

S.N	Method Name & Description
1	**Clear** Removes all data from the Clipboard.
2	**ContainsData** Indicates whether there is data on the Clipboard that is in the specified format or can be converted to that format.

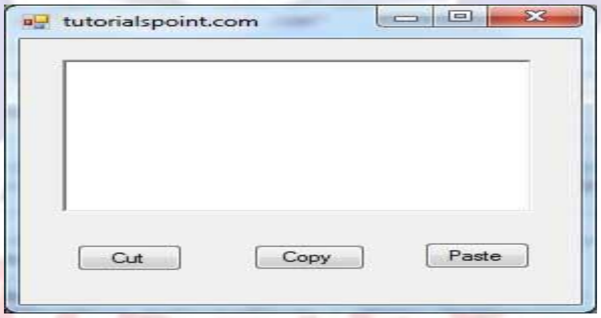
3	**ContainsImage** Indicates whether there is data on the Clipboard that is in the Bitmap format or can be converted to that format.
4	**ContainsText** Indicates whether there is data on the Clipboard in the Text or UnicodeText format, depending on the operating system.
5	**GetData** Retrieves data from the Clipboard in the specified format.
6	**GetDataObject** Retrieves the data that is currently on the system Clipboard.
7	**GetImage** Retrieves an image from the Clipboard.
8	**GetText** Retrieves text data from the Clipboard in the Text or UnicodeText format, depending on the operating system.
9	**GetText(TextDataFormat)** Retrieves text data from the Clipboard in the format indicated by the specified TextDataFormat value.
10	**SetData** Clears the Clipboard and then adds data in the specified format.
11	**SetText(String)** Clears the Clipboard and then adds text data in the Text or UnicodeText format, depending on the operating system.

Following is an example, which shows how we cut, copy and paste data using methods of the Clipboard class. Take the following steps:

- Add a rich text box control and three button controls on the form.
- Change the text property of the buttons to Cut, Copy and Paste, respectively.
- Double click on the buttons to add the following code in the code editor:

```
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) _
   Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspoint.com"
  End Sub
  Private Sub Button1_Click(sender As Object, e As EventArgs) _
    Handles Button1.Click
    Clipboard.SetDataObject(RichTextBox1.SelectedText)
    RichTextBox1.SelectedText = ""
  End Sub
  Private Sub Button2_Click(sender As Object, e As EventArgs) _
      Handles Button2.Click
    Clipboard.SetDataObject(RichTextBox1.SelectedText)
  End Sub
  Private Sub Button3_Click(sender As Object, e As EventArgs) _
   Handles Button3.Click
    Dim iData As IDataObject
    iData = Clipboard.GetDataObject()
    If (iData.GetDataPresent(DataFormats.Text)) Then
        RichTextBox1.SelectedText = iData.GetData(DataFormats.Text)
    Else
        RichTextBox1.SelectedText = " "
    End If
  End Sub
End Class
```

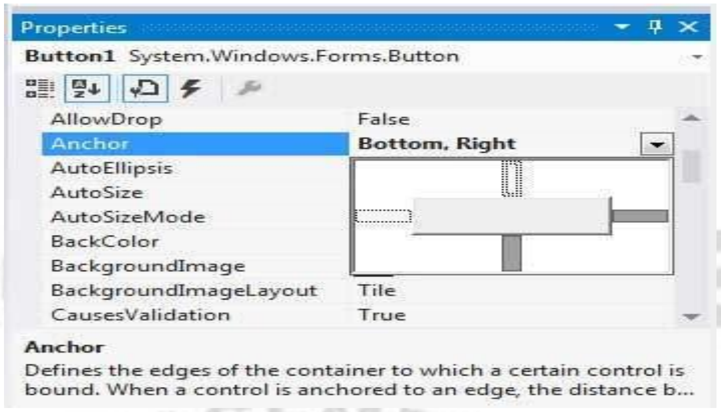
When the above code is executed and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window:



Enter some text and check how the buttons work.

Anchoring and Docking Controls in a Form

Anchoring allows you to set an anchor position for a control to the edges of its container control, for example, the form. The **Anchor** property of the Control class allows you to set values of this property. The Anchor property gets or sets the edges of the container to which a control is bound and determines how a control is resized with its parent. When you anchor a control to a form, the control maintains its distance from the edges of the form and its anchored position, when the form is resized. You can set the Anchor property values of a control from the Properties window:

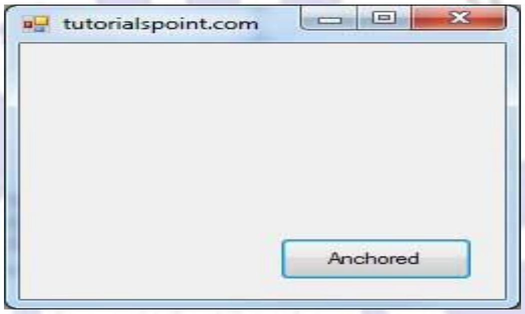

Properties

Button1 System.Windows.Forms.Button

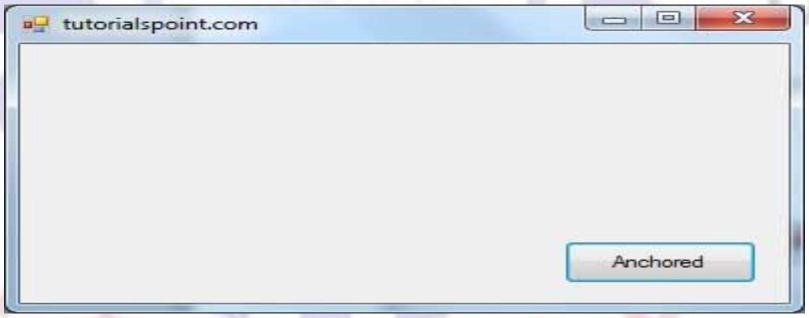
AllowDrop	False
Anchor	Bottom, Right
AutoEllipsis	
AutoSize	
AutoSizeMode	
BackColor	
BackgroundImage	
BackgroundImageLayout	Tile
CausesValidation	True

Anchor
Defines the edges of the container to which a certain control is bound. When a control is anchored to an edge, the distance b...

For example, let us add a Button control on a form and set its anchor property to Bottom, Right. Run this form to see the original position of the Button control with respect to the form.

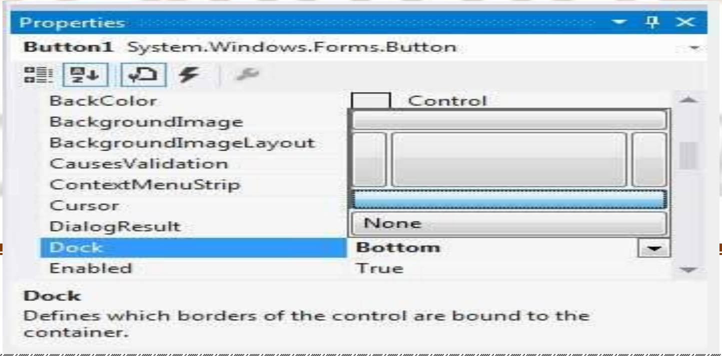


Now, when you stretch the form, the distance between the Button and the bottom right corner of the form remains same.



Docking of a control means docking it to one of the edges of its container. In docking, the control fills certain area of the container completely. The **Dock** property of the Control class does this. The Dock property gets or sets which control borders are docked to its parent control and determines how a control is resized with its parent.

You can set the Dock property values of a control from the Properties window:

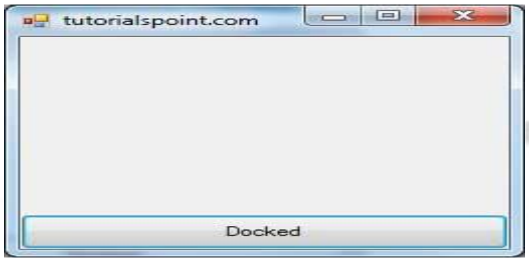

Properties

Button1 System.Windows.Forms.Button

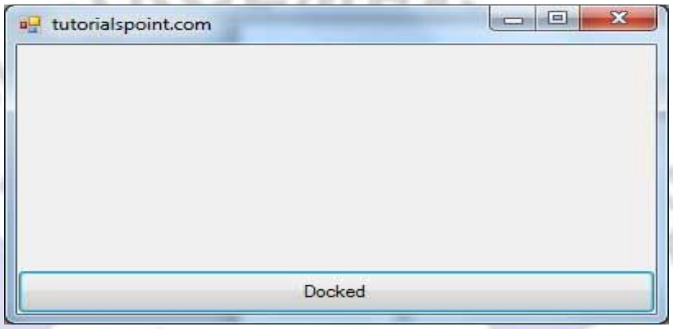
BackColor	Control
BackgroundImage	
BackgroundImageLayout	
CausesValidation	
ContextMenuStrip	
Cursor	
DialogResult	None
Dock	Bottom
Enabled	True

Dock
Defines which borders of the control are bound to the container.

For example, let us add a Button control on a form and set its Dock property to Bottom. Run this form to see the original position of the Button control with respect to the form.



Now, when you stretch the form, the Button resizes itself with the form.



Modal Forms

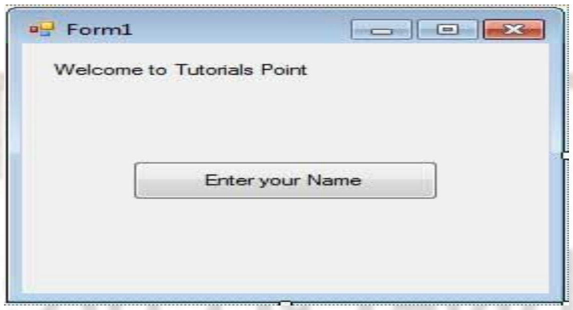
Modal Forms are those forms that need to be closed or hidden before you can continue working with the rest of the application. All dialog boxes are modal forms. A MessageBox is also a modal form.

You can call a modal form by two ways:

- Calling the **ShowDialog** method
- Calling the **Show** method

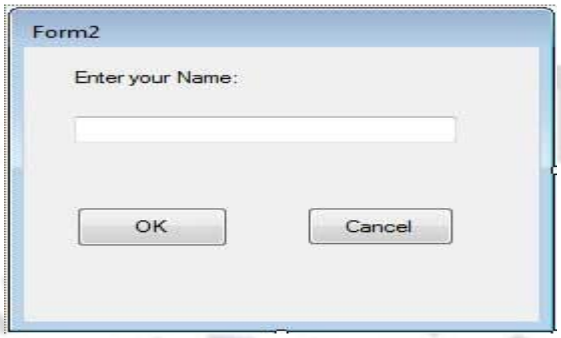
Let us take up an example in which we will create a modal form, a dialog box. Take the following steps:

- Add a form, Form1 to your application, and add two labels and a button control to Form1
- Change the text properties of the first label and the button to 'Welcome to Tutorials Point' and 'Enter your Name', respectively. Keep the text properties of the second label as blank.



- Add a new Windows Form, Form2, and add two buttons, one label, and a text box to Form2.
- Change the text properties of the buttons to OK and Cancel, respectively. Change the text properties of the label to 'Enter your name:'.
- Set the *FormBorderStyle* property of Form2 to *FixedDialog*, for giving it a dialog box border.

- Set the *ControlBox* property of Form2 to False.
- Set the *ShowInTaskbar* property of Form2 to False.
- Set the *DialogResult* property of the OK button to OK and the Cancel button to Cancel.



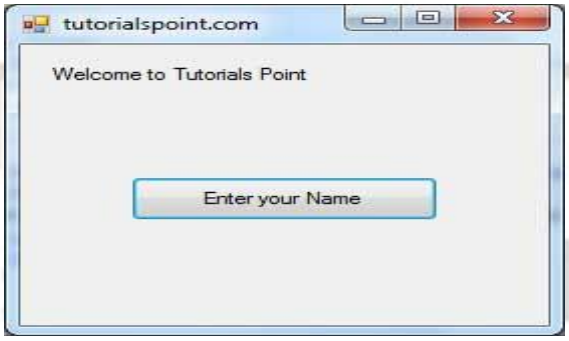
Add the following code snippets in the Form2_Load method of Form2:

```
Private Sub Form2_Load(sender As Object, e As EventArgs) _
  Handles MyBase.Load
  AcceptButton = Button1
  CancelButton = Button2
End Sub
```

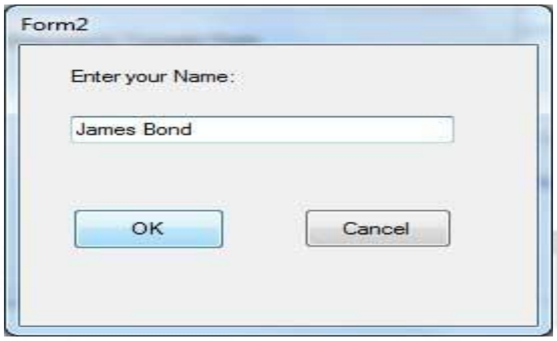
Add the following code snippets in the Button1_Click method of Form1:

```
Private Sub Button1_Click(sender As Object, e As EventArgs) _
  Handles Button1.Click
  Dim frmSecond As Form2 = New Form2()
  If frmSecond.ShowDialog() = DialogResult.OK Then
    Label2.Text = frmSecond.TextBox1.Text
  End If
End Sub
```

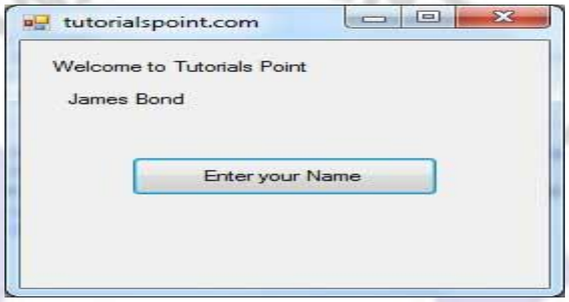
When the above code is executed and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window:



Clicking on the 'Enter your Name' button displays the second form:



Clicking on the OK button takes the control and information back from the modal form to the previous form:



VB.Net - Event Handling

Events are basically a user action like key press, clicks, mouse movements, etc., or some occurrence like system generated notifications. Applications need to respond to events when they occur. Clicking on a button, or entering some text in a text box, or clicking on a menu item, all are examples of events. An event is an action that calls a function or may cause another event.

Event handlers are functions that tell how to respond to an event.

VB.Net is an event-driven language. There are mainly two types of events:

- Mouse events
- Keyboard events

Handling Mouse Events

Mouse events occur with mouse movements in forms and controls. Following are the various mouse events related with a Control class:

- **MouseDown** - it occurs when a mouse button is pressed
- **MouseEnter** - it occurs when the mouse pointer enters the control
- **MouseHover** - it occurs when the mouse pointer hovers over the control
- **MouseLeave** - it occurs when the mouse pointer leaves the control
- **MouseMove** - it occurs when the mouse pointer moves over the control
- **MouseUp** - it occurs when the mouse pointer is over the control and the mouse button is released
- **MouseWheel** - it occurs when the mouse wheel moves and the control has focus

The event handlers of the mouse events get an argument of type **MouseEventArgs**. The MouseEventArgs object is used for handling mouse events. It has the following properties:

- **Buttons** - indicates the mouse button pressed
- **Clicks** - indicates the number of clicks
- **Delta** - indicates the number of detents the mouse wheel rotated
- **X** - indicates the x-coordinate of mouse click
- **Y** - indicates the y-coordinate of mouse click

Example

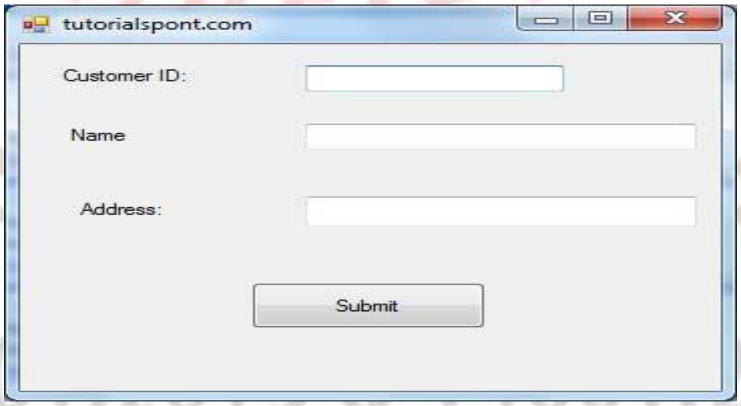
Following is an example, which shows how to handle mouse events. Take the following steps:

- Add three labels, three text boxes and a button control in the form.
- Change the text properties of the labels to - Customer ID, Name and Address, respectively.
- Change the name properties of the text boxes to txtID, txtName and txtAddress, respectively.
- Change the text property of the button to 'Submit'.
- Add the following code in the code editor window:

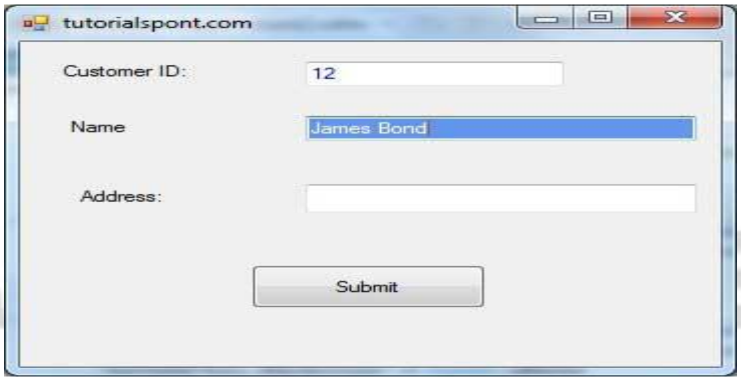
```
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspont.com"
  End Sub
  Private Sub txtID_MouseEnter(sender As Object, e As EventArgs)_
      Handles txtID.MouseEnter
    'code for handling mouse enter on ID textbox
    txtID.BackColor = Color.CornflowerBlue
    txtID.ForeColor = Color.White
  End Sub
  Private Sub txtID_MouseLeave(sender As Object, e As EventArgs) _
      Handles txtID.MouseLeave
    'code for handling mouse leave on ID textbox
    txtID.BackColor = Color.White
    txtID.ForeColor = Color.Blue
  End Sub
  Private Sub txtName_MouseEnter(sender As Object, e As EventArgs) _
     Handles txtName.MouseEnter
    'code for handling mouse enter on Name textbox
    txtName.BackColor = Color.CornflowerBlue
    txtName.ForeColor = Color.White
  End Sub
  Private Sub txtName_MouseLeave(sender As Object, e As EventArgs) _
```

```
     Handles txtName.MouseLeave
     'code for handling mouse leave on Name textbox
     txtName.BackColor = Color.White
     txtName.ForeColor = Color.Blue
   End Sub
   Private Sub txtAddress_MouseEnter(sender As Object, e As EventArgs) _
     Handles txtAddress.MouseEnter
     'code for handling mouse enter on Address textbox
     txtAddress.BackColor = Color.CornflowerBlue
     txtAddress.ForeColor = Color.White
   End Sub
   Private Sub txtAddress_MouseLeave(sender As Object, e As EventArgs) _
       Handles txtAddress.MouseLeave
     'code for handling mouse leave on Address textbox
     txtAddress.BackColor = Color.White
     txtAddress.ForeColor = Color.Blue
   End Sub
   Private Sub Button1_Click(sender As Object, e As EventArgs) _
      Handles Button1.Click
     MsgBox("Thank you " & txtName.Text & ", for your kind cooperation")
   End Sub
End Class
```

When the above code is executed and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window:



Try to enter text in the text boxes and check the mouse events:


tutorialspont.com

Customer ID: 12

Name James Bond

Address:

Submit

Handling Keyboard Events

Following are the various keyboard events related with a Control class:

- **KeyDown** - occurs when a key is pressed down and the control has focus
- **KeyPress** - occurs when a key is pressed and the control has focus
- **KeyUp** - occurs when a key is released while the control has focus

The event handlers of the KeyDown and KeyUp events get an argument of type **KeyEventArgs**. This object has the following properties:

- **Alt** - it indicates whether the ALT key is pressed/p>
- **Control** - it indicates whether the CTRL key is pressed
- **Handled** - it indicates whether the event is handled
- **KeyCode** - stores the keyboard code for the event
- **KeyData** - stores the keyboard data for the event
- **KeyValue** - stores the keyboard value for the event
- **Modifiers** - it indicates which modifier keys (Ctrl, Shift, and/or Alt) are pressed
- **Shift** - it indicates if the Shift key is pressed

The event handlers of the KeyDown and KeyUp events get an argument of type **KeyEventArgs**. This object has the following properties:

- **Handled** - indicates if the KeyPress event is handled
- **KeyChar** - stores the character corresponding to the key pressed

Example

Let us continue with the previous example to show how to handle keyboard events. The code will verify that the user enters some numbers for his customer ID and age.

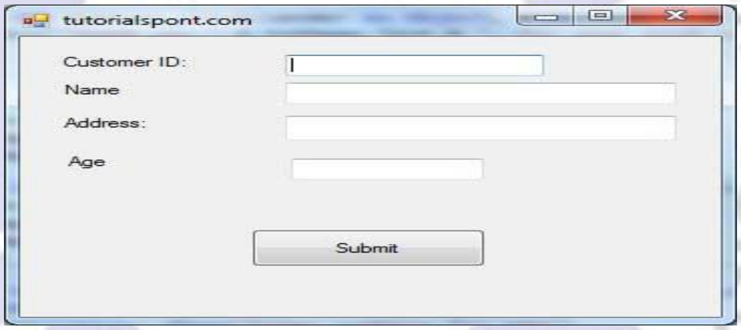
- Add a label with text Property as 'Age' and add a corresponding text box named txtAge.
- Add the following codes for handling the KeyUP events of the text box txtID.

```
• Private Sub txtID_KeyUP(sender As Object, e As KeyEventArgs) _
•    Handles txtID.KeyUp
•    If (Not Char.IsNumber(ChrW(e.KeyCode))) Then
•       MessageBox.Show("Enter numbers for your Customer ID")
•       txtID.Text = " "
•    End If
End Sub
```

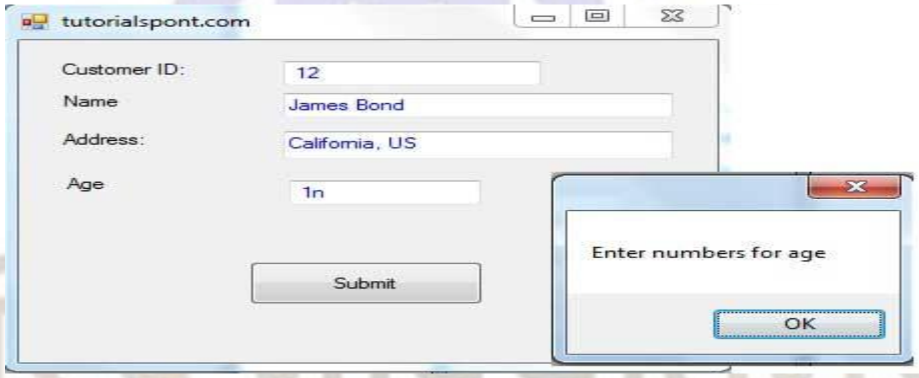
- Add the following codes for handling the KeyUP events of the text box txtID.

```
•  Private Sub txtAge_KeyUP(sender As Object, e As KeyEventArgs) _
•    Handles txtAge.KeyUp
•    If (Not Char.IsNumber(ChrW(e.keyCode))) Then
•      MessageBox.Show("Enter numbers for age")
•      txtAge.Text = " "
•    End If
End Sub
```

When the above code is executed and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window:



If you leave the text for age or ID as blank or enter some non-numeric data, it gives a warning message box and clears the respective text:



Customer ID: 12
Name: James Bond
Address: California, US
Age: 1n

Enter numbers for age

VB.Net - Regular Expressions

A **regular expression** is a pattern that could be matched against an input text. The .Net framework provides a regular expression engine that allows such matching. A pattern consists of one or more character literals, operators, or constructs.

Constructs for Defining Regular Expressions

There are various categories of characters, operators, and constructs that lets you to define regular expressions. Click the follwoing links to find these constructs.

- Character escapes
- Character classes
- Anchors
- Grouping constructs
- Quantifiers
- Backreference constructs
- Alternation constructs
- Substitutions
- Miscellaneous constructs

The Regex Class

The Regex class is used for representing a regular expression.

The Regex class has the following commonly used methods:

S.N	Methods & Description
1	**Public Function IsMatch (input As String) As Boolean** Indicates whether the regular expression specified in the Regex constructor finds a match in a specified input string.
2	**Public Function IsMatch (input As String, startat As Integer) As Boolean** Indicates whether the regular expression specified in the Regex constructor finds a match in the specified input string, beginning at the specified starting position in the string.
3	**Public Shared Function IsMatch (input As String, pattern As String) As Boolean** Indicates whether the specified regular expression finds a match in the specified input string.
4	**Public Function Matches (input As String) As MatchCollection** Searches the specified input string for all occurrences of a regular expression.
5	**Public Function Replace (input As String, replacement As String) As String** In a specified input string, replaces all strings that match a regular expression pattern with a specified replacement string.
6	**Public Function Split (input As String) As String()** Splits an input string into an array of substrings at the positions defined by a regular expression pattern specified in the Regex constructor.

For the complete list of methods and properties, please consult Microsoft documentation.

Example 1

The following example matches words that start with 'S':

```
Imports System.Text.RegularExpressions
Module regexProg
  Sub showMatch(ByVal text As String, ByVal expr As String)
    Console.WriteLine("The Expression: " + expr)
    Dim mc As MatchCollection = Regex.Matches(text, expr)
    Dim m As Match
    For Each m In mc
       Console.WriteLine(m)
    Next m
  End Sub
  Sub Main()
    Dim str As String = "A Thousand Splendid Suns"
    Console.WriteLine("Matching words that start with 'S': ")
    showMatch(str, "\bS\S*")
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Matching words that start with 'S':
The Expression: \bS\S*
Splendid
Suns
```

Example 2

The following example matches words that start with 'm' and ends with 'e':

```
Imports System.Text.RegularExpressions
Module regexProg
  Sub showMatch(ByVal text As String, ByVal expr As String)
    Console.WriteLine("The Expression: " + expr)
    Dim mc As MatchCollection = Regex.Matches(text, expr)
    Dim m As Match
    For Each m In mc
       Console.WriteLine(m)
    Next m
  End Sub
  Sub Main()
    Dim str As String = "make a maze and manage to measure it"
    Console.WriteLine("Matching words that start with 'm' and ends with 'e': ")
    showMatch(str, "\bm\S*e\b")
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

```
Matching words start with 'm' and ends with 'e':
The Expression: \bm\S*e\b
make
maze
manage
measure
```

Example 3

This example replaces extra white space:

```
Imports System.Text.RegularExpressions
Module regexProg
  Sub Main()
    Dim input As String = "Hello    World   "
    Dim pattern As String = "\\s+"
    Dim replacement As String = " "
    Dim rgx As Regex = New Regex(pattern)
    Dim result As String = rgx.Replace(input, replacement)
    Console.WriteLine("Original String: {0}", input)
    Console.WriteLine("Replacement String: {0}", result)
    Console.ReadKey()
  End Sub
End Module
```

When the above code is compiled and executed, it produces the following result:

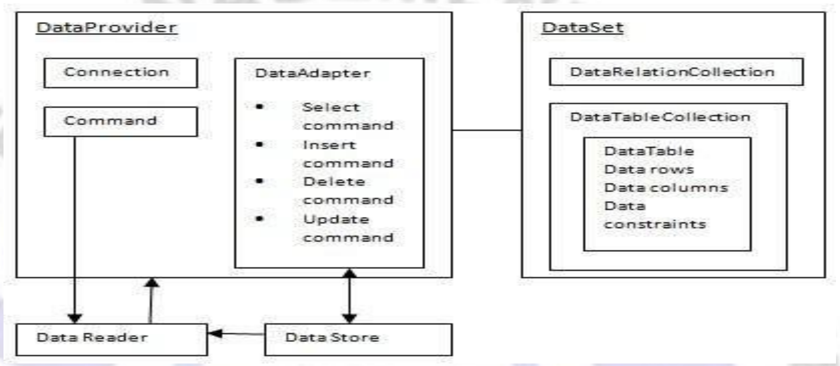
```
Original String: Hello   World
Replacement String: Hello World
```

VB.Net - Database Access

Applications communicate with a database, firstly, to retrieve the data stored there and present it in a user-friendly way, and secondly, to update the database by inserting, modifying and deleting data. Microsoft ActiveX Data Objects.Net (ADO.Net) is a model, a part of the .Net framework that is used by the .Net applications for retrieving, accessing and updating data.

ADO.Net Object Model

ADO.Net object model is nothing but the structured process flow through various components. The object model can be pictorially described as:



The data residing in a data store or database is retrieved through the **data provider**. Various components of the data provider retrieve data for the application and update data.

An application accesses data either through a dataset or a data reader.

- **Datasets** store data in a disconnected cache and the application retrieves data from it.
- **Data readers** provide data to the application in a read-only and forward-only mode.

Data Provider

A data provider is used for connecting to a database, executing commands and retrieving data, storing it in a dataset, reading the retrieved data and updating the database.

The data provider in ADO.Net consists of the following four objects:

S.N	Objects & Description
1	**Connection** This component is used to set up a connection with a data source.
2	**Command** A command is a SQL statement or a stored procedure used to retrieve, insert, delete or modify data in a data source.
3	**DataReader** Data reader is used to retrieve data from a data source in a read-only and forward-only mode.
4	**DataAdapter** This is integral to the working of ADO.Net since data is transferred to and from a database through a data adapter. It retrieves data from a database into a dataset and updates the database. When changes

	are made to the dataset, the changes in the database are actually done by the data adapter.

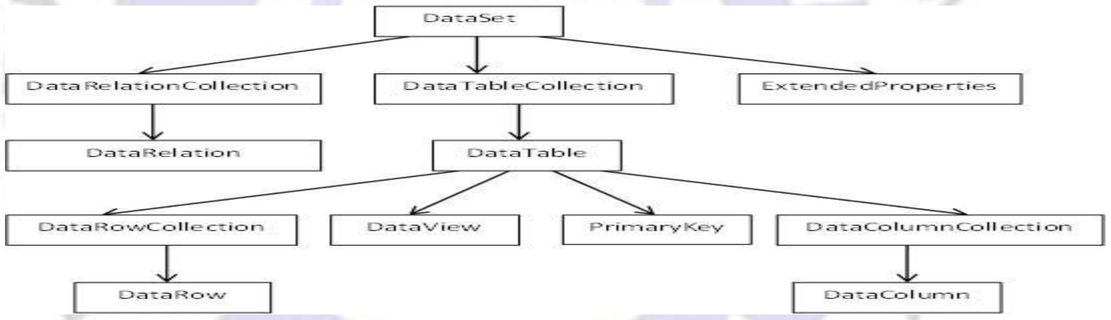
There are following different types of data providers included in ADO.Net

- The .Net Framework data provider for SQL Server - provides access to Microsoft SQL Server.
- The .Net Framework data provider for OLE DB - provides access to data sources exposed by using OLE DB.
- The .Net Framework data provider for ODBC - provides access to data sources exposed by ODBC.
- The .Net Framework data provider for Oracle - provides access to Oracle data source.
- The EntityClient provider - enables accessing data through Entity Data Model (EDM) applications.

DataSet

DataSet is an in-memory representation of data. It is a disconnected, cached set of records that are retrieved from a database. When a connection is established with the database, the data adapter creates a dataset and stores data in it. After the data is retrieved and stored in a dataset, the connection with the database is closed. This is called the 'disconnected architecture'. The dataset works as a virtual database containing tables, rows, and columns.

The following diagram shows the dataset object model:



The DataSet class is present in the **System.Data** namespace. The following table describes all the components of DataSet:

S.N	Components & Description
1	**DataTableCollection** It contains all the tables retrieved from the data source.
2	**DataRelationCollection** It contains relationships and the links between tables in a data set.
3	**ExtendedProperties** It contains additional information, like the SQL statement for retrieving data, time of retrieval, etc.
4	**DataTable** It represents a table in the DataTableCollection of a dataset. It consists of the DataRow and DataColumn objects. The DataTable objects are case-sensitive.
5	**DataRelation** It represents a relationship in the DataRelationshipCollection of the dataset. It is used to relate two DataTable objects to each other

	through the DataColumn objects.
6	**DataRowCollection** It contains all the rows in a DataTable.
7	**DataView** It represents a fixed customized view of a DataTable for sorting, filtering, searching, editing and navigation.
8	**PrimaryKey** It represents the column that uniquely identifies a row in a DataTable.
9	**DataRow** It represents a row in the DataTable. The DataRow object and its properties and methods are used to retrieve, evaluate, insert, delete, and update values in the DataTable. The NewRow method is used to create a new row and the Add method adds a row to the table.
10	**DataColumnCollection** It represents all the columns in a DataTable.
11	**DataColumn** It consists of the number of columns that comprise a DataTable.

Connecting to a Database

The .Net Framework provides two types of Connection classes:

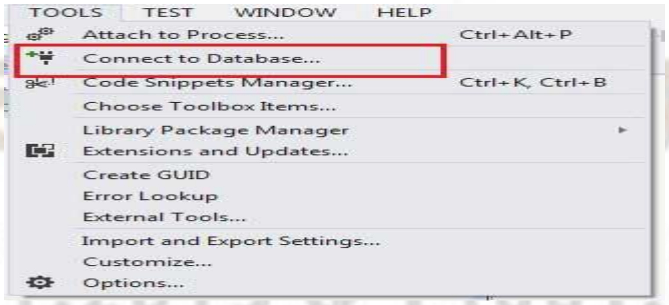
- **SqlConnection** - designed for connecting to Microsoft SQL Server.
- **OleDbConnection** - designed for connecting to a wide range of databases, like Microsoft Access and Oracle.

Example 1

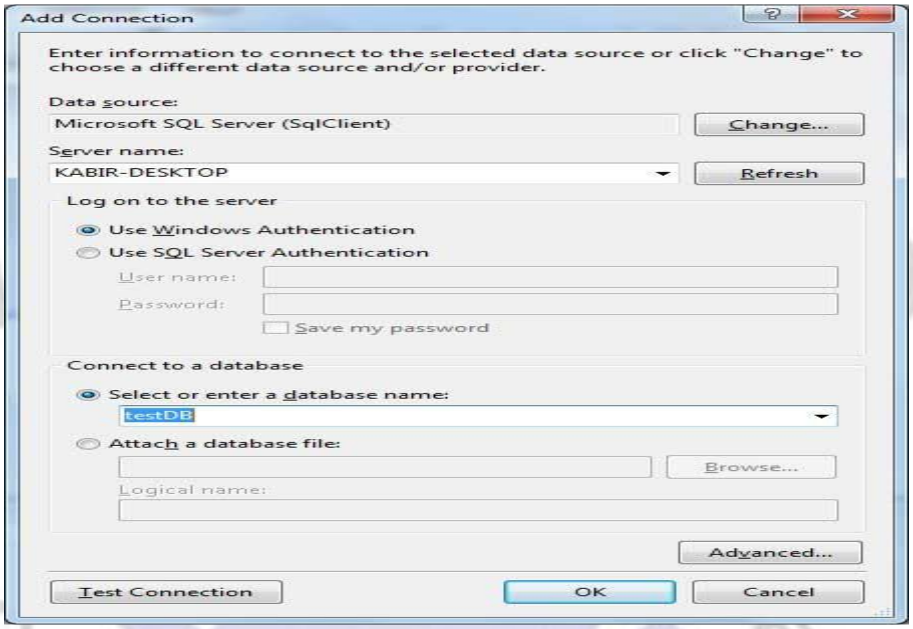
We have a table stored in Microsoft SQL Server, named Customers, in a database named testDB. Please consult 'SQL Server' tutorial for creating databases and database tables in SQL Server.

Let us connect to this database. Take the following steps:

- Select TOOLS -> Connect to Database



- Select a server name and the database name in the Add Connection dialog box.


Add Connection

Enter information to connect to the selected data source or click "Change" to choose a different data source and/or provider.

Data source: Microsoft SQL Server (SqlClient)

Server name: KABIR-DESKTOP

Log on to the server

- [✓ Use Windows Authentication]
- Use SQL Server Authentication

User name:

Password:

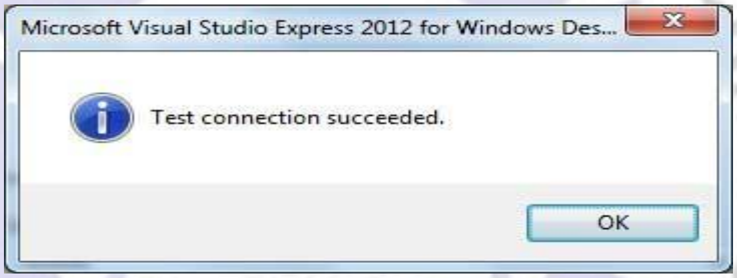
Save my password

Connect to a database

- [✓ Select or enter a database name:] testDB
- Attach a database file:

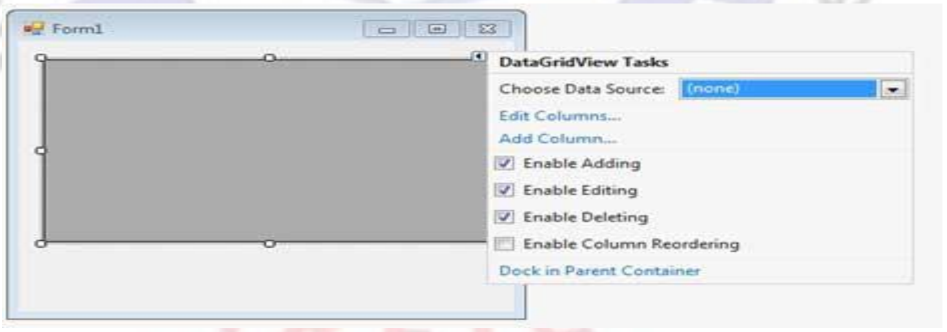
Logical name:

- Click on the Test Connection button to check if the connection succeeded.

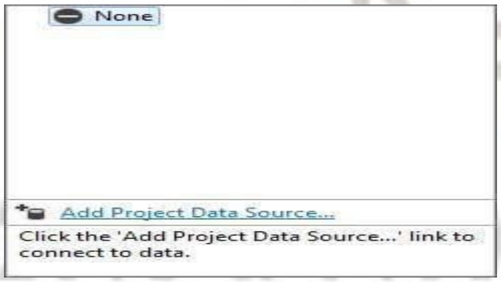

Microsoft Visual Studio Express 2012 for Windows Des...

Test connection succeeded.

- Add a DataGridView on the form.

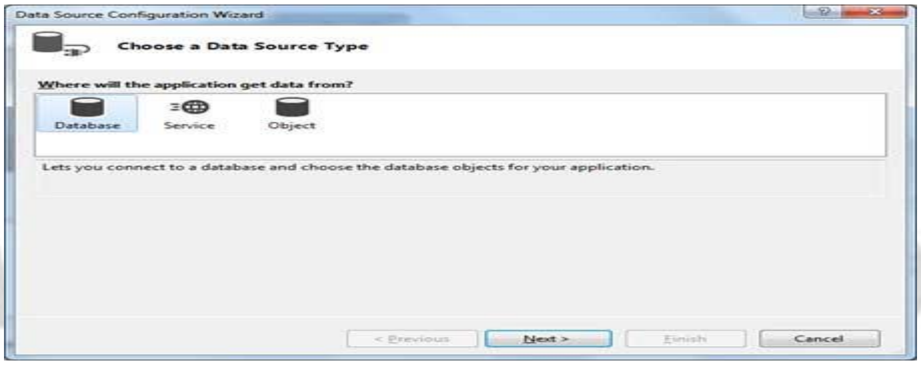


- Click on the Choose Data Source combo box.
- Click on the Add Project Data Source link.



Click the 'Add Project Data Source...' link to connect to data.

- This opens the Data Source Configuration Wizard.
- Select Database as the data source type

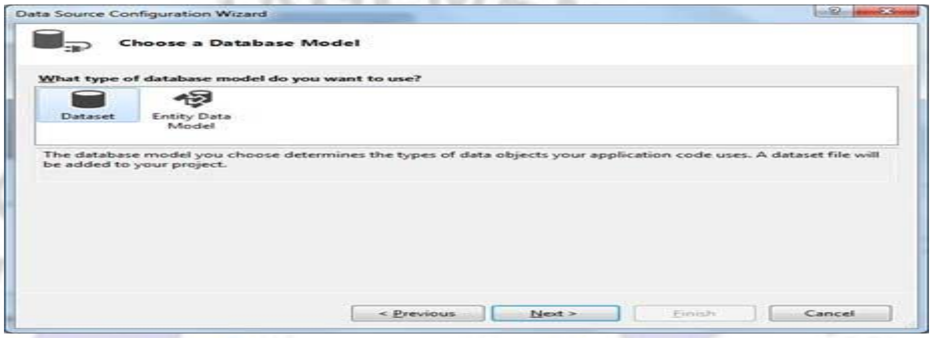

Data Source Configuration Wizard

Choose a Data Source Type

Where will the application get data from?

Lets you connect to a database and choose the database objects for your application.

- Choose DataSet as the database model.

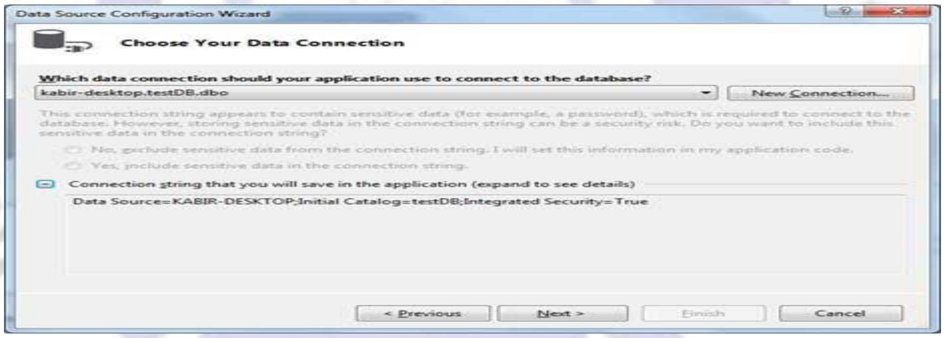

Data Source Configuration Wizard

Choose a Database Model

What type of database model do you want to use?

The database model you choose determines the types of data objects your application code uses. A dataset file will be added to your project.

- Choose the connection already set up.


Data Source Configuration Wizard

Choose Your Data Connection

Which data connection should your application use to connect to the database?

kabir-desktop.testDB.dbo

This connection string appears to contain sensitive data (for example, a password), which is required to connect to the database. However, storing sensitive data in the connection string can be a security risk. Do you want to include this sensitive data in the connection string?

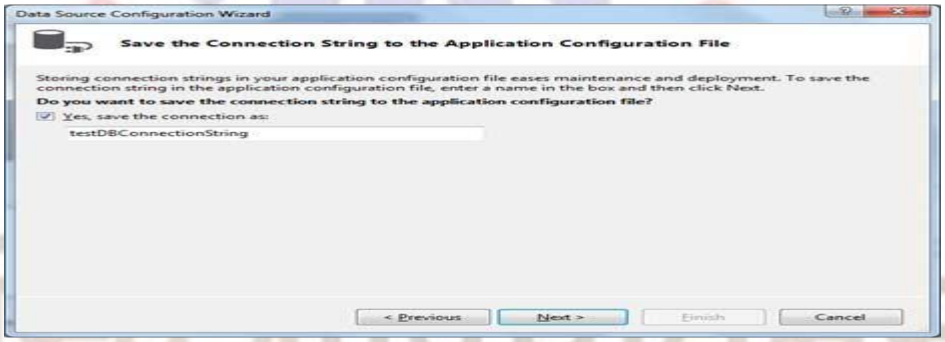
No, exclude sensitive data from the connection string. I will set this information in my application code.

Yes, include sensitive data in the connection string.

Connection string that you will save in the application (expand to see details)

Data Source=KABIR-DESKTOP;Initial Catalog=testDB;Integrated Security=True

- Save the connection string.


Data Source Configuration Wizard

Save the Connection String to the Application Configuration File

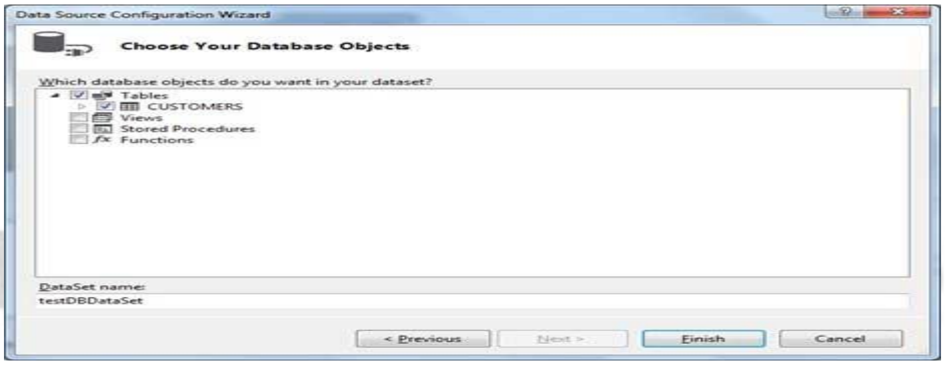
Storing connection strings in your application configuration file eases maintenance and deployment. To save the connection string in the application configuration file, enter a name in the box and then click Next.

Do you want to save the connection string to the application configuration file?

☑ Yes, save the connection as:

testDBConnectionString

- Choose the database object, Customers table in our example, and click the Finish button.


Data Source Configuration Wizard

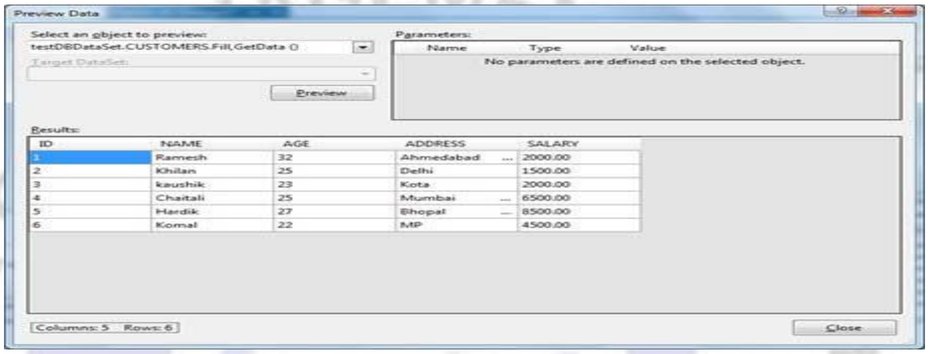
Choose Your Database Objects

Which database objects do you want in your dataset?

- ☑ Tables
  - ☑ CUSTOMERS
- ☐ Views
- ☐ Stored Procedures
- ☐ Functions

DataSet name:
testDBDataSet

- Select the Preview Data link to see the data in the Results grid:


Preview Data

Select an object to preview:
testDBDataSet.CUSTOMERS.Fill,GetData ()

Target DataSet:

Parameters:

Name	Type	Value

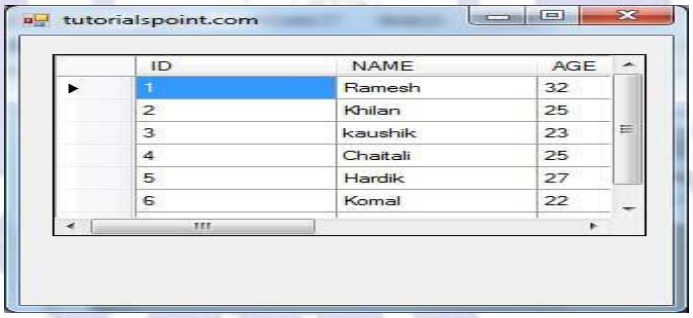
No parameters are defined on the selected object.

Results:

ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad	2000.00
2	Khilan	25	Delhi	1500.00
3	kaushik	23	Kota	2000.00
4	Chaitali	25	Mumbai	6500.00
5	Hardik	27	Bhopal	8500.00
6	Komal	22	MP	4500.00

Columns: 5 Rows: 6

When the application is run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window:


tutorialspoint.com

ID	NAME	AGE
1	Ramesh	32
2	Khilan	25
3	kaushik	23
4	Chaitali	25
5	Hardik	27
6	Komal	22

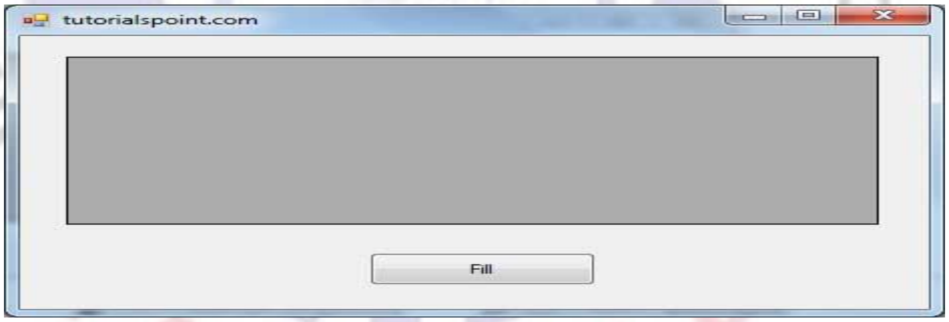
Example 2

In this example, let us access data in a DataGridView control using code. Take the following steps:

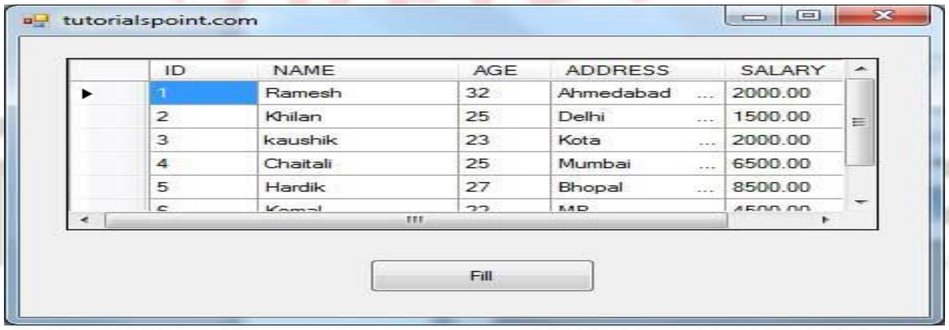
- Add a DataGridView control and a button in the form.
- Change the text of the button control to 'Fill'.
- Double click the button control to add the required code for the Click event of the button, as shown below:

```
Imports System.Data.SqlClient
Public Class Form1
   Private Sub Form1_Load(sender As Object, e As EventArgs) _
   Handles MyBase.Load
        'TODO: This line of code loads data into the 'TestDBDataSet.CUSTOMERS' table.   You can move, or remove it, as needed.
      Me.CUSTOMERSTableAdapter.Fill(Me.TestDBDataSet.CUSTOMERS)
      ' Set the caption bar text of the form.
      Me.Text = "tutorialspoint.com"
   End Sub
   Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
      Dim connection As SqlConnection = New sqlconnection()
      connection.ConnectionString = "Data Source=KABIR-DESKTOP; _
         Initial Catalog=testDB;Integrated Security=True"
      connection.Open()
      Dim adp As SqlDataAdapter = New SqlDataAdapter _
      ("select * from Customers", connection)
      Dim ds As DataSet = New DataSet()
      adp.Fill(ds)
      DataGridView1.DataSource = ds.Tables(0)
   End Sub
End Class
```

When the above code is executed and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window:



Clicking the Fill button displays the table on the data grid view control:



ID	NAME	AGE	ADDRESS	SALARY
1	Ramesh	32	Ahmedabad ...	2000.00
2	Khilan	25	Delhi ...	1500.00
3	kaushik	23	Kota ...	2000.00
4	Chaitali	25	Mumbai ...	6500.00
5	Hardik	27	Bhopal ...	8500.00
6	Komal	22	MP	4500.00

Creating Table, Columns and Rows

We have discussed that the DataSet components like DataTable, DataColumn and DataRow allow us to create tables, columns and rows, respectively.

The following example demonstrates the concept:

Example 3

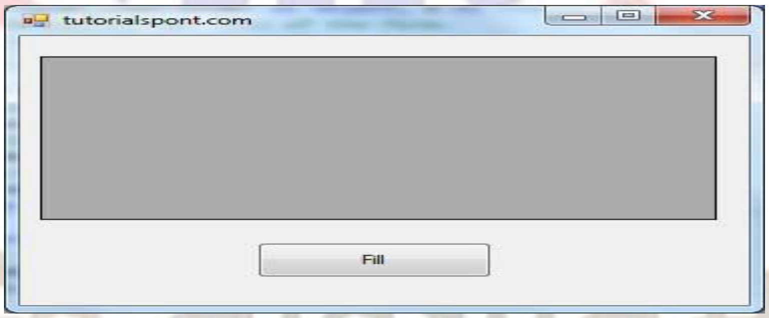
So far, we have used tables and databases already existing in our computer. In this example, we will create a table, add columns, rows and data into it and display the table using a DataGridView object.

Take the following steps:

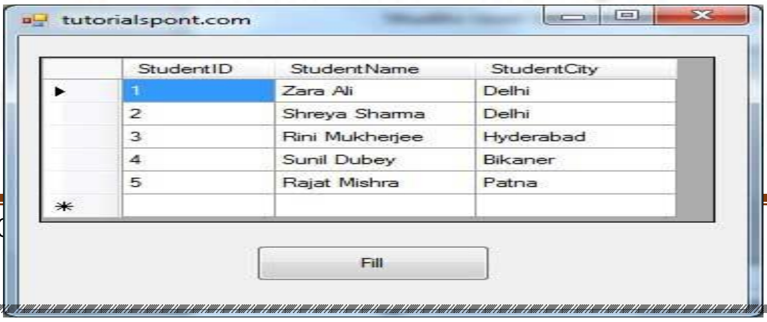
- Add a DataGridView control and a button in the form.
- Change the text of the button control to 'Fill'.
- Add the following code in the code editor.

```
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspont.com"
  End Sub
  Private Function CreateDataSet() As DataSet
    'creating a DataSet object for tables
    Dim dataset As DataSet = New DataSet()
    ' creating the student table
    Dim Students As DataTable = CreateStudentTable()
    dataset.Tables.Add(Students)
    Return dataset
  End Function
  Private Function CreateStudentTable() As DataTable
    Dim Students As DataTable
    Students = New DataTable("Student")
    ' adding columns
    AddNewColumn(Students, "System.Int32", "StudentID")
    AddNewColumn(Students, "System.String", "StudentName")
    AddNewColumn(Students, "System.String", "StudentCity")
    ' adding rows
    AddNewRow(Students, 1, "Zara Ali", "Kolkata")
    AddNewRow(Students, 2, "Shreya Sharma", "Delhi")
    AddNewRow(Students, 3, "Rini Mukherjee", "Hyderabad")
    AddNewRow(Students, 4, "Sunil Dubey", "Bikaner")
    AddNewRow(Students, 5, "Rajat Mishra", "Patna")
    Return Students
  End Function
  Private Sub AddNewColumn(ByRef table As DataTable, _
  ByVal columnType As String, ByVal columnName As String)
     Dim column As DataColumn = _
      table.Columns.Add(columnName, Type.GetType(columnType))
  End Sub
  'adding data into the table
  Private Sub AddNewRow(ByRef table As DataTable, ByRef id As Integer,_
  ByRef name As String, ByRef city As String)
     Dim newrow As DataRow = table.NewRow()
     newrow("StudentID") = id
     newrow("StudentName") = name
     newrow("StudentCity") = city
     table.Rows.Add(newrow)
  End Sub
  Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
     Dim ds As New DataSet
     ds = CreateDataSet()
     DataGridView1.DataSource = ds.Tables("Student")
  End Sub
End Class
```

When the above code is executed and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window:



Clicking the Fill button displays the table on the data grid view control:



	StudentID	StudentName	StudentCity
▶	1	Zara Ali	Delhi
	2	Shreya Sharma	Delhi
	3	Rini Mukherjee	Hyderabad
	4	Sunil Dubey	Bikaner
	5	Rajat Mishra	Patna
*			

VB.Net - Excel Sheet

VB.Net provides support for interoperability between the COM object model of Microsoft Excel 2010 and your application.

To avail this interoperability in your application, you need to import the namespace **Microsoft.Office.Interop.Excel** in your Windows Form Application.

Creating an Excel Application from VB.Net

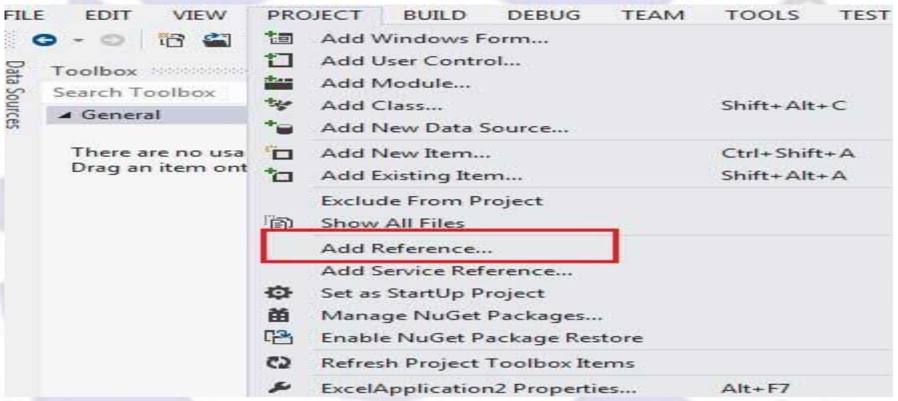
Let's start with creating a Window Forms Application by following the following steps in Microsoft Visual Studio: **File -> New Project -> Windows Forms Applications**

Finally, select OK, Microsoft Visual Studio creates your project and displays following **Form1**.

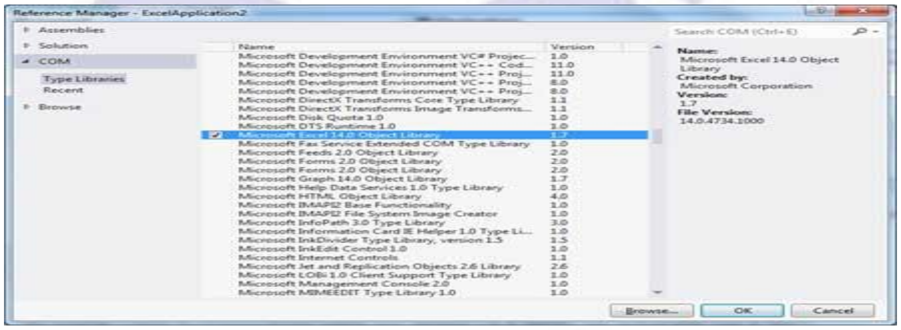
Insert a Button control Button1 in the form.

Add a reference to Microsoft Excel Object Library to your project. To do this:

- Select Add Reference from the Project Menu.



- On the COM tab, locate Microsoft Excel Object Library and then click Select.



	Name	Version
	Microsoft Development Environment VC# Projec...	1.0
	Microsoft Development Environment VC++ Cod...	11.0
	Microsoft Development Environment VC++ Proj...	11.0
	Microsoft Development Environment VC++ Proj...	8.0
	Microsoft Development Environment VC++ Proj...	8.0
	Microsoft DirectX Transforms Core Type Library	1.1
	Microsoft DirectX Transforms Image Transforms...	1.1
	Microsoft Disk Quota 1.0	1.0
	Microsoft DTS Runtime 1.0	1.0
✓	Microsoft Excel 14.0 Object Library	1.7
	Microsoft Fax Service Extended COM Type Library	1.0
	Microsoft Feeds 2.0 Object Library	2.0
	Microsoft Forms 2.0 Object Library	2.0
	Microsoft Forms 2.0 Object Library	2.0
	Microsoft Graph 14.0 Object Library	1.7
	Microsoft Help Data Services 1.0 Type Library	1.0
	Microsoft HTML Object Library	4.0
	Microsoft IMAPI2 Base Functionality	1.0
	Microsoft IMAPI2 File System Image Creator	1.0
	Microsoft InfoPath 3.0 Type Library	3.0
	Microsoft Information Card IE Helper 1.0 Type Li...	1.0
	Microsoft InkDivider Type Library, version 1.5	1.5
	Microsoft InkEdit Control 1.0	1.0
	Microsoft Internet Controls	1.1
	Microsoft Jet and Replication Objects 2.6 Library	2.6
	Microsoft LOBi 1.0 Client Support Type Library	1.0
	Microsoft Management Console 2.0	1.0
	Microsoft MIMEEDIT Type Library 1.0	1.0

Name:
Microsoft Excel 14.0 Object Library
Created by:
Microsoft Corporation
Version:
1.7
File Version:
14.0.4734.1000

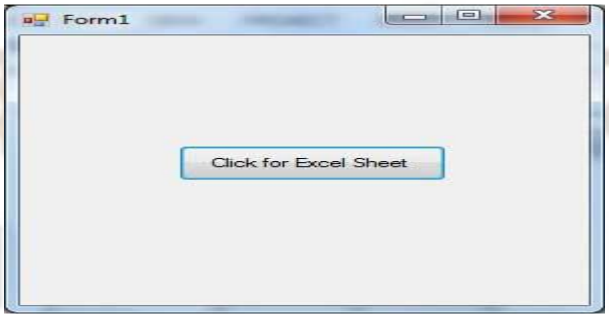
- Click OK.

Double click the code window and populate the Click event of Button1, as shown below.

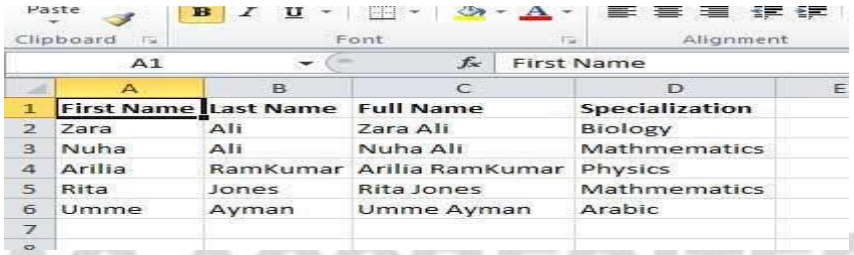
```
'  Add the following code snippet on top of Form1.vb
Imports Excel = Microsoft.Office.Interop.Excel
Public Class Form1
   Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
      Dim appXL As Excel.Application
      Dim wbXl As Excel.Workbook
      Dim shXL As Excel.Worksheet
Dim raXL As Excel.Range
' Start Excel and get Application object.
appXL = CreateObject("Excel.Application")
appXL.Visible = True
' Add a new workbook.
wbXl = appXL.Workbooks.Add
shXL = wbXl.ActiveSheet
' Add table headers going cell by cell.
shXL.Cells(1, 1).Value = "First Name"
shXL.Cells(1, 2).Value = "Last Name"
shXL.Cells(1, 3).Value = "Full Name"
shXL.Cells(1, 4).Value = "Specialization"
' Format A1:D1 as bold, vertical alignment = center.
With shXL.Range("A1", "D1")
    .Font.Bold = True
    .VerticalAlignment = Excel.XlVAlign.xlVAlignCenter
End With
' Create an array to set multiple values at once.
Dim students(5, 2) As String
students(0, 0) = "Zara"
students(0, 1) = "Ali"
students(1, 0) = "Nuha"
students(1, 1) = "Ali"
students(2, 0) = "Arilia"
students(2, 1) = "RamKumar"
students(3, 0) = "Rita"
students(3, 1) = "Jones"
students(4, 0) = "Umme"
students(4, 1) = "Ayman"
' Fill A2:B6 with an array of values (First and Last Names).
shXL.Range("A2", "B6").Value = students
 ' Fill C2:C6 with a relative formula (=A2 & " " & B2).
raXL = shXL.Range("C2", "C6")
raXL.Formula = "=A2 & "" "" & B2"
 ' Fill D2:D6 values.
With shXL
```

```
        .Cells(2, 4).Value = "Biology"
        .Cells(3, 4).Value = "Mathmematics"
        .Cells(4, 4).Value = "Physics"
        .Cells(5, 4).Value = "Mathmematics"
        .Cells(6, 4).Value = "Arabic"
    End With
    ' AutoFit columns A:D.
    raXL = shXL.Range("A1", "D1")
    raXL.EntireColumn.AutoFit()
     ' Make sure Excel is visible and give the user control
    ' of Excel's lifetime.
    appXL.Visible = True
    appXL.UserControl = True
     ' Release object references.
    raXL = Nothing
    shXL = Nothing
    wbXl = Nothing
    appXL.Quit()
    appXL = Nothing
    Exit Sub
Err_Handler:
    MsgBox(Err.Description, vbCritical, "Error: " & Err.Number)
  End Sub
End Class
```

When the above code is executed and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window:



Clicking on the Button would display the following excel sheet. You will be asked to save the workbook.



	A	B	C	D	E
1	First Name	Last Name	Full Name	Specialization	
2	Zara	Ali	Zara Ali	Biology	
3	Nuha	Ali	Nuha Ali	Mathmematics	
4	Arilia	RamKumar	Arilia RamKumar	Physics	
5	Rita	Jones	Rita Jones	Mathmematics	
6	Umme	Ayman	Umme Ayman	Arabic	
7					

VB.Net - Send Email

VB.Net allows sending e-mails from your application. The **System.Net.Mail** namespace contains classes used for sending e-mails to a Simple Mail Transfer Protocol (SMTP) server for delivery.

The following table lists some of these commonly used classes:

S.N	Class	Description
1	**Attachment**	Represents an attachment to an e-mail.
2	**AttachmentCollection**	Stores attachments to be sent as part of an e-mail message.
3	**MailAddress**	Represents the address of an electronic mail sender or recipient.
4	**MailAddressCollection**	Stores e-mail addresses that are associated with an e-mail message.
5	**MailMessage**	Represents an e-mail message that can be sent using the SmtpClient class.
6	**SmtpClient**	Allows applications to send e-mail by using the Simple Mail Transfer Protocol (SMTP).
7	**SmtpException**	Represents the exception that is thrown when the SmtpClient is not able to complete a Send or SendAsync operation.

The SmtpClient Class

The SmtpClient class allows applications to send e-mail by using the Simple Mail Transfer Protocol (SMTP).

Following are some commonly used properties of the SmtpClient class:

S.N	Property	Description
1	**ClientCertificates**	Specifies which certificates should be used to establish the Secure Sockets Layer (SSL) connection.
2	**Credentials**	Gets or sets the credentials used to authenticate the sender.

3	**EnableSsl**	Specifies whether the SmtpClient uses Secure Sockets Layer (SSL) to encrypt the connection.
4	**Host**	Gets or sets the name or IP address of the host used for SMTP transactions.
5	**Port**	Gets or sets the port used for SMTP transactions.
6	**Timeout**	Gets or sets a value that specifies the amount of time after which a synchronous Send call times out.
7	**UseDefaultCredentials**	Gets or sets a Boolean value that controls whether the DefaultCredentials are sent with requests.

Following are some commonly used methods of the SmtpClient class:

S.N	**Method & Description**
1	**Dispose** Sends a QUIT message to the SMTP server, gracefully ends the TCP connection, and releases all resources used by the current instance of the SmtpClient class.
2	**Dispose(Boolean)** Sends a QUIT message to the SMTP server, gracefully ends the TCP connection, releases all resources used by the current instance of the SmtpClient class, and optionally disposes of the managed resources.
3	**OnSendCompleted** Raises the SendCompleted event.
4	**Send(MailMessage)** Sends the specified message to an SMTP server for delivery.
5	**Send(String, String, String, String)** Sends the specified e-mail message to an SMTP server for delivery. The message sender, recipients, subject, and message body are specified using String objects.
6	**SendAsync(MailMessage, Object)** Sends the specified e-mail message to an SMTP server for delivery. This method does not block the calling thread and allows the caller to pass an object to the method that is invoked when the operation completes.
7	**SendAsync(String, String, String, String, Object)** Sends an e-mail message to an SMTP server for delivery. The message sender, recipients, subject, and message body are specified using String objects. This method does not block the calling thread

	and allows the caller to pass an object to the method that is invoked when the operation completes.
8	**SendAsyncCancel** Cancels an asynchronous operation to send an e-mail message.
9	**SendMailAsync(MailMessage)** Sends the specified message to an SMTP server for delivery as an asynchronous operation.
10	**SendMailAsync(String, String, String, String)** Sends the specified message to an SMTP server for delivery as an asynchronous operation. . The message sender, recipients, subject, and message body are specified using String objects.
11	**ToString** Returns a string that represents the current object.

The following example demonstrates how to send mail using the SmtpClient class. Following points are to be noted in this respect:

- You must specify the SMTP host server that you use to send e-mail. The **Host** and **Port** properties will be different for different host server. We will be using gmail server.
- You need to give the **Credentials** for authentication, if required by the SMTP server.
- You should also provide the email address of the sender and the e-mail address or addresses of the recipients using the **MailMessage.From** and **MailMessage.To** properties, respectively.
- You should also specify the message content using the **MailMessage.Body** property.

Example

In this example, let us create a simple application that would send an e-mail. Take the following steps:

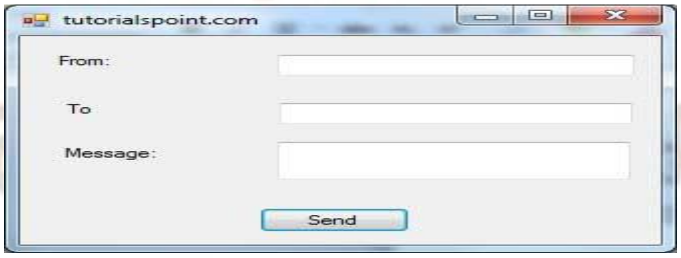
- Add three labels, three text boxes and a button control in the form.
- Change the text properties of the labels to - 'From', 'To:' and 'Message:' respectively.
- Change the name properties of the texts to txtFrom, txtTo and txtMessage respectively.
- Change the text property of the button control to 'Send'
- Add the following code in the code editor.

```
Imports System.Net.Mail
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspoint.com"
  End Sub
  Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Try
```

```
        Dim Smtp_Server As New SmtpClient
        Dim e_mail As New MailMessage()
        Smtp_Server.UseDefaultCredentials = False
        Smtp_Server.Credentials = New Net.NetworkCredential("username@gmail.com",
"password")
        Smtp_Server.Port = 587
        Smtp_Server.EnableSsl = True
        Smtp_Server.Host = "smtp.gmail.com"
        e_mail = New MailMessage()
        e_mail.From = New MailAddress(txtFrom.Text)
        e_mail.To.Add(txtTo.Text)
        e_mail.Subject = "Email Sending"
        e_mail.IsBodyHtml = False
        e_mail.Body = txtMessage.Text
        Smtp_Server.Send(e_mail)
        MsgBox("Mail Sent")
      Catch error_t As Exception
        MsgBox(error_t.ToString)
      End Try

  End Sub
```

You must provide your gmail address and real password for credentials.

When the above code is executed and run using **Start** button available at the Microsoft Visual Studio tool bar, it will show the following window, which you will use to send your e-mails, try it yourself.



VB.Net - XML Processing

The Extensible Markup Language (XML) is a markup language much like HTML or SGML. This is recommended by the World Wide Web Consortium and available as an open standard.

The **System.Xml** namespace in the .Net Framework contains classes for processing XML documents. Following are some of the commonly used classes in the System.Xml namespace.

S.N	Class	Description
1	**XmlAttribute**	Represents an attribute. Valid and default values for the attribute are defined in a document type definition (DTD) or schema.
2	**XmlCDataSection**	Represents a CDATA section.
3	**XmlCharacterData**	Provides text manipulation methods that are used by several classes.
4	**XmlComment**	Represents the content of an XML comment.
5	**XmlConvert**	Encodes and decodes XML names and provides methods for converting between common language runtime types and XML Schema definition language (XSD) types. When converting data types, the values returned are locale independent.
6	**XmlDeclaration**	Represents the XML declaration node <?xml version='1.0'...?>.
7	**XmlDictionary**	Implements a dictionary used to optimize Windows Communication Foundation (WCF)'s XML reader/writer implementations.
8	**XmlDictionaryReader**	An abstract class that the Windows Communication Foundation (WCF) derives from XmlReader to do serialization and deserialization.
9	**XmlDictionaryWriter**	Represents an abstract class that Windows Communication Foundation (WCF) derives from XmlWriter to do serialization and deserialization.
10	**XmlDocument**	Represents an XML document.
11	**XmlDocumentFragment**	Represents a lightweight object that is useful for tree insert operations.
12	**XmlDocumentType**	Represents the document type declaration.
13	**XmlElement**	Represents an element.
14	**XmlEntity**	Represents an entity declaration, such as <!ENTITY... >.
15	**XmlEntityReference**	Represents an entity reference node.

16	**XmlException**	Returns detailed information about the last exception.
17	**XmlImplementation**	Defines the context for a set of XmlDocument objects.
18	**XmlLinkedNode**	Gets the node immediately preceding or following this node.
19	**XmlNode**	Represents a single node in the XML document.
20	**XmlNodeList**	Represents an ordered collection of nodes.
21	**XmlNodeReader**	Represents a reader that provides fast, non-cached forward only access to XML data in an XmlNode.
22	**XmlNotation**	Represents a notation declaration, such as <!NOTATION... >.
23	**XmlParserContext**	Provides all the context information required by the XmlReader to parse an XML fragment.
24	**XmlProcessingInstruction**	Represents a processing instruction, which XML defines to keep processor-specific information in the text of the document.
25	**XmlQualifiedName**	Represents an XML qualified name.
26	**XmlReader**	Represents a reader that provides fast, noncached, forward-only access to XML data.
27	**XmlReaderSettings**	Specifies a set of features to support on the XmlReader object created by the Create method.
28	**XmlResolver**	Resolves external XML resources named by a Uniform Resource Identifier (URI).
29	**XmlSecureResolver**	Helps to secure another implementation of XmlResolver by wrapping the XmlResolver object and restricting the resources that the underlying XmlResolver has access to.
30	**XmlSignificantWhitespace**	Represents white space between markup in a mixed content node or white space within an xml:space= 'preserve' scope. This is also referred to as significant white space.
31	**XmlText**	Represents the text content of an element or attribute.
32	**XmlTextReader**	Represents a reader that provides fast, non-cached, forward-only access to XML data.
33	**XmlTextWriter**	Represents a writer that provides a fast, non-

		cached, forward-only way of generating streams or files containing XML data that conforms to the W3C Extensible Markup Language (XML) 1.0 and the Namespaces in XML recommendations.
34	**XmlUrlResolver**	Resolves external XML resources named by a Uniform Resource Identifier (URI).
35	**XmlWhitespace**	Represents white space in element content.
36	**XmlWriter**	Represents a writer that provides a fast, non-cached, forward-only means of generating streams or files containing XML data.
37	**XmlWriterSettings**	Specifies a set of features to support on the XmlWriter object created by the XmlWriter.Create method.

XML Parser APIs

The two most basic and broadly used APIs to XML data are the SAX and DOM interfaces.

- **Simple API for XML (SAX)** : Here, you register callbacks for events of interest and then let the parser proceed through the document. This is useful when your documents are large or you have memory limitations, it parses the file as it reads it from disk, and the entire file is never stored in memory.
- **Document Object Model (DOM) API** : This is World Wide Web Consortium recommendation wherein the entire file is read into memory and stored in a hierarchical (tree-based) form to represent all the features of an XML document.

SAX obviously can't process information as fast as DOM can when working with large files. On the other hand, using DOM exclusively can really kill your resources, especially if used on a lot of small files. SAX is read-only, while DOM allows changes to the XML file. Since these two different APIs literally complement each other there is no reason why you can't use them both for large projects.

For all our XML code examples, let's use a simple XML file movies.xml as an input:

```
<?xml version="1.0"?>
<collection shelf="New Arrivals">
<movie title="Enemy Behind">
   <type>War, Thriller</type>
   <format>DVD</format>
   <year>2003</year>
   <rating>PG</rating>
   <stars>10</stars>
   <description>Talk about a US-Japan war</description>
</movie>
<movie title="Transformers">
   <type>Anime, Science Fiction</type>
   <format>DVD</format>
   <year>1989</year>
   <rating>R</rating>
   <stars>8</stars>
   <description>A schientific fiction</description>
</movie>
   <movie title="Trigun">
   <type>Anime, Action</type>
   <format>DVD</format>
   <episodes>4</episodes>
   <rating>PG</rating>
   <stars>10</stars>
   <description>Vash the Stampede!</description>
</movie>
<movie title="Ishtar">
   <type>Comedy</type>
   <format>VHS</format>
   <rating>PG</rating>
   <stars>2</stars>
   <description>Viewable boredom</description>
</movie>
</collection>
```

Parsing XML with SAX API

In SAX model, you use the **XmlReader** and **XmlWriter** classes to work with the XML data.

The **XmlReader** class is used to read XML data in a fast, forward-only and non-cached manner. It reads an XML document or a stream.

Example 1

This example demonstrates reading XML data from the file movies.xml.

Take the following steps:

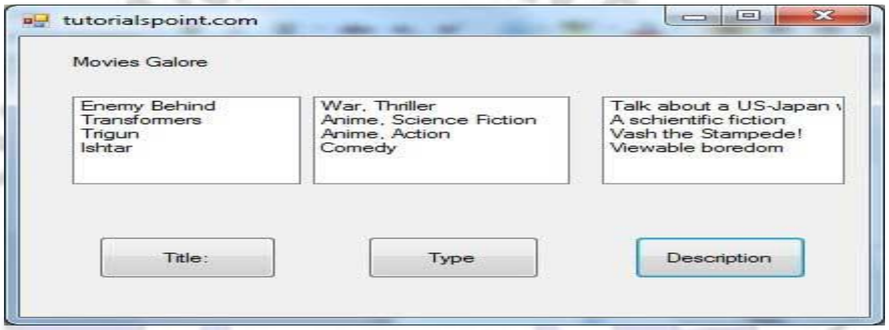
- Add the movies.xml file in the bin\Debug folder of your application.
- Import the System.Xml namespace in Form1.vb file.

- Add a label in the form and change its text to 'Movies Galore'.
- Add three list boxes and three buttons to show the title, type and description of a movie from the xml file.
- Add the following code using the code editor window.

```
Imports System.Xml
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspoint.com"
  End Sub
  Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    ListBox1().Items.Clear()
    Dim xr As XmlReader = XmlReader.Create("movies.xml")
    Do While xr.Read()
       If xr.NodeType = XmlNodeType.Element AndAlso xr.Name = "movie" Then
          ListBox1.Items.Add(xr.GetAttribute(0))
       End If
    Loop
  End Sub
  Private Sub Button2_Click(sender As Object, e As EventArgs) Handles Button2.Click
    ListBox2().Items.Clear()
    Dim xr As XmlReader = XmlReader.Create("movies.xml")
    Do While xr.Read()
       If xr.NodeType = XmlNodeType.Element AndAlso xr.Name = "type" Then
          ListBox2.Items.Add(xr.ReadElementString)
       Else
          xr.Read()
       End If
    Loop
  End Sub
  Private Sub Button3_Click(sender As Object, e As EventArgs) Handles Button3.Click
    ListBox3().Items.Clear()
    Dim xr As XmlReader = XmlReader.Create("movies.xml")
    Do While xr.Read()
       If xr.NodeType = XmlNodeType.Element AndAlso xr.Name = "description" Then
          ListBox3.Items.Add(xr.ReadElementString)
```

```
        Else
            xr.Read()
        End If
    Loop
  End Sub
End Class
```

Execute and run the above code using **Start** button available at the Microsoft Visual Studio tool bar. Clicking on the buttons would display, title, type and description of the movies from the file.



```
Enemy Behind          War, Thriller                Talk about a US-Japan w
Transformers          Anime, Science Fiction       A schientific fiction
Trigun                Anime, Action                Vash the Stampede!
Ishtar                Comedy                       Viewable boredom
```

The **XmlWriter** class is used to write XML data into a stream, a file or a TextWriter object. It also works in a forward-only, non-cached manner.

Example 2

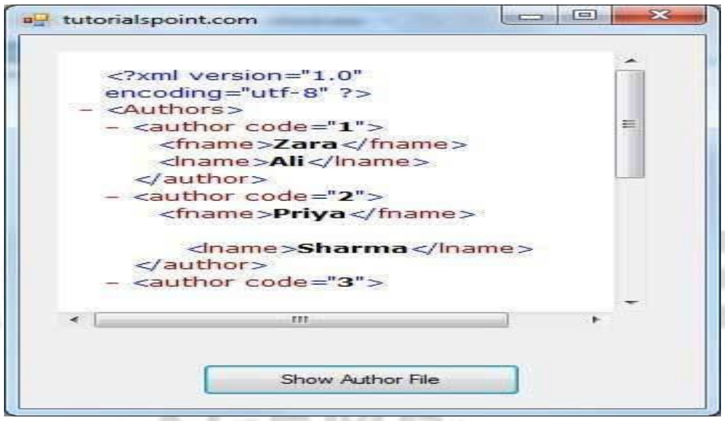
Let us create an XML file by adding some data at runtime. Take the following steps:

- Add a WebBrowser control and a button control in the form.
- Change the Text property of the button to Show Authors File.
- Add the following code in the code editor.

```
Imports System.Xml
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspoint.com"
  End Sub
  Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    Dim xws As XmlWriterSettings = New XmlWriterSettings()
    xws.Indent = True
    xws.NewLineOnAttributes = True
    Dim xw As XmlWriter = XmlWriter.Create("authors.xml", xws)
    xw.WriteStartDocument()
    xw.WriteStartElement("Authors")
    xw.WriteStartElement("author")
```

```
      xw.WriteAttributeString("code", "1")
      xw.WriteElementString("fname", "Zara")
      xw.WriteElementString("lname", "Ali")
      xw.WriteEndElement()
      xw.WriteStartElement("author")
      xw.WriteAttributeString("code", "2")
      xw.WriteElementString("fname", "Priya")
      xw.WriteElementString("lname", "Sharma")
      xw.WriteEndElement()
      xw.WriteStartElement("author")
      xw.WriteAttributeString("code", "3")
      xw.WriteElementString("fname", "Anshuman")
      xw.WriteElementString("lname", "Mohan")
      xw.WriteEndElement()
      xw.WriteStartElement("author")
      xw.WriteAttributeString("code", "4")
      xw.WriteElementString("fname", "Bibhuti")
      xw.WriteElementString("lname", "Banerjee")
      xw.WriteEndElement()
      xw.WriteStartElement("author")
      xw.WriteAttributeString("code", "5")
      xw.WriteElementString("fname", "Riyan")
      xw.WriteElementString("lname", "Sengupta")
      xw.WriteEndElement()
      xw.WriteEndElement()
      xw.WriteEndDocument()
      xw.Flush()
      xw.Close()
      WebBrowser1.Url = New Uri(AppDomain.CurrentDomain.BaseDirectory + "authors.xml")
   End Sub
End Class
```

Execute and run the above code using **Start** button available at the Microsoft Visual Studio tool bar. Clicking on the Show Author File would display the newly created authors.xml file on the web browser.



```
<?xml version="1.0"
encoding="utf-8" ?>
- <Authors>
- <author code="1">
    <fname>Zara</fname>
    <lname>Ali</lname>
  </author>
- <author code="2">
    <fname>Priya</fname>

      <lname>Sharma</lname>
  </author>
- <author code="3">
```

Parsing XML with DOM API

According to the Document Object Model (DOM), an XML document consists of nodes and attributes of the nodes. The **XmlDocument** class is used to implement the XML DOM parser of the .Net Framework. It also allows you to modify an existing XML document by inserting, deleting or updating data in the document.

Following are some of the commonly used methods of the **XmlDocument** class:

S.N	Method Name & Description
1	**AppendChild** Adds the specified node to the end of the list of child nodes, of this node.
2	**CreateAttribute(String)** Creates an XmlAttribute with the specified Name.
3	**CreateComment** Creates an XmlComment containing the specified data.
4	**CreateDefaultAttribute** Creates a default attribute with the specified prefix, local name and namespace URI.
5	**CreateElement(String)** Creates an element with the specified name.
6	**CreateNode(String, String, String)** Creates an XmlNode with the specified node type, Name, and NamespaceURI.
7	**CreateNode(XmlNodeType, String, String)** Creates an XmlNode with the specified XmlNodeType, Name, and NamespaceURI.
8	**CreateNode(XmlNodeType, String, String, String)** Creates a XmlNode with the specified XmlNodeType, Prefix, Name, and NamespaceURI.
9	**CreateProcessingInstruction** Creates an XmlProcessingInstruction with the specified name and

	data.
10	**CreateSignificantWhitespace** Creates an XmlSignificantWhitespace node.
11	**CreateTextNode** Creates an XmlText with the specified text.
12	**CreateWhitespace** Creates an XmlWhitespace node.
13	**CreateXmlDeclaration** Creates an XmlDeclaration node with the specified values.
14	**GetElementById** Gets the XmlElement with the specified ID.
15	**GetElementsByTagName(String)** Returns an XmlNodeList containing a list of all descendant elements that match the specified Name.
16	**GetElementsByTagName(String, String)** Returns an XmlNodeList containing a list of all descendant elements that match the specified LocalName and NamespaceURI.
17	**InsertAfter** Inserts the specified node immediately after the specified reference node.
18	**InsertBefore** Inserts the specified node immediately before the specified reference node.
19	**Load(Stream)** Loads the XML document from the specified stream.
20	**Load(String)** Loads the XML document from the specified URL.
21	**Load(TextReader)** Loads the XML document from the specified TextReader.
22	**Load(XmlReader)** Loads the XML document from the specified XmlReader.
23	**LoadXml** Loads the XML document from the specified string.
24	**PrependChild** Adds the specified node to the beginning of the list of child nodes for this node.
25	**ReadNode** Creates an XmlNode object based on the information in the

	XmlReader. The reader must be positioned on a node or attribute.
26	**RemoveAll** Removes all the child nodes and/or attributes of the current node.
27	**RemoveChild** Removes specified child node.
28	**ReplaceChild** Replaces the child node oldChild with newChild node.
29	**Save(Stream)** Saves the XML document to the specified stream.
30	**Save(String)** Saves the XML document to the specified file.
31	**Save(TextWriter)** Saves the XML document to the specified TextWriter.
32	**Save(XmlWriter)** Saves the XML document to the specified XmlWriter.

Example 3

In this example, let us insert some new nodes in the xml document authors.xml and then show all the authors' first names in a list box.

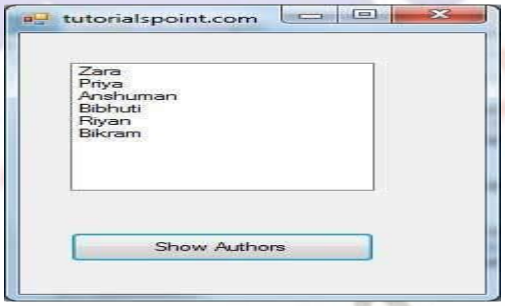
Take the following steps:

- Add the authors.xml file in the bin/Debug folder of your application(it should be there if you have tried the last example)
- Import the System.Xml namespace
- Add a list box and a button control in the form and set the text property of the button control to Show Authors.
- Add the following code using the code editor.

```
Imports System.Xml
Public Class Form1
  Private Sub Form1_Load(sender As Object, e As EventArgs) Handles MyBase.Load
    ' Set the caption bar text of the form.
    Me.Text = "tutorialspoint.com"
  End Sub
  Private Sub Button1_Click(sender As Object, e As EventArgs) Handles Button1.Click
    ListBox1.Items.Clear()
    Dim xd As XmlDocument = New XmlDocument()
    xd.Load("authors.xml")
    Dim newAuthor As XmlElement = xd.CreateElement("author")
```

```
      newAuthor.SetAttribute("code", "6")
      Dim fn As XmlElement = xd.CreateElement("fname")
      fn.InnerText = "Bikram"
      newAuthor.AppendChild(fn)
      Dim ln As XmlElement = xd.CreateElement("lname")
      ln.InnerText = "Seth"
      newAuthor.AppendChild(ln)
      xd.DocumentElement.AppendChild(newAuthor)
      Dim tr As XmlTextWriter = New XmlTextWriter("movies.xml", Nothing)
      tr.Formatting = Formatting.Indented
      xd.WriteContentTo(tr)
      tr.Close()
      Dim nl As XmlNodeList = xd.GetElementsByTagName("fname")
      For Each node As XmlNode In nl
         ListBox1.Items.Add(node.InnerText)
      Next node
   End Sub
End Class
```

Execute and run the above code using **Start** button available at the Microsoft Visual Studio tool bar. Clicking on the Show Author button would display the first names of all the authors including the one we have added at runtime.



```
Zara
Priya
Anshuman
Bibhuti
Riyan
Bikram
```

VB.Net - Web Programming

A dynamic web application consists of either or both of the following two types of programs:

- **Server-side scripting** - these are programs executed on a web server, written using server-side scripting languages like ASP (Active Server Pages) or JSP (Java Server Pages).
- **Client-side scripting** - these are programs executed on the browser, written using scripting languages like JavaScript, VBScript, etc.

ASP.Net is the .Net version of ASP, introduced by Microsoft, for creating dynamic web pages by using server-side scripts. ASP.Net applications are compiled codes written using the extensible and reusable components or objects present in .Net framework. These codes can use the entire hierarchy of classes in .Net framework.

The ASP.Net application codes could be written in either of the following languages:

- Visual Basic .Net
- C#
- Jscript
- J#

In this chapter, we will give a very brief introduction to writing ASP.Net applications using VB.Net. For detailed discussion, please consult the ASP.Net Tutorial.

ASP.Net Built-in Objects

ASP.Net has some built-in objects that run on a web server. These objects have methods, properties and collections that are used in application development.

The following table lists the ASP.Net built-in objects with a brief description:

Object	**Description**
Application	Describes the methods, properties, and collections of the object that stores information related to the entire Web application, including variables and objects that exist for the lifetime of the application. You use this object to store and retrieve information to be shared among all users of an application. For example, you can use an Application object to create an e-commerce page.
Request	Describes the methods, properties, and collections of the object that stores information related to the HTTP request. This includes forms, cookies, server variables, and certificate data. You use this object to access the information sent in a request from a browser to the server. For example, you can use a Request object to access information entered by a user in an HTML form.
Response	Describes the methods, properties, and collections of the object that stores information related to the server's response. This includes displaying content, manipulating headers, setting locales, and redirecting requests. You use this object to send information to the browser. For example, you use a Response object to send output from your scripts to a browser.
Server	Describes the methods and properties of the object that provides methods for various server tasks. With these methods you can execute code, get error conditions, encode text strings, create objects for use by the Web page, and map physical paths.

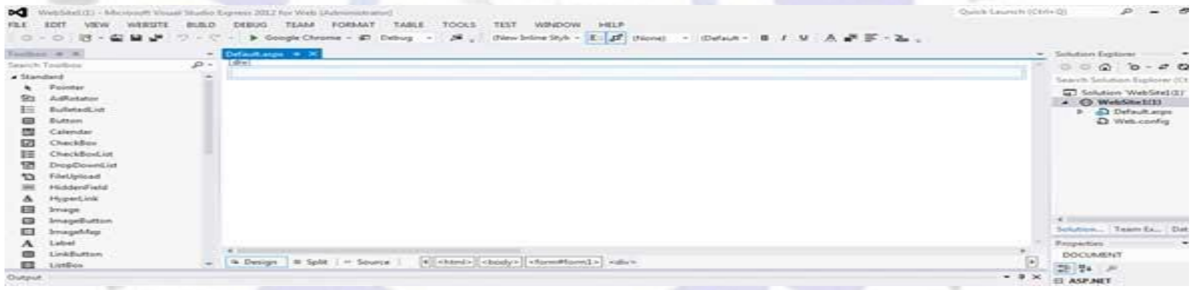
	You use this object to access various utility functions on the server. For example, you may use the Server object to set a time out for a script.
Session	Describes the methods, properties, and collections of the object that stores information related to the user's session, including variables and objects that exist for the lifetime of the session. You use this object to store and retrieve information about particular user sessions. For example, you can use Session object to keep information about the user and his preference and keep track of pending operations.

ASP.Net Programming Model

ASP.Net provides two types of programming models:

- **Web Forms** - this enables you to create the user interface and the application logic that would be applied to various components of the user interface.
- **WCF Services** - this enables you to remote access some server-side functionalities.

For this chapter, you need to use Visual Studio Web Developer, which is free. The IDE is almost same as you have already used for creating the Windows Applications.



Web Forms

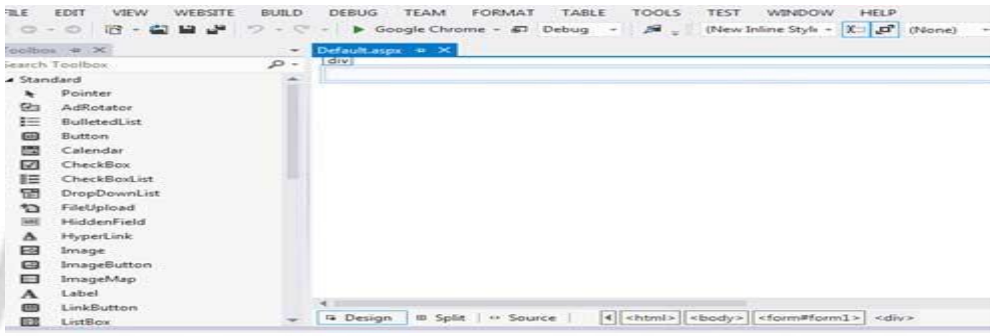
Web forms consists of:

- User interface
- Application logic

User interface consists of static HTML or XML elements and ASP.Net server controls. When you create a web application, HTML or XML elements and server controls are stored in a file with **.aspx** extension. This file is also called the page file.

The application logic consists of code applied to the user interface elements in the page. You write this code in any of .Net language like, VB.Net, or C#.

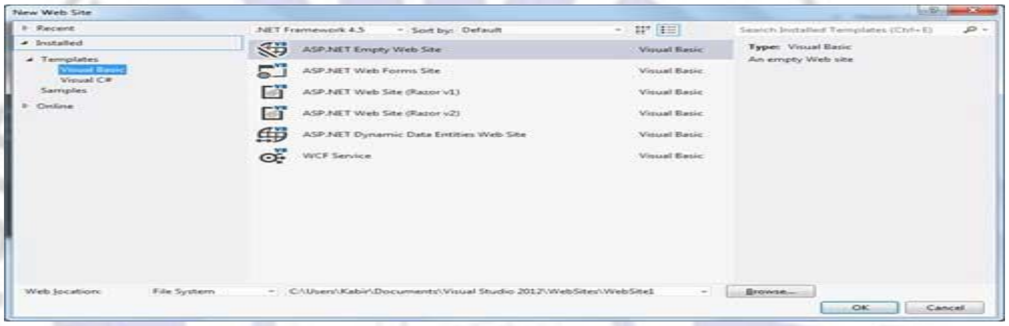
The following figure shows a Web Form in Design view:



Example

Let us create a new web site with a web form, which will show the current date and time, when a user clicks a button. Take the following steps:

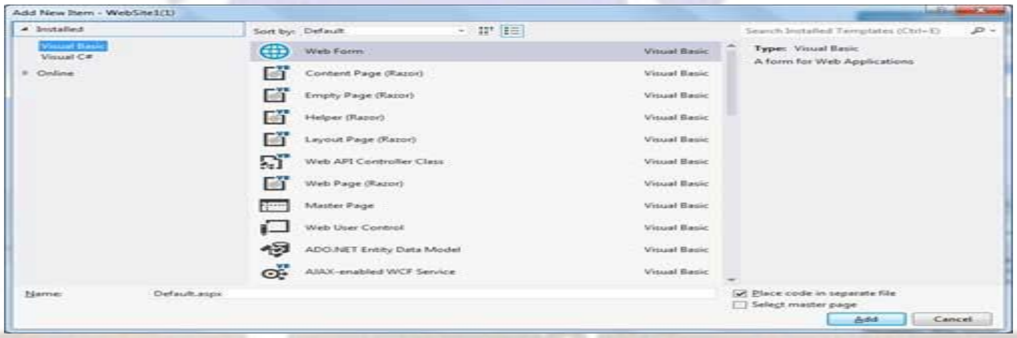
- Select File -> New -> Web Site. The New Web Site Dialog Box appears.



Type: Visual Basic
An empty Web site

Web location: File System C:\Users\Kabir\Documents\Visual Studio 2012\WebSites\WebSite1

- Select the ASP.Net Empty Web Site templates. Type a name for the web site and select a location for saving the files.
- You need to add a Default page to the site. Right click the web site name in the Solution Explorer and select Add New Item option from the context menu. The Add New Item dialog box is displayed:

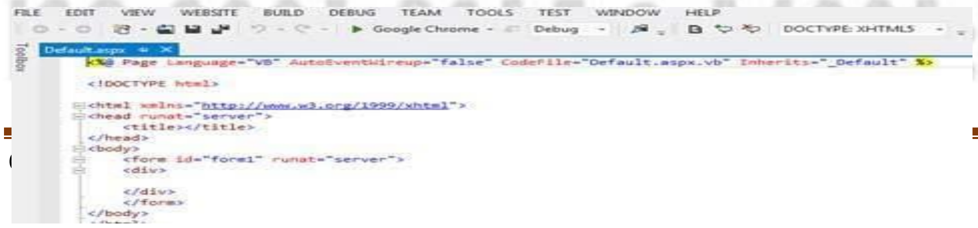


Type: Visual Basic
A form for Web Applications

Name: Default.aspx

☑ Place code in separate file
☐ Select master page

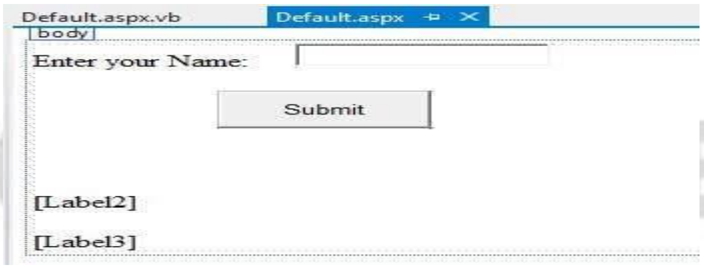
- Select Web Form option and provide a name for the default page. We have kept it as Default.aspx. Click the Add button.
- The Default page is shown in Source view


FILE EDIT VIEW WEBSITE BUILD DEBUG TEAM TOOLS TEST WINDOW HELP
Google Chrome
Debug
DOCTYPE: XHTML5
Toolbox
Default.aspx

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb" Inherits="_Default" %>

<!DOCTYPE html>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

    </div>
    </form>
</body>
```

- Set the title for the Default web page by adding a value to the

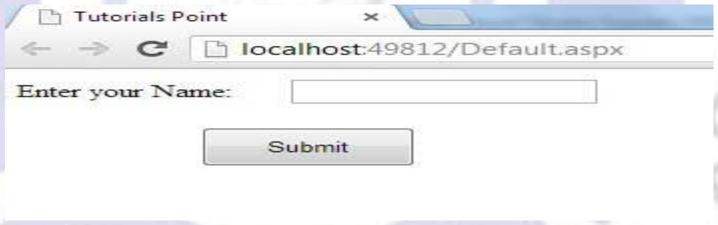
- To add controls on the web page, go to the design view. Add three labels, a text box and a button on the form.



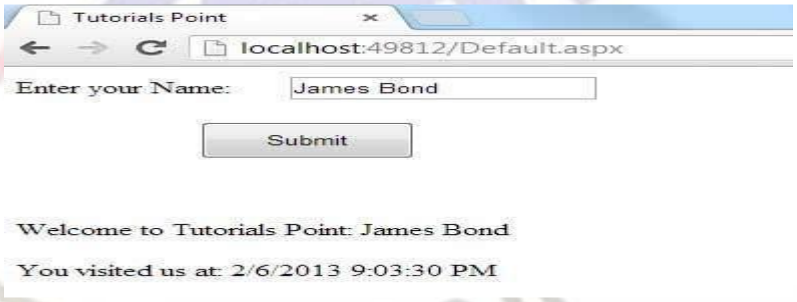
- Double-click the button and add the following code to the Click event of the button:

```
Protected Sub Button1_Click(sender As Object, e As EventArgs) _
Handles Button1.Click
   Label2.Visible = True
   Label2.Text = "Welcome to Tutorials Point: " + TextBox1.Text
   Label3.Text = "You visited us at: " + DateTime.Now.ToString()
End Sub
```

When the above code is executed and run using Start button available at the Microsoft Visual Studio tool bar, the following page opens in the browser:



Enter your name and click on the Submit button:



```
Welcome to Tutorials Point: James Bond
You visited us at: 2/6/2013 9:03:30 PM
```

Web Services

A web service is a web application, which is basically a class consisting of methods that could be used by other applications. It also follows a code-behind architecture like the ASP.Net web pages, although it does not have an user interface.

The previous versions of .Net Framework used this concept of ASP.Net Web Service, which had .asmx file extension. However, from .Net Framework 4.0 onwards, the Windows Communication Foundation (WCF) technology has evolved as the new successor of Web Services, .Net Remoting and some other related technologies. It has rather clubbed all these technologies together. In the next section, we will provide a brief introduction to Windows Communication Foundation(WCF).

If you are using previous versions of .Net Framework, you can still create traditional web services. Please consult ASP.Net - Web Services tutorial for detailed description.

Windows Communication Foundation

Windows Communication Foundation or WCF provides an API for creating distributed service-oriented applications, known as WCF Services.

Like Web services, WCF services also enable communication between applications. However, unlike web services, the communication here is not limited to HTTP only. WCF can be configured to be used over HTTP, TCP, IPC, and Message Queues. Another strong point in favour of WCF is, it provides support for duplex communication, whereas with web services we could achieve simplex communication only.

From beginners' point of view, writing a WCF service is not altogether so different from writing a Web Service. To keep the things simple, we will see how to:

- Create a WCF Service
- Create a Service Contract and define the operations
- Implement the contract
- Test the Service
- Utilize the Service

Example

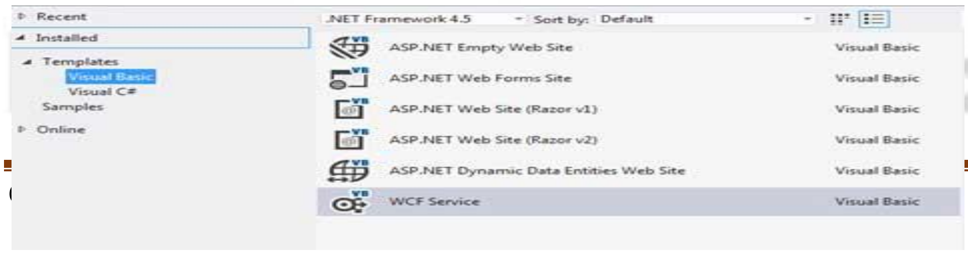
To understand the concept let us create a simplistic service that will provide stock price information. The clients can query about the name and price of a stock based on the stock symbol. To keep this example simple, the values are hardcoded in a two-dimensional array. This service will have two methods:

- GetPrice Method - it will return the price of a stock, based on the symbol provided.
- GetName Method - it will return the name of the stock, based on the symbol provided.

Creating a WCF Service

Take the following steps:

- Open VS Express for Web 2012
- Select New Web Site to open the New Web Site dialog box.
- Select WCF Service template from list of templates:


Recent
Installed
Templates
Visual Basic
Visual C#
Samples
Online
.NET Framework 4.5
Sort by: Default
ASP.NET Empty Web Site — Visual Basic
ASP.NET Web Forms Site — Visual Basic
ASP.NET Web Site (Razor v1) — Visual Basic
ASP.NET Web Site (Razor v2) — Visual Basic
ASP.NET Dynamic Data Entities Web Site — Visual Basic
WCF Service — Visual Basic

- Select File System from the Web location drop-down list.

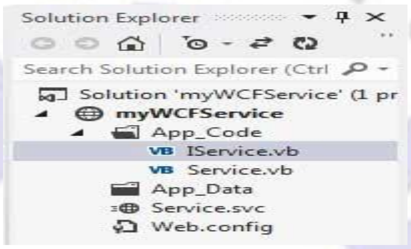
- Provide a name and location for the WCF Service and click OK.

  A new WCF Service is created.

  Creating a Service Contract and Defining the Operations

  A service contract defines the operation that a service performs. In the WCF Service application, you will find two files automatically created in the App_Code folder in the Solution Explorer

- IService.vb - this will have the service contract; in simpler words, it will have the interface for the service, with the definitions of methods the service will provide, which you will implement in your service.
- Service.vb - this will implement the service contract.



Replace the code of the IService.vb file with the given code:

```
Public Interface IService
    <OperationContract()>
    Function GetPrice(ByVal symbol As String) As Double
    <OperationContract()>
    Function GetName(ByVal symbol As String) As String
End Interface
```

Implementing the Contract

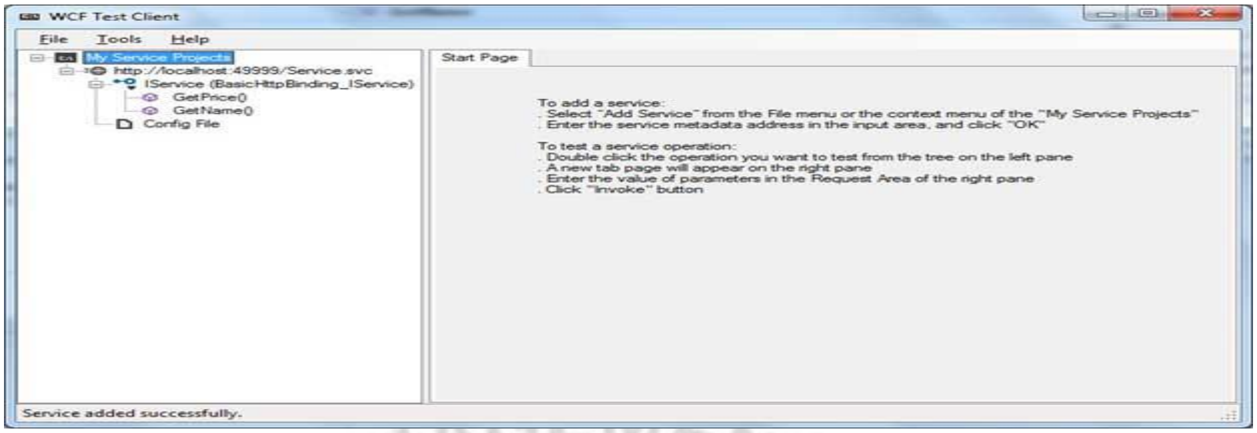
In the Service.vb file, you will find a class named **Service** which will implement the Service Contract defined in the **IService** interface.

Replace the code of IService.vb with the following code:

```
' NOTE: You can use the "Rename" command on the context menu to change the class name
"Service" in code, svc and config file together.
Public Class Service
   Implements IService
   Public Sub New()
   End Sub
   Dim stocks As String(,) =
   {
   {"RELIND", "Reliance Industries", "1060.15"},
   {"ICICI", "ICICI Bank", "911.55"},
   {"JSW", "JSW Steel", "1201.25"},
   {"WIPRO", "Wipro Limited", "1194.65"},
   {"SATYAM", "Satyam Computers", "91.10"}
   }
   Public Function GetPrice(ByVal symbol As String) As Double _
   Implements IService.GetPrice
     Dim i As Integer
     'it takes the symbol as parameter and returns price
     For i = 0 To i = stocks.GetLength(0) - 1
         If (String.Compare(symbol, stocks(i, 0)) = 0) Then
             Return Convert.ToDouble(stocks(i, 2))
         End If
     Next i
     Return 0
   End Function

   Public Function GetName(ByVal symbol As String) As String _
   Implements IService.GetName
     ' It takes the symbol as parameter and
     ' returns name of the stock
     Dim i As Integer
     For i = 0 To i = stocks.GetLength(0) - 1
         If (String.Compare(symbol, stocks(i, 0)) = 0) Then
             Return stocks(i, 1)
         End If
     Next i
     Return "Stock Not Found"
   End Function
End Class
```

Testing the Service

To run the WCF Service, so created, select the Debug->Start Debugging option from the menu bar. The output would be:



To add a service:
. Select "Add Service" from the File menu or the context menu of the "My Service Projects"
. Enter the service metadata address in the input area, and click "OK"

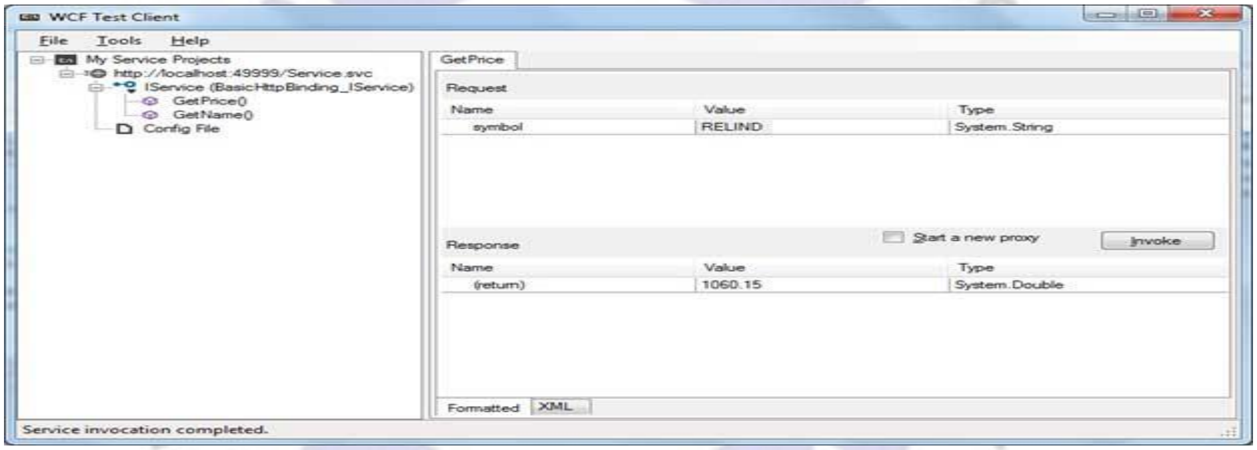
To test a service operation:
. Double click the operation you want to test from the tree on the left pane
. A new tab page will appear on the right pane
. Enter the value of parameters in the Request Area of the right pane
. Click "Invoke" button

Service added successfully.

For testing the service operations, double click the name of the operation from the tree on the left pane. A new tab will appear on the right pane.

Enter the value of parameters in the Request area of the right pane and click the 'Invoke' button.

The following diagram displays the result of testing the **GetPrice** operation:



Request

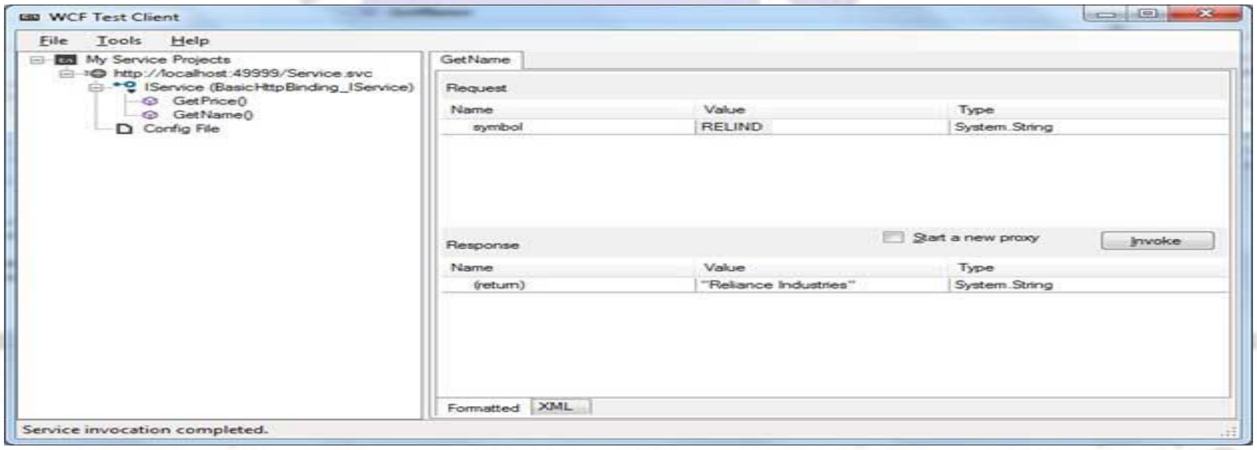
Name	Value	Type
symbol	RELIND	System.String

Response

Name	Value	Type
(return)	1060.15	System.Double

Service invocation completed.

The following diagram displays the result of testing the **GetName** operation:



Request

Name	Value	Type
symbol	RELIND	System.String

Response

Name	Value	Type
(return)	"Reliance Industries"	System.String

Service invocation completed.

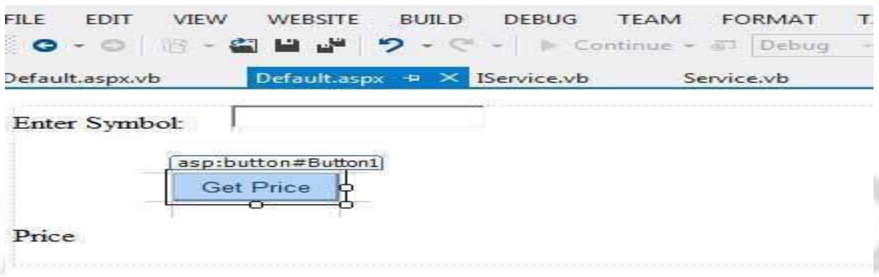
Utilizing the Service

Let us add a default page, a ASP.NET web form in the same solution from which we will be using the WCF Service we have just created.

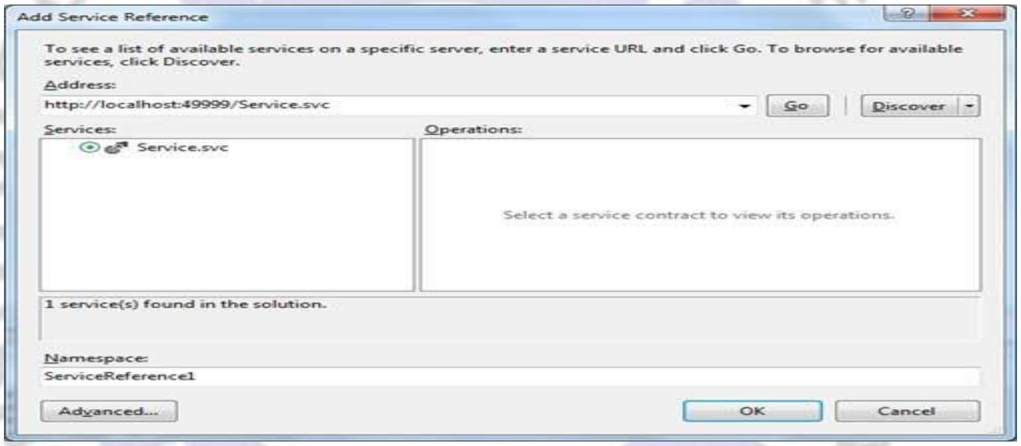
Take the following steps:

- Right click on the solution name in the Solution Explorer and add a new web form to the solution. It will be named Default.aspx.

- Add two labels, a text box and a button on the form.



- We need to add a service reference to the WCF service we just created. Right click the website in the Solution Explorer and select Add Service Reference option. This opens the Add Service Reference Dialog box.
- Enter the URL(location) of the Service in the Address text box and click the Go button. It creates a service reference with the default name **ServiceReference1**. Click the OK button.


Add Service Reference

To see a list of available services on a specific server, enter a service URL and click Go. To browse for available services, click Discover.

Address: http://localhost:49999/Service.svc

Services: Service.svc

Operations: Select a service contract to view its operations.

1 service(s) found in the solution.

Namespace: ServiceReference1

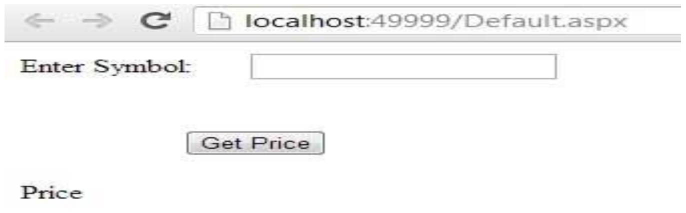
Adding the reference does two jobs for your project:

- Creates the Address and Binding for the service in the web.config file.
- Creates a proxy class to access the service.

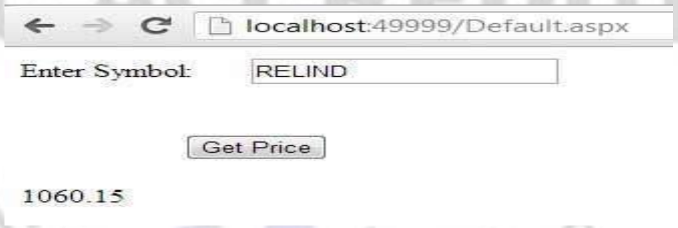
- Double click the Get Price button in the form, to enter the following code snippet on its Click event:

```
Partial Class _Default
  Inherits System.Web.UI.Page
  Protected Sub Button1_Click(sender As Object, e As EventArgs) _
  Handles Button1.Click
    Dim ser As ServiceReference1.ServiceClient = _
    New ServiceReference1.ServiceClient
    Label2.Text = ser.GetPrice(TextBox1.Text).ToString()
  End Sub
End Class
```

When the above code is executed and run using Start button available at the Microsoft Visual Studio tool bar, the following page opens in the browser:



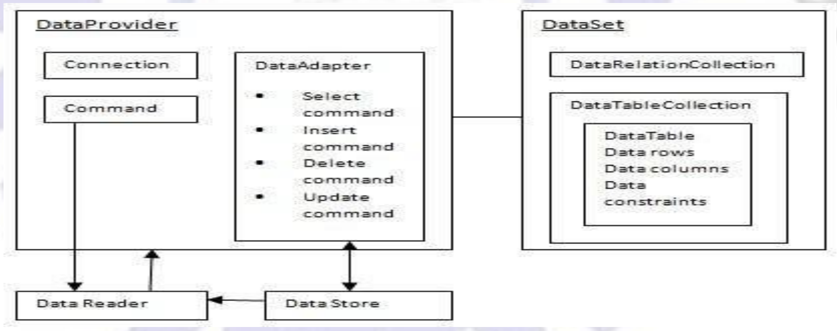
Enter a symbol and click the Get Price button to get the hard-coded price:



ADO.NET

ADO.NET provides a bridge between the front end controls and the back end database. The ADO.NET objects encapsulate all the data access operations and the controls interact with these objects to display data, thus hiding the details of movement of data.

The following figure shows the ADO.NET objects at a glance:



The DataSet Class

The dataset represents a subset of the database. It does not have a continuous connection to the database. To update the database a reconnection is required. The DataSet contains DataTable objects and DataRelation objects. The DataRelation objects represent the relationship between two tables.

Following table shows some important properties of the DataSet class:

Properties	Description
CaseSensitive	Indicates whether string comparisons within the data tables are case-sensitive.
Container	Gets the container for the component.
DataSetName	Gets or sets the name of the current data set.
DefaultViewManager	Returns a view of data in the data set.

DesignMode	Indicates whether the component is currently in design mode.
EnforceConstraints	Indicates whether constraint rules are followed when attempting any update operation.
Events	Gets the list of event handlers that are attached to this component.
ExtendedProperties	Gets the collection of customized user information associated with the DataSet.
HasErrors	Indicates if there are any errors.
IsInitialized	Indicates whether the DataSet is initialized.
Locale	Gets or sets the locale information used to compare strings within the table.
Namespace	Gets or sets the namespace of the DataSet.
Prefix	Gets or sets an XML prefix that aliases the namespace of the DataSet.
Relations	Returns the collection of DataRelation objects.
Tables	Returns the collection of DataTable objects.

The following table shows some important methods of the DataSet class:

Methods	**Description**
AcceptChanges	Accepts all changes made since the DataSet was loaded or this method was called.
BeginInit	Begins the initialization of the DataSet. The initialization occurs at run time.
Clear	Clears data.
Clone	Copies the structure of the DataSet, including all DataTable schemas, relations, and constraints. Does not copy any data.
Copy	Copies both structure and data.
CreateDataReader()	Returns a DataTableReader with one result set per DataTable, in the same sequence as the tables appear in the Tables collection.
CreateDataReader(DataTable[])	Returns a DataTableReader with one result set per DataTable.
EndInit	Ends the initialization of the data set.

Equals(Object)	Determines whether the specified Object is equal to the current Object.
Finalize	Free resources and perform other cleanups.
GetChanges	Returns a copy of the DataSet with all changes made since it was loaded or the AcceptChanges method was called.
GetChanges(DataRowState)	Gets a copy of DataSet with all changes made since it was loaded or the AcceptChanges method was called, filtered by DataRowState.
GetDataSetSchema	Gets a copy of XmlSchemaSet for the DataSet.
GetObjectData	Populates a serialization information object with the data needed to serialize the DataSet.
GetType	Gets the type of the current instance.
GetXML	Returns the XML representation of the data.
GetXMLSchema	Returns the XSD schema for the XML representation of the data.
HasChanges()	Gets a value indicating whether the DataSet has changes, including new, deleted, or modified rows.
HasChanges(DataRowState)	Gets a value indicating whether the DataSet has changes, including new, deleted, or modified rows, filtered by DataRowState.
IsBinarySerialized	Inspects the format of the serialized representation of the DataSet.
Load(IDataReader, LoadOption, DataTable[])	Fills a DataSet with values from a data source using the supplied IDataReader, using an array of DataTable instances to supply the schema and namespace information.
Load(IDataReader, LoadOption, String[])	Fills a DataSet with values from a data source using the supplied IDataReader, using an array of strings to supply the names for the tables within the DataSet.
Merge()	Merges the data with data from another DataSet. This method has different overloaded forms.
ReadXML()	Reads an XML schema and data into the DataSet. This method has different overloaded forms.
ReadXMLSchema(0)	Reads an XML schema into the DataSet. This method has different overloaded forms.

RejectChanges	Rolls back all changes made since the last call to AcceptChanges.
WriteXML()	Writes an XML schema and data from the DataSet. This method has different overloaded forms.
WriteXMLSchema()	Writes the structure of the DataSet as an XML schema. This method has different overloaded forms.

The DataTable Class

The DataTable class represents the tables in the database. It has the following important properties; most of these properties are read only properties except the PrimaryKey property:

Properties	**Description**
ChildRelations	Returns the collection of child relationship.
Columns	Returns the Columns collection.
Constraints	Returns the Constraints collection.
DataSet	Returns the parent DataSet.
DefaultView	Returns a view of the table.
ParentRelations	Returns the ParentRelations collection.
PrimaryKey	Gets or sets an array of columns as the primary key for the table.
Rows	Returns the Rows collection.

The following table shows some important methods of the DataTable class:

Methods	**Description**
AcceptChanges	Commits all changes since the last AcceptChanges.
Clear	Clears all data from the table.
GetChanges	Returns a copy of the DataTable with all changes made since the AcceptChanges method was called.
GetErrors	Returns an array of rows with errors.
ImportRows	Copies a new row into the table.
LoadDataRow	Finds and updates a specific row, or creates a new one, if not found any.
Merge	Merges the table with another DataTable.

NewRow	Creates a new DataRow.
RejectChanges	Rolls back all changes made since the last call to AcceptChanges.
Reset	Resets the table to its original state.
Select	Returns an array of DataRow objects.

The DataRow Class

The DataRow object represents a row in a table. It has the following important properties:

Properties	Description
HasErrors	Indicates if there are any errors.
Items	Gets or sets the data stored in a specific column.
ItemArrays	Gets or sets all the values for the row.
Table	Returns the parent table.

The following table shows some important methods of the DataRow class:

Methods	Description
AcceptChanges	Accepts all changes made since this method was called.
BeginEdit	Begins edit operation.
CancelEdit	Cancels edit operation.
Delete	Deletes the DataRow.
EndEdit	Ends the edit operation.
GetChildRows	Gets the child rows of this row.
GetParentRow	Gets the parent row.
GetParentRows	Gets parent rows of DataRow object.
RejectChanges	Rolls back all changes made since the last call to AcceptChanges.

The DataAdapter Object

The DataAdapter object acts as a mediator between the DataSet object and the database. This helps the Dataset to contain data from multiple databases or other data source.

The DataReader Object

The DataReader object is an alternative to the DataSet and DataAdapter combination. This object provides a connection oriented access to the data records in the database. These objects are suitable for read-only access, such as populating a list and then breaking the connection.

DbCommand and DbConnection Objects

The DbConnection object represents a connection to the data source. The connection could be shared among different command objects.

The DbCommand object represents the command or a stored procedure sent to the database from retrieving or manipulating data.

Example

So far, we have used tables and databases already existing in our computer. In this example, we will create a table, add column, rows and data into it and display the table using a GridView object.

The source file code is as given:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="createdatabase._Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>
    <body>
    <form id="form1" runat="server">
          <div>
        <asp:GridView ID="GridView1" runat="server">
        </asp:GridView>
      </div>
          </form>
  </body>
  </html>
```

The code behind file is as given:

```
namespace createdatabase
{
  public partial class _Default : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
```

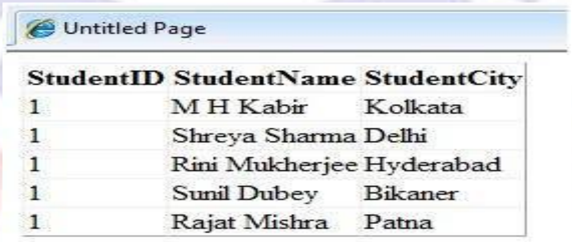
```
    if (!IsPostBack)
    {
      DataSet ds = CreateDataSet();
      GridView1.DataSource = ds.Tables["Student"];
      GridView1.DataBind();
    }
  }
      private DataSet CreateDataSet()
  {
    //creating a DataSet object for tables
    DataSet dataset = new DataSet();
    // creating the student table
    DataTable Students = CreateStudentTable();
    dataset.Tables.Add(Students);
    return dataset;
  }
      private DataTable CreateStudentTable()
  {
    DataTable Students = new DataTable("Student");
    // adding columns
    AddNewColumn(Students, "System.Int32", "StudentID");
    AddNewColumn(Students, "System.String", "StudentName");
    AddNewColumn(Students, "System.String", "StudentCity");
    // adding rows
    AddNewRow(Students, 1, "M H Kabir", "Kolkata");
    AddNewRow(Students, 1, "Shreya Sharma", "Delhi");
    AddNewRow(Students, 1, "Rini Mukherjee", "Hyderabad");
    AddNewRow(Students, 1, "Sunil Dubey", "Bikaner");
    AddNewRow(Students, 1, "Rajat Mishra", "Patna");

    return Students;
  }
  private void AddNewColumn(DataTable table, string columnType, string  columnName)
  {
    DataColumn column = table.Columns.Add(columnName,  Type.GetType(columnType));
  }
```

```
      //adding data into the table
      private void AddNewRow(DataTable table, int id, string name, string city)
      {
        DataRow newrow = table.NewRow();
        newrow["StudentID"] = id;
        newrow["StudentName"] = name;
        newrow["StudentCity"] = city;
        table.Rows.Add(newrow);
      }
   }
}
```

When you execute the program, observe the following:

- The application first creates a data set and binds it with the grid view control using the DataBind() method of the GridView control.
- The Createdataset() method is a user defined function, which creates a new DataSet object and then calls another user defined method CreateStudentTable() to create the table and add it to the Tables collection of the data set.
- The CreateStudentTable() method calls the user defined methods AddNewColumn() and AddNewRow() to create the columns and rows of the table as well as to add data to the rows.

When the page is executed, it returns the rows of the table as shown:



StudentID	StudentName	StudentCity
1	M H Kabir	Kolkata
1	Shreya Sharma	Delhi
1	Rini Mukherjee	Hyderabad
1	Sunil Dubey	Bikaner
1	Rajat Mishra	Patna

ASP.NET - File Uploading

ASP.NET has two controls that allow users to upload files to the web server. Once the server receives the posted file data, the application can save it, check it, or ignore it. The following controls allow the file uploading:

- **HtmlInputFile** - an HTML server control
- **FileUpload** - and ASP.NET web control

Both controls allow file uploading, but the FileUpload control automatically sets the encoding of

the form, whereas the HtmlInputFile does not do so.

In this tutorial, we use the FileUpload control. The FileUpload control allows the user to browse for and select the file to be uploaded, providing a browse button and a text box for entering the filename.

Once, the user has entered the filename in the text box by typing the name or browsing, the SaveAs method of the FileUpload control can be called to save the file to the disk.

The basic syntax of FileUpload is:

```
<asp:FileUpload ID= "Uploader" runat = "server" />
```

The FileUpload class is derived from the WebControl class, and inherits all its members. Apart from those, the FileUpload class has the following read-only properties:

Properties	**Description**
FileBytes	Returns an array of the bytes in a file to be uploaded.
FileContent	Returns the stream object pointing to the file to be uploaded.
FileName	Returns the name of the file to be uploaded.
HasFile	Specifies whether the control has a file to upload.
PostedFile	Returns a reference to the uploaded file.

The posted file is encapsulated in an object of type HttpPostedFile, which could be accessed through the PostedFile property of the FileUpload class.

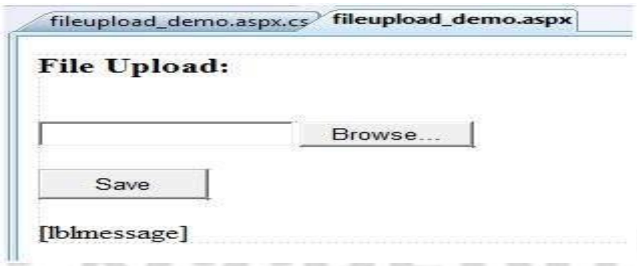
The HttpPostedFile class has the following frequently used properties:

Properties	**Description**
ContentLength	Returns the size of the uploaded file in bytes.
ContentType	Returns the MIME type of the uploaded file.
FileName	Returns the full filename.
InputStream	Returns a stream object pointing to the uploaded file.

Example

The following example demonstrates the FileUpload control and its properties. The form has a FileUpload control along with a save button and a label control for displaying the file name, file type, and file length.

In the design view, the form looks as follows:



The content file code is as given:

```
<body>
  <form id="form1" runat="server">
      <div>
      <h3> File Upload:</h3>
      <br />
      <asp:FileUpload ID="FileUpload1" runat="server" />
      <br /><br />
      <asp:Button  ID="btnsave"  runat="server"  onclick="btnsave_Click"  Text="Save"
style="width:85px" />
      <br /><br />
      <asp:Label ID="lblmessage" runat="server" />
    </div>
      </form>
</body>
```

The code behind the save button is as given:

```
protected void btnsave_Click(object sender, EventArgs e)
{
  StringBuilder sb = new StringBuilder();
    if (FileUpload1.HasFile)
  {
    try
    {
      sb.AppendFormat(" Uploading file: {0}", FileUpload1.FileName);
            //saving the file
      FileUpload1.SaveAs("<c:\\SaveDirectory>" + FileUpload1.FileName);
          //Showing the file information
      sb.AppendFormat("<br/> Save As: {0}",  FileUpload1.PostedFile.FileName);
      sb.AppendFormat("<br/> File type: {0}",    FileUpload1.PostedFile.ContentType);
      sb.AppendFormat("<br/> File length: {0}",  FileUpload1.PostedFile.ContentLength);
```

```
        sb.AppendFormat("<br/> File name: {0}",  FileUpload1.PostedFile.FileName);
            }catch (Exception ex)
      {
        sb.Append("<br/> Error <br/>");
        sb.AppendFormat("Unable to save file <br/> {0}", ex.Message);
      }
   }
   else
   {
     lblmessage.Text = sb.ToString();
   }
}
```

Note the following:

- The StringBuilder class is derived from System.IO namespace, so it needs to be included.
- The try and catch blocks are used for catching errors, and display the error message.

ASP.NET - Ad Rotator

The AdRotator control randomly selects banner graphics from a list, which is specified in an external XML schedule file. This external XML schedule file is called the advertisement file.

The AdRotator control allows you to specify the advertisement file and the type of window that the link should follow in the AdvertisementFile and the Target property respectively.

The basic syntax of adding an AdRotator is as follows:

```
<asp:AdRotator  runat = "server" AdvertisementFile = "adfile.xml"  Target =  "_blank" />
```

Before going into the details of the AdRotator control and its properties, let us look into the construction of the advertisement file.

The Advertisement File

The advertisement file is an XML file, which contains the information about the advertisements to be displayed.

Extensible Markup Language (XML) is a W3C standard for text document markup. It is a text-based markup language that enables you to store data in a structured format by using meaningful tags. The term 'extensible' implies that you can extend your ability to describe a document by defining meaningful tags for the application.

XML is not a language in itself, like HTML, but a set of rules for creating new markup languages. It is a meta-markup language. It allows developers to create custom tag sets for special uses. It structures, stores, and transports the information.

Following is an example of XML file:

```
<BOOK>
  <NAME> Learn XML </NAME>
  <AUTHOR> Samuel Peterson </AUTHOR>
  <PUBLISHER> NSS Publications </PUBLISHER>
  <PRICE> $30.00</PRICE>
</BOOK>
```

Like all XML files, the advertisement file needs to be a structured text file with well-defined tags delineating the data. There are the following standard XML elements that are commonly used in the advertisement file:

Element	**Description**
Advertisements	Encloses the advertisement file.
Ad	Delineates separate ad.
ImageUrl	The path of image that will be displayed.
NavigateUrl	The link that will be followed when the user clicks the ad.
AlternateText	The text that will be displayed instead of the picture if it cannot be displayed.
Keyword	Keyword identifying a group of advertisements. This is used for filtering.
Impressions	The number indicating how often an advertisement will appear.
Height	Height of the image to be displayed.
Width	Width of the image to be displayed.

Apart from these tags, customs tags with custom attributes could also be included. The following code illustrates an advertisement file ads.xml:

```
<Advertisements>
  <Ad>
    <ImageUrl>rose1.jpg</ImageUrl>
    <NavigateUrl>http://www.1800flowers.com</NavigateUrl>
    <AlternateText>
      Order flowers, roses, gifts and more
    </AlternateText>
    <Impressions>20</Impressions>
    <Keyword>flowers</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>rose2.jpg</ImageUrl>
    <NavigateUrl>http://www.babybouquets.com.au</NavigateUrl>
    <AlternateText>Order roses and flowers</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>gifts</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>rose3.jpg</ImageUrl>
    <NavigateUrl>http://www.flowers2moscow.com</NavigateUrl>
    <AlternateText>Send flowers to Russia</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>russia</Keyword>
  </Ad>
  <Ad>
    <ImageUrl>rose4.jpg</ImageUrl>
    <NavigateUrl>http://www.edibleblooms.com</NavigateUrl>
    <AlternateText>Edible Blooms</AlternateText>
    <Impressions>20</Impressions>
    <Keyword>gifts</Keyword>
  </Ad>
</Advertisements>
```

Properties and Events of the AdRotator Class

The AdRotator class is derived from the WebControl class and inherits its properties. Apart from those, the AdRotator class has the following properties:

Properties	**Description**
AdvertisementFile	The path to the advertisement file.
AlternateTextFeild	The element name of the field where alternate text is provided. The default value is AlternateText.
DataMember	The name of the specific list of data to be bound when advertisement file is not used.
DataSource	Control from where it would retrieve data.

DataSourceID	Id of the control from where it would retrieve data.
Font	Specifies the font properties associated with the advertisement banner control.
ImageUrlField	The element name of the field where the URL for the image is provided. The default value is ImageUrl.
KeywordFilter	For displaying the keyword based ads only.
NavigateUrlField	The element name of the field where the URL to navigate to is provided. The default value is NavigateUrl.
Target	The browser window or frame that displays the content of the page linked.
UniqueID	Obtains the unique, hierarchically qualified identifier for the AdRotator control.

Following are the important events of the AdRotator class:

Events	Description
AdCreated	It is raised once per round trip to the server after creation of the control, but before the page is rendered
DataBinding	Occurs when the server control binds to a data source.
DataBound	Occurs after the server control binds to a data source.
Disposed	Occurs when a server control is released from memory, which is the last stage of the server control lifecycle when an ASP.NET page is requested
Init	Occurs when the server control is initialized, which is the first step in its lifecycle.
Load	Occurs when the server control is loaded into the Page object.
PreRender	Occurs after the Control object is loaded but prior to rendering.
Unload	Occurs when the server control is unloaded from memory.

Working with AdRotator Control

Create a new web page and place an AdRotator control on it.

```
<form id="form1" runat="server">
   <div>
      <asp:AdRotator ID="AdRotator1" runat="server" AdvertisementFile ="~/ads.xml"
onadcreated="AdRotator1_AdCreated" />
   </div>
</form>
```

The ads.xml file and the image files should be located in the root directory of the web site.

Try to execute the above application and observe that each time the page is reloaded, the ad is changed.

ASP.NET – Calendars

The calendar control is a functionally rich web control, which provides the following capabilities:

- Displaying one month at a time
- Selecting a day, a week or a month
- Selecting a range of days
- Moving from month to month
- Controlling the display of the days programmatically

The basic syntax of a calendar control is:

```
<asp:Calender ID = "Calendar1" runat = "server">
</asp:Calender>
```

Properties and Events of the Calendar Control

The calendar control has many properties and events, using which you can customize the actions and display of the control. The following table provides some important properties of the Calendar control:

Properties	**Description**
Caption	Gets or sets the caption for the calendar control.
CaptionAlign	Gets or sets the alignment for the caption.
CellPadding	Gets or sets the number of spaces between the data and the cell border.
CellSpacing	Gets or sets the space between cells.
DayHeaderStyle	Gets the style properties for the section that displays the day of the week.
DayNameFormat	Gets or sets format of days of the week.

DayStyle	Gets the style properties for the days in the displayed month.
FirstDayOfWeek	Gets or sets the day of week to display in the first column.
NextMonthText	Gets or sets the text for next month navigation control. The default value is >.
NextPrevFormat	Gets or sets the format of the next and previous month navigation control.
OtherMonthDayStyle	Gets the style properties for the days on the Calendar control that are not in the displayed month.
PrevMonthText	Gets or sets the text for previous month navigation control. The default value is <.
SelectedDate	Gets or sets the selected date.
SelectedDates	Gets a collection of DateTime objects representing the selected dates.
SelectedDayStyle	Gets the style properties for the selected dates.
SelectionMode	Gets or sets the selection mode that specifies whether the user can select a single day, a week or an entire month.
SelectMonthText	Gets or sets the text for the month selection element in the selector column.
SelectorStyle	Gets the style properties for the week and month selector column.
SelectWeekText	Gets or sets the text displayed for the week selection element in the selector column.
ShowDayHeader	Gets or sets the value indicating whether the heading for the days of the week is displayed.
ShowGridLines	Gets or sets the value indicating whether the gridlines would be shown.
ShowNextPrevMonth	Gets or sets a value indicating whether next and previous month navigation elements are shown in the title section.
ShowTitle	Gets or sets a value indicating whether the title section is displayed.
TitleFormat	Gets or sets the format for the title section.

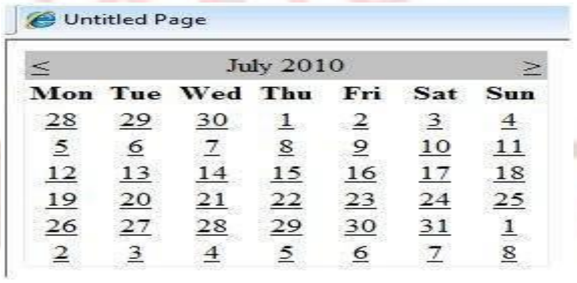
Titlestyle	Get the style properties of the title heading for the Calendar control.
TodayDayStyle	Gets the style properties for today's date on the Calendar control.
TodaysDate	Gets or sets the value for today's date.
UseAccessibleHeader	Gets or sets a value that indicates whether to render the table header <th> HTML element for the day headers instead of the table data <td> HTML element.
VisibleDate	Gets or sets the date that specifies the month to display.
WeekendDayStyle	Gets the style properties for the weekend dates on the Calendar control.

The Calendar control has the following three most important events that allow the developers to program the calendar control. They are:

Events	Description
SelectionChanged	It is raised when a day, a week or an entire month is selected.
DayRender	It is raised when each data cell of the calendar control is rendered.
VisibleMonthChanged	It is raised when user changes a month.

Working with the Calendar Control

Putting a bare-bone calendar control without any code behind file provides a workable calendar to a site, which shows the months and days of the year. It also allows navigation to next and previous months.



<			July 2010			>
Mon	Tue	Wed	Thu	Fri	Sat	Sun
28	29	30	1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	1
2	3	4	5	6	7	8

Calendar controls allow the users to select a single day, a week, or an entire month. This is done by using the SelectionMode property. This property has the following values:

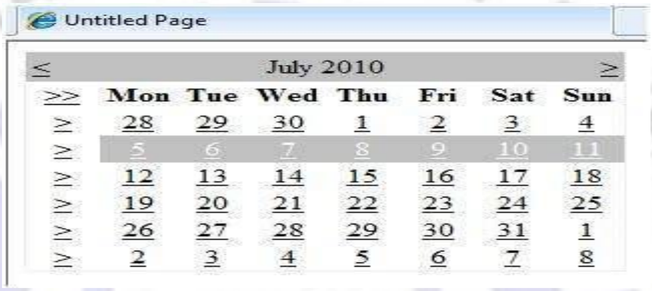
Properties	Description
Day	To select a single day.
DayWeek	To select a single day or an entire week.

DayWeekMonth	To select a single day, a week, or an entire month.
None	Nothing can be selected.

The syntax for selecting days:

```
<asp:Calender ID = "Calendar1" runat = "server" SelectionMode="DayWeekMonth">
</asp:Calender>
```

When the selection mode is set to the value DayWeekMonth, an extra column with the > symbol appears for selecting the week, and a >> symbol appears to the left of the days name for selecting the month.



<			July 2010				>
>>	Mon	Tue	Wed	Thu	Fri	Sat	Sun
>	28	29	30	1	2	3	4
>	5	6	7	8	9	10	11
>	12	13	14	15	16	17	18
>	19	20	21	22	23	24	25
>	26	27	28	29	30	31	1
>	2	3	4	5	6	7	8

Example

The following example demonstrates selecting a date and displays the date in a label:

The content file code is as follows:

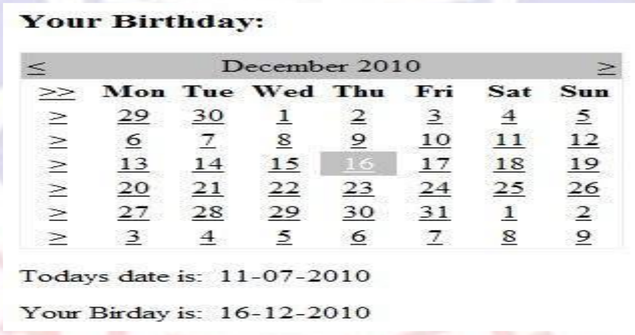
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="calendardemo._Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>
    <body>
    <form id="form1" runat="server">
          <div>
        <h3> Your Birthday:</h3>
        <asp:Calendar ID="Calendar1" runat="server SelectionMode="DayWeekMonth"
onselectionchanged="Calendar1_SelectionChanged">
        </asp:Calendar>
      </div>
```

```
        <p>Todays date is:
           <asp:Label ID="lblday" runat="server"></asp:Label>
        </p>
              <p>Your Birday is:
           <asp:Label ID="lblbday" runat="server"></asp:Label>
        </p>
            </form>
   </body>
</html>
```

The event handler for the event SelectionChanged:

```
protected void Calendar1_SelectionChanged(object sender, EventArgs e)
{
   lblday.Text = Calendar1.TodaysDate.ToShortDateString();
   lblbday.Text = Calendar1.SelectedDate.ToShortDateString();
}
```

When the file is run, it should produce the following output:



```
Your Birthday:

<              December 2010              >
>>   Mon  Tue  Wed  Thu  Fri  Sat  Sun
>    29   30   1    2    3    4    5
>    6    7    8    9    10   11   12
>    13   14   15   16   17   18   19
>    20   21   22   23   24   25   26
>    27   28   29   30   31   1    2
>    3    4    5    6    7    8    9

Todays date is:  11-07-2010

Your Birday is:  16-12-2010
```

ASP.NET - Multi Views

MultiView and View controls allow you to divide the content of a page into different groups, displaying only one group at a time. Each View control manages one group of content and all the View controls are held together in a MultiView control. The MultiView control is responsible for displaying one View control at a time. The View displayed is called the active view.

The syntax of MultiView control is:

```
<asp:MultView ID= "MultiView1" runat= "server">
</asp:MultiView>
```

The syntax of View control is:

```
<asp:View ID= "View1" runat= "server">
```

```
</asp:View>
```

However, the View control cannot exist on its own. It would render error if you try to use it stand-alone. It is always used with a Multiview control as:

```
<asp:MultView ID= "MultiView1" runat= "server">
   <asp:View ID= "View1" runat= "server"> </asp:View>
</asp:MultiView>
```

Properties of View and MultiView Controls

Both View and MultiView controls are derived from Control class and inherit all its properties, methods, and events. The most important property of the View control is Visible property of type Boolean, which sets the visibility of a view.

The MultiView control has the following important properties:

Properties	Description
Views	Collection of View controls within the MultiView.
ActiveViewIndex	A zero based index that denotes the active view. If no view is active, then the index is -1.

The CommandName attribute of the button control associated with the navigation of the MultiView control are associated with some related field of the MultiView control.

For example, if a button control with CommandName value as NextView is associated with the navigation of the multiview, it automatically navigates to the next view when the button is clicked.

The following table shows the default command names of the above properties:

Properties	Description
NextViewCommandName	NextView
PreviousViewCommandName	PrevView
SwitchViewByIDCommandName	SwitchViewByID
SwitchViewByIndexCommandName	SwitchViewByIndex

The important methods of the multiview control are:

Methods	Description
SetActiveview	Sets the active view
GetActiveview	Retrieves the active view

Every time a view is changed, the page is posted back to the server and a number of events are raised. Some important events are:

Events	Description
ActiveViewChanged	Raised when a view is changed
Activate	Raised by the active view
Deactivate	Raised by the inactive view

Apart from the above mentioned properties, methods and events, multiview control inherits the members of the control and object class.

Example

The example page has three views. Each view has two button for navigating through the views.

The content file code is as follows:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="multiviewdemo._Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>
    <body>
    <form id="form1" runat="server">
          <div>
        <h2>MultiView and View Controls</h2>
                <asp:DropDownList ID="DropDownList1" runat="server" onselectedindexchanged="DropDownList1_SelectedIndexChanged">
        </asp:DropDownList>
                <hr />
                <asp:MultiView ID="MultiView1" runat="server" ActiveViewIndex="2" onactiveviewchanged="MultiView1_ActiveViewChanged" >
          <asp:View ID="View1" runat="server">
            <h3>This is view 1</h3>
            <br />
```

```
              <asp:Button CommandName="NextView" ID="btnnext1" runat="server" Text = "Go To Next" />
              <asp:Button CommandArgument="View3" CommandName="SwitchViewByID" ID="btnlast" runat="server" Text  ="Go To Last" />
            </asp:View>

            <asp:View ID="View2" runat="server">
              <h3>This is view 2</h3>
              <asp:Button CommandName="NextView" ID="btnnext2" runat="server" Text = "Go To Next" />
              <asp:Button CommandName="PrevView" ID="btnprevious2" runat="server" Text = "Go To Previous View" />
            </asp:View>
            <asp:View ID="View3" runat="server">
              <h3> This is view 3</h3>
              <br />
              <asp:Calendar ID="Calender1" runat="server"></asp:Calendar>
              <br />
              <asp:Button CommandArgument="0" CommandName="SwitchViewByIndex" ID="btnfirst"   runat="server" Text = "Go To Next" />
              <asp:Button CommandName="PrevView" ID="btnprevious" runat="server" Text = "Go To Previous View" />
            </asp:View>
                  </asp:MultiView>
        </div>
            </form>
  </body>
</html>
```

Observe the following:

The MultiView.ActiveViewIndex determines which view will be shown. This is the only view rendered on the page. The default value for the ActiveViewIndex is -1, when no view is shown. Since the ActiveViewIndex is defined as 2 in the example, it shows the third view, when executed.

ASP.NET - Panel Controls

The Panel control works as a container for other controls on the page. It controls the appearance and visibility of the controls it contains. It also allows generating controls programmatically.

The basic syntax of panel control is as follows:

```
<asp:Panel ID= "Panel1"  runat = "server">
</asp:Panel>
```

The Panel control is derived from the WebControl class. Hence it inherits all the properties, methods and events of the same. It does not have any method or event of its own. However it has the following properties of its own:

Properties	**Description**
BackImageUrl	URL of the background image of the panel.
DefaultButton	Gets or sets the identifier for the default button that is contained in the Panel control.
Direction	Text direction in the panel.
GroupingText	Allows grouping of text as a field.
HorizontalAlign	Horizontal alignment of the content in the panel.
ScrollBars	Specifies visibility and location of scrollbars within the panel.
Wrap	Allows text wrapping.

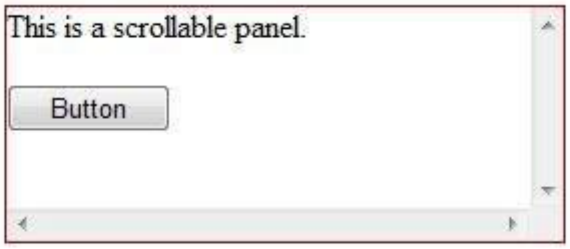
Working with the Panel Control

Let us start with a simple scrollable panel of specific height and width and a border style. The ScrollBars property is set to both the scrollbars, hence both the scrollbars are rendered.

The source file has the following code for the panel tag:

```
<asp:Panel ID="Panel1" runat="server" BorderColor="#990000" BorderStyle="Solid"
   Borderstyle="width:1px" Height="116px" ScrollBars="Both" style="width:278px">
      This is a scrollable panel.
   <br />
   <br />
   <asp:Button ID="btnpanel" runat="server" Text="Button" style="width:82px" />
</asp:Panel>
```

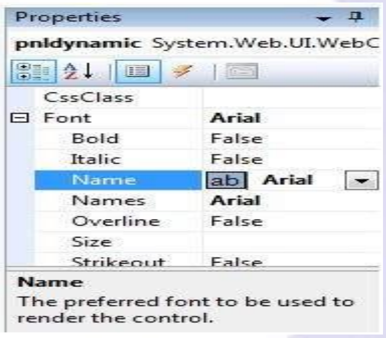
The panel is rendered as follows:



Example

The following example demonstrates dynamic content generation. The user provides the number of label controls and textboxes to be generated on the panel. The controls are generated programmatically.

Change the properties of the panel using the properties window. When you select a control on the design view, the properties window displays the properties of that particular control and allows you to make changes without typing.


Properties
pnldynamic System.Web.UI.WebC

CssClass	
⊟ Font	Arial
Bold	False
Italic	False
Name	Arial
Names	Arial
Overline	False
Size	
Strikeout	False

Name
The preferred font to be used to render the control.

The source file for the example is as follows:

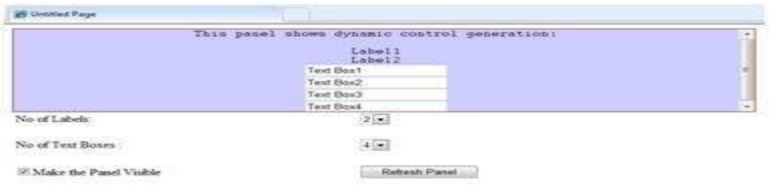
```
<form id="form1" runat="server">
   <div>
      <asp:Panel ID="pnldynamic" runat="server" BorderColor="#990000"
         BorderStyle="Solid"    Borderstyle="width:1px"    Height="150px"       ScrollBars="Auto"
style="width:60%" BackColor="#CCCCFF"  Font-Names="Courier" HorizontalAlign="Center">
            This panel shows dynamic control generation:
         <br />
         <br />
      </asp:Panel>
   </div>
   <table style="width: 51%;">
      <tr>
         <td class="style2">No of Labels:</td>
         <td class="style1">
            <asp:DropDownList ID="ddllabels" runat="server">
               <asp:ListItem>0</asp:ListItem>
               <asp:ListItem>1</asp:ListItem>
               <asp:ListItem>2</asp:ListItem>
               <asp:ListItem>3</asp:ListItem>
               <asp:ListItem>4</asp:ListItem>
            </asp:DropDownList>
         </td>
      </tr>
      <tr>
         <td class="style2"> </td>
         <td class="style1"> </td>
      </tr>
      <tr>
         <td class="style2">No of Text Boxes :</td>
         <td class="style1">
            <asp:DropDownList ID="ddltextbox" runat="server">
               <asp:ListItem>0</asp:ListItem>
               <asp:ListItem Value="1"></asp:ListItem>
               <asp:ListItem>2</asp:ListItem>
               <asp:ListItem>3</asp:ListItem>
               <asp:ListItem Value="4"></asp:ListItem>
            </asp:DropDownList>
         </td>
      </tr>
      <tr>
         <td class="style2"> </td>
         <td class="style1"> </td>
      </tr>
      <tr>
         <td class="style2">
            <asp:CheckBox ID="chkvisible" runat="server"
               Text="Make the Panel Visible" />
         </td>

         <td class="style1">
            <asp:Button ID="btnrefresh" runat="server" Text="Refresh Panel"
```

```
            style="width:129px" />
        </td>
      </tr>
  </table>
</form>
```

The code behind the Page_Load event is responsible for generating the controls dynamically:

```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    //make the panel visible
    pnldynamic.Visible = chkvisible.Checked;
    //generating the lable controls:
    int n = Int32.Parse(ddllabels.SelectedItem.Value);
    for (int i = 1; i <= n; i++)
    {
      Label lbl = new Label();
      lbl.Text = "Label" + (i).ToString();
      pnldynamic.Controls.Add(lbl);
      pnldynamic.Controls.Add(new LiteralControl("<br />"));
    }
        //generating the text box controls:
    int m = Int32.Parse(ddltextbox.SelectedItem.Value);
    for (int i = 1; i <= m; i++)
    {
      TextBox txt = new TextBox();
      txt.Text = "Text Box" + (i).ToString();
      pnldynamic.Controls.Add(txt);
      pnldynamic.Controls.Add(new LiteralControl("<br />"));
    }
  }
}
```

When executed, the panel is rendered as:

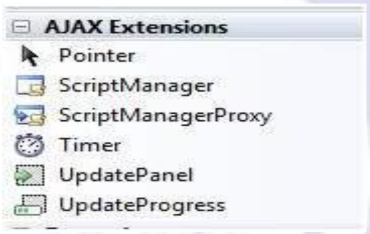


ASP.NET - Ajax Control

AJAX stands for Asynchronous JavaScript and XML. This is a cross platform technology which speeds up response time. The AJAX server controls add script to the page which is executed and processed by the browser.

However like other ASP.NET server controls, these AJAX server controls also can have methods and event handlers associated with them, which are processed on the server side.

The control toolbox in the Visual Studio IDE contains a group of controls called the 'AJAX Extensions'



The ScriptManager Control

The ScriptManager control is the most important control and must be present on the page for other controls to work.

It has the basic syntax:

```
<asp:ScriptManager ID="ScriptManager1" runat="server">
</asp:ScriptManager>
```

If you create an 'Ajax Enabled site' or add an 'AJAX Web Form' from the 'Add Item' dialog box, the web form automatically contains the script manager control. The ScriptManager control takes care of the client-side script for all the server side controls.

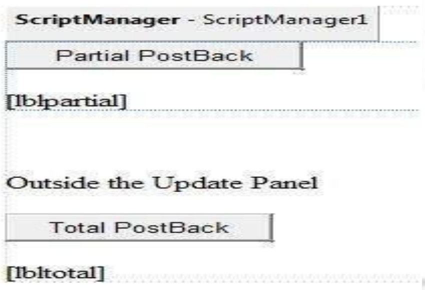
The UpdatePanel Control

The UpdatePanel control is a container control and derives from the Control class. It acts as a container for the child controls within it and does not have its own interface. When a control inside it triggers a post back, the UpdatePanel intervenes to initiate the post asynchronously and update just that portion of the page. For example, if a button control is inside the update panel and it is clicked, only the controls within the update panel will be affected, the controls on the other parts of the page will not be affected. This is called the partial post back or the asynchronous post back.

Example

Add an AJAX web form in your application. It contains the script manager control by default. Insert an update panel. Place a button control along with a label control within the update panel control. Place another set of button and label outside the panel.

The design view looks as follows:



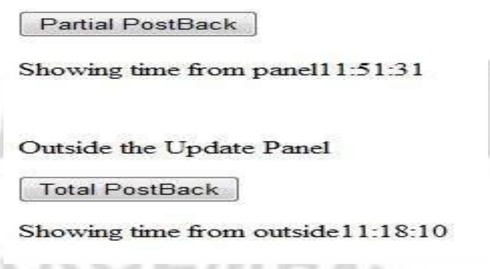
The source file is as follows:

```
<form id="form1" runat="server">
   <div>
      <asp:ScriptManager ID="ScriptManager1" runat="server" />
   </div>
    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
      <ContentTemplate>
         <asp:Button ID="btnpartial" runat="server" onclick="btnpartial_Click" Text="Partial
PostBack"/>
         <br />
         <br />
         <asp:Label ID="lblpartial" runat="server"></asp:Label>
      </ContentTemplate>
   </asp:UpdatePanel>
      <p> </p>
   <p>Outside the Update Panel</p>
   <p>
      <asp:Button ID="btntotal" runat="server" onclick="btntotal_Click" Text="Total PostBack" />
   </p>
      <asp:Label ID="lbltotal" runat="server"></asp:Label>
</form>
```

Both the button controls have same code for the event handler:

```
string time = DateTime.Now.ToLongTimeString();
lblpartial.Text = "Showing time from panel" + time;
lbltotal.Text = "Showing time from outside" + time;
```

Observe that when the page is executed, if the total post back button is clicked, it updates time in both the labels but if the partial post back button is clicked, it only updates the label within the update panel.



Showing time from panel11:51:31

Outside the Update Panel

Showing time from outside11:18:10

A page can contain multiple update panels with each panel containing other controls like a grid and displaying different part of data.
When a total post back occurs, the update panel content is updated by default. This default mode could be changed by changing the UpdateMode property of the control. Let us look at other properties of the update panel.

Properties of the UpdatePanel Control

The following table shows the properties of the update panel control:

Properties	**Description**
ChildrenAsTriggers	This property indicates whether the post backs are coming from the child controls, which cause the update panel to refresh.
ContentTemplate	It is the content template and defines what appears in the update panel when it is rendered.
ContentTemplateContainer	Retrieves the dynamically created template container object and used for adding child controls programmatically.
IsInPartialRendering	Indicates whether the panel is being updated as part of the partial post back.
RenderMode	Shows the render modes. The available modes are Block and Inline.
UpdateMode	Gets or sets the rendering mode by determining some conditions.
Triggers	Defines the collection trigger objects each corresponding to an event causing the panel to refresh automatically.

Methods of the UpdatePanel Control

The following table shows the methods of the update panel control:

Methods	**Description**
CreateContentTemplateContainer	Creates a Control object that acts as a container for child controls that define the UpdatePanel control's content.
CreateControlCollection	Returns the collection of all controls that are contained in the UpdatePanel control.
Initialize	Initializes the UpdatePanel control trigger collection if partial-page rendering is enabled.
Update	Causes an update of the content of an UpdatePanel control.

The behavior of the update panel depends upon the values of the UpdateMode property and ChildrenAsTriggers property.

UpdateMode	**ChildrenAsTriggers**	**Effect**
Always	False	Illegal parameters.
Always	True	UpdatePanel refreshes if whole page refreshes or a child control on it posts back.
Conditional	False	UpdatePanel refreshes if whole page refreshes or a triggering control outside it initiates a refresh.
Conditional	True	UpdatePanel refreshes if whole page refreshes or a child control on it posts back or a triggering control outside it initiates a refresh.

The UpdateProgress Control

The UpdateProgress control provides a sort of feedback on the browser while one or more update panel controls are being updated. For example, while a user logs in or waits for server response while performing some database oriented job.

It provides a visual acknowledgement like "Loading page...", indicating the work is in progress.

The syntax for the UpdateProgress control is:

```
<asp:UpdateProgress ID="UpdateProgress1" runat="server" DynamicLayout="true"
AssociatedUpdatePanelID="UpdatePanel1" >

   <ProgressTemplate>
```

```
    Loading...
  </ProgressTemplate>
  </asp:UpdateProgress>
```

The above snippet shows a simple message within the ProgressTemplate tag. However, it could be an image or other relevant controls. The UpdateProgress control displays for every asynchronous postback unless it is assigned to a single update panel using the AssociatedUpdatePanelID property.

Properties of the UpdateProgress Control

The following table shows the properties of the update progress control:

Properties	**Description**
AssociatedUpdatePanelID	Gets and sets the ID of the update panel with which this control is associated.
Attributes	Gets or sets the cascading style sheet (CSS) attributes of the UpdateProgress control.
DisplayAfter	Gets and sets the time in milliseconds after which the progress template is displayed. The default is 500.
DynamicLayout	Indicates whether the progress template is dynamically rendered.
ProgressTemplate	Indicates the template displayed during an asynchronous post back which takes more time than the DisplayAfter time.

Methods of the UpdateProgress Control

The following table shows the methods of the update progress control:

Methods	**Description**
GetScriptDescriptors	Returns a list of components, behaviors, and client controls that are required for the UpdateProgress control's client functionality.
GetScriptReferences	Returns a list of client script library dependencies for the UpdateProgress control.

The Timer Control

The timer control is used to initiate the post back automatically. This could be done in two ways:

(1) Setting the Triggers property of the UpdatePanel control:

```
<Triggers>
  <asp:AsyncPostBackTrigger ControlID="btnpanel2" EventName="Click" />
</Triggers>
```

(2) Placing a timer control directly inside the UpdatePanel to act as a child control trigger. A single timer can be the trigger for multiple UpdatePanels.

```
<asp:UpdatePanel ID="UpdatePanel1" runat="server" UpdateMode="Always">
  <ContentTemplate>
    <asp:Timer ID="Timer1" runat="server" Interval="1000">
      </asp:Timer>
          <asp:Label ID="Label1" runat="server" Height="101px" style="width:304px" >
      </asp:Label>
  </ContentTemplate>
  </asp:UpdatePanel>
```

ASP.NET - Data Sources

A data source control interacts with the data-bound controls and hides the complex data binding processes. These are the tools that provide data to the data bound controls and support execution of operations like insertions, deletions, sorting, and updates.

Each data source control wraps a particular data provider-relational databases, XML documents, or custom classes and helps in:

- Managing connection
- Selecting data
- Managing presentation aspects like paging, caching, etc.
- Manipulating data

There are many data source controls available in ASP.NET for accessing data from SQL Server, from ODBC or OLE DB servers, from XML files, and from business objects.

Based on type of data, these controls could be divided into two categories:

- Hierarchical data source controls
- Table-based data source controls

The data source controls used for hierarchical data are:

- **XMLDataSource** - It allows binding to XML files and strings with or without schema information.
- **SiteMapDataSource** - It allows binding to a provider that supplies site map information.

The data source controls used for tabular data are:

Data source controls	Description
SqlDataSource	It represents a connection to an ADO.NET data provider that returns SQL data, including data sources accessible via OLEDB and ODBC.
ObjectDataSource	It allows binding to a custom .Net business object that returns data.
LinqdataSource	It allows binding to the results of a Linq-to-SQL query (supported by ASP.NET 3.5 only).
AccessDataSource	It represents connection to a Microsoft Access database.

Data Source Views

Data source views are objects of the DataSourceView class. Which represent a customized view of data for different data operations such as sorting, filtering, etc. The DataSourceView class serves as the base class for all data source view classes, which define the capabilities of data source controls.

The following table provides the properties of the DataSourceView class:

Properties	Description
CanDelete	Indicates whether deletion is allowed on the underlying data source.
CanInsert	Indicates whether insertion is allowed on the underlying data source.
CanPage	Indicates whether paging is allowed on the underlying data source.
CanRetrieveTotalRowCount	Indicates whether total row count information is available.
CanSort	Indicates whether the data could be sorted.
CanUpdate	Indicates whether updates are allowed on the underlying data source.
Events	Gets a list of event-handler delegates for the data source view.
Name	Name of the view.

The following table provides the methods of the DataSourceView class:

Methods	Description
CanExecute	Determines whether the specified command can be executed.

ExecuteCommand	Executes the specific command.
ExecuteDelete	Performs a delete operation on the list of data that the DataSourceView object represents.
ExecuteInsert	Performs an insert operation on the list of data that the DataSourceView object represents.
ExecuteSelect	Gets a list of data from the underlying data storage.
ExecuteUpdate	Performs an update operation on the list of data that the DataSourceView object represents.
Delete	Performs a delete operation on the data associated with the view.
Insert	Performs an insert operation on the data associated with the view.
Select	Returns the queried data.
Update	Performs an update operation on the data associated with the view.
OnDataSourceViewChanged	Raises the DataSourceViewChanged event.
RaiseUnsupportedCapabilitiesError	Called by the RaiseUnsupportedCapabilitiesError method to compare the capabilities requested for an ExecuteSelect operation against those that the view supports.

The SqlDataSource Control

The SqlDataSource control represents a connection to a relational database such as SQL Server or Oracle database, or data accessible through OLEDB or Open Database Connectivity (ODBC). Connection to data is made through two important properties ConnectionString and ProviderName.

The following code snippet provides the basic syntax of the control:

```
<asp:SqlDataSource runat="server" ID="MySqlSource"
   ProviderName='<%$ ConnectionStrings:LocalNWind.ProviderName  %>'
   ConnectionString='<%$ ConnectionStrings:LocalNWind %>'
   SelectionCommand= "SELECT * FROM EMPLOYEES" />
<asp:GridView ID="GridView1" runat="server" DataSourceID="MySqlSource" />
```

Configuring various data operations on the underlying data depends upon the various properties (property groups) of the data source control.

The following table provides the related sets of properties of the SqlDataSource control, which provides the programming interface of the control:

Property Group	Description
DeleteCommand, DeleteParameters, DeleteCommandType	Gets or sets the SQL statement, parameters, and type for deleting rows in the underlying data.
FilterExpression, FilterParameters	Gets or sets the data filtering string and parameters.
InsertCommand, InsertParameters, InsertCommandType	Gets or sets the SQL statement, parameters, and type for inserting rows in the underlying database.
SelectCommand, SelectParameters, SelectCommandType	Gets or sets the SQL statement, parameters, and type for retrieving rows from the underlying database.
SortParameterName	Gets or sets the name of an input parameter that the command's stored procedure will use to sort data.
UpdateCommand, UpdateParameters, UpdateCommandType	Gets or sets the SQL statement, parameters, and type for updating rows in the underlying data store.

The following code snippet shows a data source control enabled for data manipulation:

```
<asp:SqlDataSource runat="server" ID= "MySqlSource"
   ProviderName='<%$ ConnectionStrings:LocalNWind.ProviderName  %>'
   ConnectionString=' <%$ ConnectionStrings:LocalNWind %>'
   SelectCommand= "SELECT * FROM EMPLOYEES"
   UpdateCommand= "UPDATE EMPLOYEES SET LASTNAME=@lame"
   DeleteCommand= "DELETE FROM EMPLOYEES WHERE EMPLOYEEID=@eid"
   FilterExpression= "EMPLOYEEID > 10">
   .....
   .....
</asp:SqlDataSource>
```

The ObjectDataSource Control

The ObjectDataSource Control enables user-defined classes to associate the output of their methods to data bound controls. The programming interface of this class is almost same as the SqlDataSource control.

Following are two important aspects of binding business objects:

- The bindable class should have a default constructor, it should be stateless, and have methods that can be mapped to select, update, insert, and delete semantics.
- The object must update one item at a time, batch operations are not supported.

Let us go directly to an example to work with this control. The student class is the class to be used with an object data source. This class has three properties: a student id, name, and city. It has a default constructor and a GetStudents method for retrieving data.

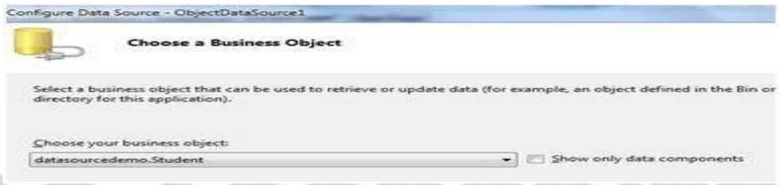
The student class:

```
public class Student
{
  public int StudentID { get; set; }
  public string Name { get; set; }
  public string City { get; set; }
    public Student()
  { }

  public DataSet GetStudents()
  {
    DataSet ds = new DataSet();
    DataTable dt = new DataTable("Students");
    dt.Columns.Add("StudentID", typeof(System.Int32));
    dt.Columns.Add("StudentName", typeof(System.String));
    dt.Columns.Add("StudentCity", typeof(System.String));
    dt.Rows.Add(new object[] { 1, "M. H. Kabir", "Calcutta" });
    dt.Rows.Add(new object[] { 2, "Ayan J. Sarkar", "Calcutta" });
    ds.Tables.Add(dt);
        return ds;
  }
}
```

Take the following steps to bind the object with an object data source and retrieve data:

- Create a new web site.
- Add a class (Students.cs) to it by right clicking the project from the Solution Explorer, adding a class template, and placing the above code in it.
- Build the solution so that the application can use the reference to the class.
- Place an object data source control in the web form.

- Configure the data source by selecting the object.



Configure Data Source - ObjectDataSource1

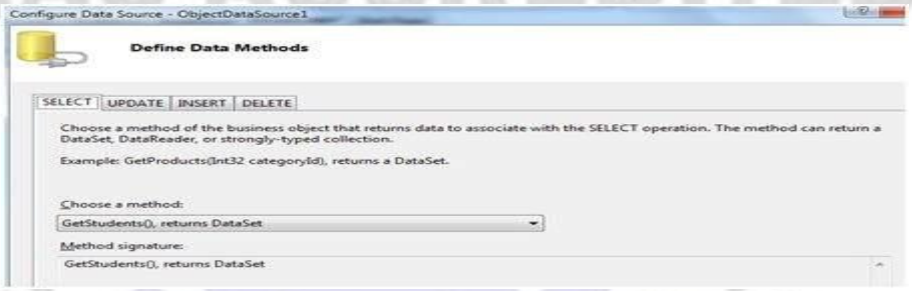
Choose a Business Object

Select a business object that can be used to retrieve or update data (for example, an object defined in the Bin or directory for this application).

Choose your business object: datasourcedemo.Student

Show only data components

- Select a data method(s) for different operations on data. In this example, there is only one method.



Configure Data Source - ObjectDataSource1

Define Data Methods

SELECT | UPDATE | INSERT | DELETE

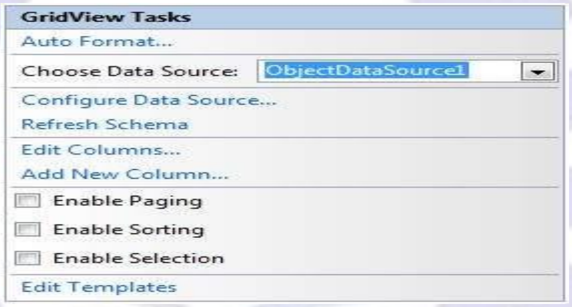
Choose a method of the business object that returns data to associate with the SELECT operation. The method can return a DataSet, DataReader, or strongly-typed collection.

Example: GetProducts(Int32 categoryId), returns a DataSet.

Choose a method: GetStudents(), returns DataSet

Method signature: GetStudents(), returns DataSet

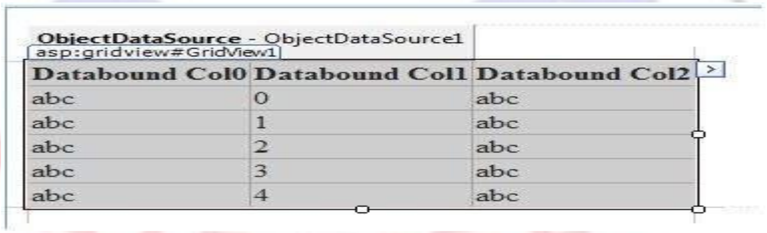
- Place a data bound control such as grid view on the page and select the object data source as its underlying data source.



GridView Tasks

Choose Data Source: ObjectDataSource1

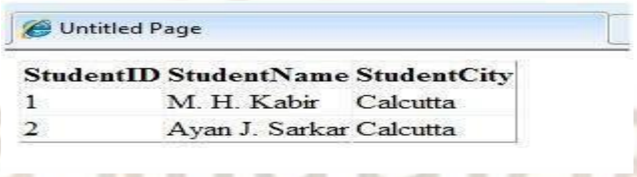
- At this stage, the design view should look like the following:



ObjectDataSource - ObjectDataSource1
asp:gridview#GridView1

Databound Col0	Databound Col1	Databound Col2
abc	0	abc
abc	1	abc
abc	2	abc
abc	3	abc
abc	4	abc

- Run the project, it retrieves the hard coded tuples from the students class.



Untitled Page

StudentID	StudentName	StudentCity
1	M. H. Kabir	Calcutta
2	Ayan J. Sarkar	Calcutta

The AccessDataSource Control

The AccessDataSource control represents a connection to an Access database. It is based on the SqlDataSource control and provides simpler programming interface. The following code snippet provides the basic syntax for the data source:

```
<asp:AccessDataSource ID="AccessDataSource1 runat="server"
  DataFile="~/App_Data/ASPDotNetStepByStep.mdb" SelectCommand="SELECT * FROM
[DotNetReferences]">
```

```
</asp:AccessDataSource>
```

The AccessDataSource control opens the database in read-only mode. However, it can also be used for performing insert, update, or delete operations. This is done using the ADO.NET commands and parameter collection.

Updates are problematic for Access databases from within an ASP.NET application because an Access database is a plain file and the default account of the ASP.NET application might not have the permission to write to the database file.

ASP.NET - Data Binding

Every ASP.NET web form control inherits the DataBind method from its parent Control class, which gives it an inherent capability to bind data to at least one of its properties. This is known as **simple data binding** or **inline data binding**.

Simple data binding involves attaching any collection (item collection) which implements the IEnumerable interface, or the DataSet and DataTable classes to the DataSource property of the control. On the other hand, some controls can bind records, lists, or columns of data into their structure through a DataSource control. These controls derive from the BaseDataBoundControl class. This is called **declarative data binding**.

The data source controls help the data-bound controls implement functionalities such as, sorting, paging, and editing data collections.

The BaseDataBoundControl is an abstract class, which is inherited by two more abstract classes:

- DataBoundControl
- HierarchicalDataBoundControl

The abstract class DataBoundControl is again inherited by two more abstract classes:

- ListControl
- CompositeDataBoundControl

The controls capable of simple data binding are derived from the ListControl abstract class and these controls are:

- BulletedList
- CheckBoxList
- DropDownList
- ListBox
- RadioButtonList

The controls capable of declarative data binding (a more complex data binding) are derived from the abstract class CompositeDataBoundControl. These controls are:

- DetailsView
- FormView
- GridView

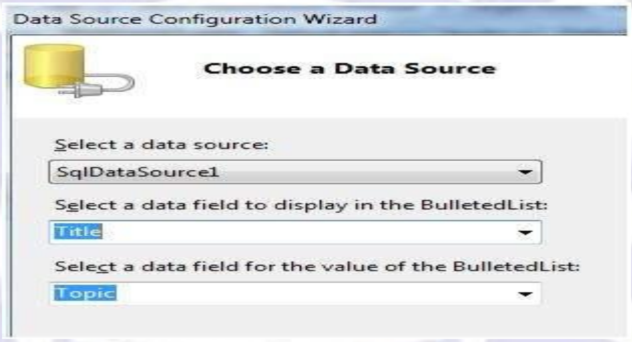
- RecordList

Simple Data Binding

Simple data binding involves the read-only selection lists. These controls can bind to an array list or fields from a database. Selection lists takes two values from the database or the data source; one value is displayed by the list and the other is considered as the value corresponding to the display. Let us take up a small example to understand the concept. Create a web site with a bulleted list and a SqlDataSource control on it. Configure the data source control to retrieve two values from your database (we use the same DotNetReferences table as in the previous chapter).

Choosing a data source for the bulleted list control involves:

- Selecting the data source control
- Selecting a field to display, which is called the data field
- Selecting a field for the value



Data Source Configuration Wizard

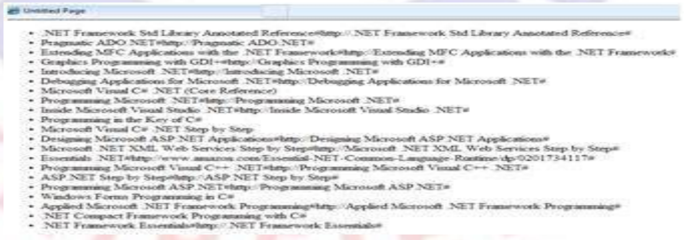
Choose a Data Source

Select a data source: SqlDataSource1

Select a data field to display in the BulletedList: Title

Select a data field for the value of the BulletedList: Topic

When the application is executed, check that the entire title column is bound to the bulleted list and displayed.



Untitled Page

- .NET Framework Std Library Annotated Reference#http://.NET Framework Std Library Annotated Reference#
- Pragmatic ADO.NET#http://Pragmatic ADO.NET#
- Extending MFC Applications with the .NET Framework#http://Extending MFC Applications with the .NET Framework#
- Graphics Programming with GDI+#http://Graphics Programming with GDI+#
- Introducing Microsoft .NET#http://Introducing Microsoft .NET#
- Debugging Applications for Microsoft .NET#http://Debugging Applications for Microsoft .NET#
- Microsoft Visual C# .NET (Core Reference)
- Programming Microsoft .NET#http://Programming Microsoft .NET#
- Inside Microsoft Visual Studio .NET#http://Inside Microsoft Visual Studio .NET#
- Programming in the Key of C#
- Microsoft Visual C# .NET Step by Step
- Designing Microsoft ASP.NET Applications#http://Designing Microsoft ASP.NET Applications#
- Microsoft .NET XML Web Services Step by Step#http://Microsoft .NET XML Web Services Step by Step#
- Essentials .NET#http://www.amazon.com/Essential-NET-Common-Language-Runtime/dp/0201734117#
- Programming Microsoft Visual C++ .NET#http://Programming Microsoft Visual C++ .NET#
- ASP.NET Step by Step#http://ASP.NET Step by Step#
- Programming Microsoft ASP.NET#http://Programming Microsoft ASP.NET#
- Windows Forms Programming in C#
- Applied Microsoft .NET Framework Programming#http://Applied Microsoft .NET Framework Programming#
- .NET Compact Framework Programming with C#
- .NET Framework Essentials#http://.NET Framework Essentials#

Declarative Data Binding

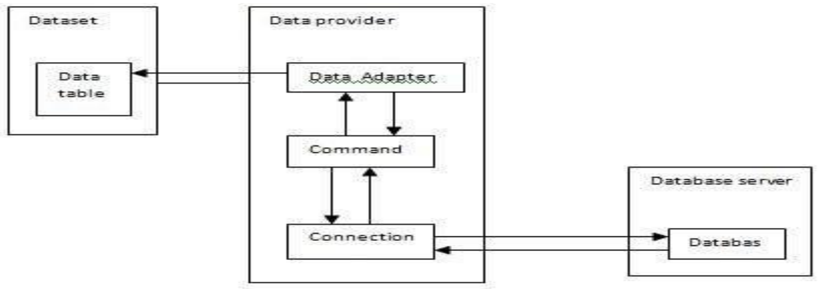
We have already used declarative data binding in the previous tutorial using GridView control. The other composite data bound controls capable of displaying and manipulating data in a tabular manner are the DetailsView, FormView, and RecordList control.

In the next tutorial, we will look into the technology for handling database, i.e, ADO.NET.

However, the data binding involves the following objects:

- A dataset that stores the data retrieved from the database.
- The data provider, which retrieves data from the database by using a command over a connection.
- The data adapter that issues the select statement stored in the command object; it is also capable of update the data in a database by issuing Insert, Delete, and Update statements.

Relation between the data binding objects:



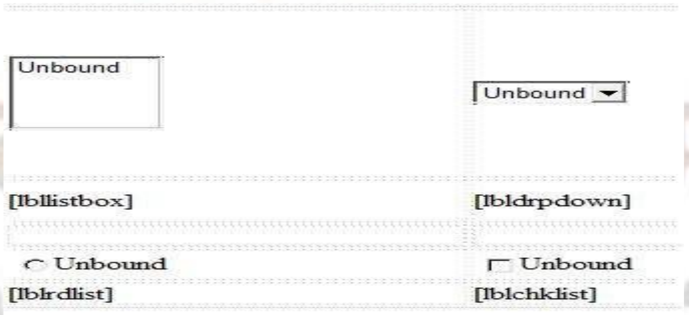
Example

Let us take the following steps:

Step (1) : Create a new website. Add a class named booklist by right clicking on the solution name in the Solution Explorer and choosing the item 'Class' from the 'Add Item' dialog box. Name it as booklist.cs.

```
using System;
using System.Data;
using System.Configuration;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
namespace databinding
{
   public class booklist
   {
      protected String bookname;
      protected String authorname;
      public booklist(String bname, String aname)
      {
         this.bookname = bname;
         this.authorname = aname;
      }
            public String Book
      {
        get
        {
          return this.bookname;
        }
        set
        {
          this.bookname = value;
        }
      }
          public String Author
      {
        get
        {
          return this.authorname;
        }
        set
        {
          this.authorname = value;
        }
      }
   }
}
```

Step (2) : Add four list controls on the page a list box control, a radio button list, a check box list, and a drop down list and four labels along with these list controls. The page should look like this in design view:



The source file should look as the following:

```
<form id="form1" runat="server">
  <div>
        <table style="width: 559px">
        <tr>
          <td style="width: 228px; height: 157px;">
            <asp:ListBox ID="ListBox1" runat="server" AutoPostBack="True"
              OnSelectedIndexChanged="ListBox1_SelectedIndexChanged">
            </asp:ListBox>
          </td>
          <td style="height: 157px">
            <asp:DropDownList ID="DropDownList1" runat="server"
              AutoPostBack="True"
OnSelectedIndexChanged="DropDownList1_SelectedIndexChanged">
            </asp:DropDownList>
          </td>
        </tr>
        <tr>
          <td style="width: 228px; height: 40px;">
            <asp:Label ID="lbllistbox" runat="server"></asp:Label>
          </td>
          <td style="height: 40px">
            <asp:Label ID="lbldrpdown" runat="server">
            </asp:Label>
          </td>
        </tr>
        <tr>
          <td style="width: 228px; height: 21px">
          </td>
          <td style="height: 21px">
          </td>
        </tr>
        <tr>
          <td style="width: 228px; height: 21px">
            <asp:RadioButtonList ID="RadioButtonList1" runat="server"
              AutoPostBack="True"
OnSelectedIndexChanged="RadioButtonList1_SelectedIndexChanged">
            </asp:RadioButtonList>
          </td>
          <td style="height: 21px">
```

```
           <asp:CheckBoxList ID="CheckBoxList1" runat="server"
             AutoPostBack="True"
OnSelectedIndexChanged="CheckBoxList1_SelectedIndexChanged">
           </asp:CheckBoxList>
         </td>
       </tr>
       <tr>
         <td style="width: 228px; height: 21px">
           <asp:Label ID="lblrdlist" runat="server">
           </asp:Label>
         </td>
         <td style="height: 21px">
           <asp:Label ID="lblchklist" runat="server">
           </asp:Label>
         </td>
       </tr>
     </table>
       </div>
</form>
```

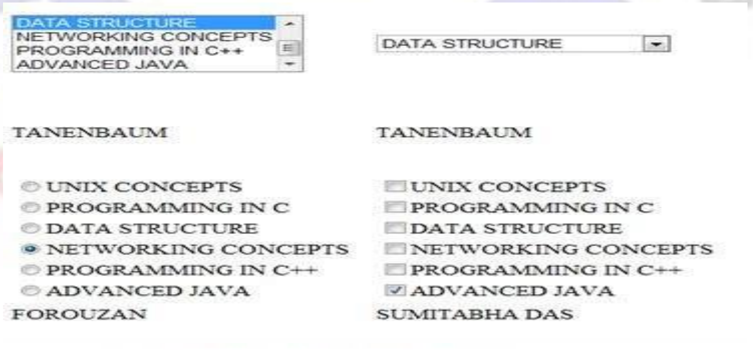
Step (3) : Finally, write the following code behind routines of the application:

```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
     IList bklist = createbooklist();
      if (!this.IsPostBack)
     {
        this.ListBox1.DataSource = bklist;
        this.ListBox1.DataTextField = "Book";
        this.ListBox1.DataValueField = "Author";
        this.DropDownList1.DataSource = bklist;
        this.DropDownList1.DataTextField = "Book";
        this.DropDownList1.DataValueField = "Author";
        this.RadioButtonList1.DataSource = bklist;
        this.RadioButtonList1.DataTextField = "Book";
        this.RadioButtonList1.DataValueField = "Author";
        this.CheckBoxList1.DataSource = bklist;
        this.CheckBoxList1.DataTextField = "Book";
        this.CheckBoxList1.DataValueField = "Author";
         this.DataBind();
    }
}
   protected IList createbooklist()
{
   ArrayList allbooks = new ArrayList();
   booklist bl;
        bl = new booklist("UNIX CONCEPTS", "SUMITABHA DAS");
   allbooks.Add(bl);
        bl = new booklist("PROGRAMMING IN C", "RICHI KERNIGHAN");
   allbooks.Add(bl);
        bl = new booklist("DATA STRUCTURE", "TANENBAUM");
   allbooks.Add(bl);
        bl = new booklist("NETWORKING CONCEPTS", "FOROUZAN");
   allbooks.Add(bl);
        bl = new booklist("PROGRAMMING IN C++", "B. STROUSTROUP");
   allbooks.Add(bl);

   bl = new booklist("ADVANCED JAVA", "SUMITABHA DAS");
   allbooks.Add(bl);
        return allbooks;
}
   protected void ListBox1_SelectedIndexChanged(object sender, EventArgs e)
{
   this.lbllistbox.Text = this.ListBox1.SelectedValue;
}
   protected void DropDownList1_SelectedIndexChanged(object sender, EventArgs e)
{
   this.lbldrpdown.Text = this.DropDownList1.SelectedValue;
}

protected void RadioButtonList1_SelectedIndexChanged(object sender, EventArgs e)
```

```
  {
    this.lblrdlist.Text = this.RadioButtonList1.SelectedValue;
  }
    protected void CheckBoxList1_SelectedIndexChanged(object sender, EventArgs e)
  {
    this.lblchklist.Text = this.CheckBoxList1.SelectedValue;
  }
}
```

Observe the following:

- The booklist class has two properties: bookname and authorname.
- The createbooklist method is a user defined method that creates an array of booklist objects named allbooks.
- The Page_Load event handler ensures that a list of books is created. The list is of IList type, which implements the IEnumerable interface and capable of being bound to the list controls. The page load event handler binds the IList object 'bklist' with the list controls. The bookname property is to be displayed and the authorname property is considered as the value.
- When the page is run, if the user selects a book, its name is selected and displayed by the list controls whereas the corresponding labels display the author name, which is the corresponding value for the selected index of the list control.



ASP.NET - Custom Controls

ASP.NET allows the users to create controls. These user defined controls are categorized into:

- User controls
- Custom controls

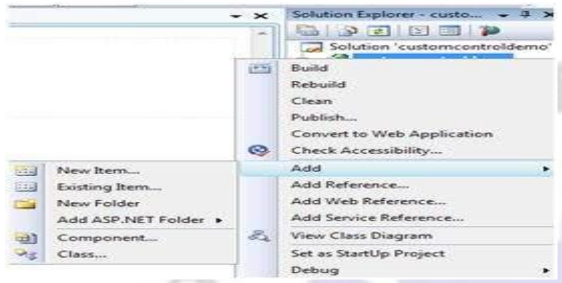
User Controls

User controls behaves like miniature ASP.NET pages or web forms, which could be used by many other pages. These are derived from the System.Web.UI.UserControl class. These controls have the following characteristics:

- They have an .ascx extension.

- They may not contain any <html>, <body>, or <form> tags.
- They have a Control directive instead of a Page directive.

  To understand the concept, let us create a simple user control, which will work as footer for the web pages. To create and use the user control, take the following steps:
- Create a new web application.
- Right click on the project folder on the Solution Explorer and choose Add New Item.



- Select Web User Control from the Add New Item dialog box and name it footer.ascx. Initially, the footer.ascx contains only a Control directive.

```
• <%@ Control Language="C#" AutoEventWireup="true" CodeBehind="footer.ascx.cs"
  Inherits="customcontroldemo.footer" %>
```

- Add the following code to the file:

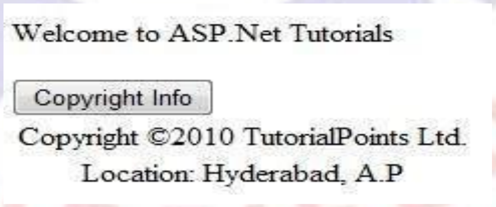
```
• <table>
•   <tr>
•     <td align="center"> Copyright ©2010 TutorialPoints Ltd.</td>
•   </tr>
•   <tr>
•     <td align="center"> Location: Hyderabad, A.P </td>
•   </tr>
</table>
```

To add the user control to your web page, you must add the Register directive and an instance of the user control to the page. The following code shows the content file:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="customcontroldemo._Default" %>
  <%@ Register Src="~/footer.ascx" TagName="footer" TagPrefix="Tfooter" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
```

```
      <title>
        Untitled Page
      </title>
   </head>
      <body>
        <form id="form1" runat="server">
        <div>
                <asp:Label ID="Label1" runat="server" Text="Welcome to ASP.Net Tutorials
"></asp:Label>
          <br />  <br />
          <asp:Button ID="Button1" runat="server" onclick="Button1_Click"  Text="Copyright
Info" />


        </div>
        <Tfooter:footer ID="footer1" runat="server" />
      </form>
        </body>
</html>
```

When executed, the page shows the footer and this control could be used in all the pages of your website.


Welcome to ASP.Net Tutorials

Copyright Info

Copyright ©2010 TutorialPoints Ltd.
Location: Hyderabad, A.P

Observe the following:

(1) The Register directive specifies a tag name as well as tag prefix for the control.

```
<%@ Register Src="~/footer.ascx" TagName="footer" TagPrefix="Tfooter" %>
```

(2) The following tag name and prefix should be used while adding the user control on the page:

```
<Tfooter:footer ID="footer1" runat="server" />
```

Custom Controls

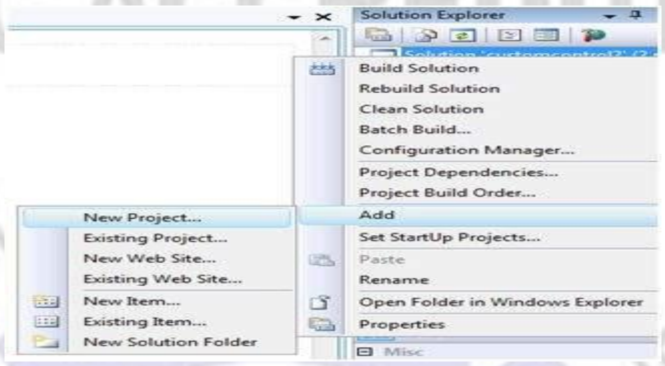
Custom controls are deployed as individual assemblies. They are compiled into a Dynamic Link Library (DLL) and used as any other ASP.NET server control. They could be created in either of the following way:

- By deriving a custom control from an existing control
- By composing a new custom control combing two or more existing controls.

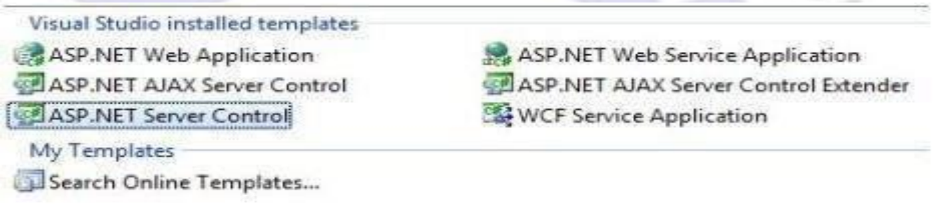
- By deriving from the base control class.

  To understand the concept, let us create a custom control, which will simply render a text message on the browser. To create this control, take the following steps:

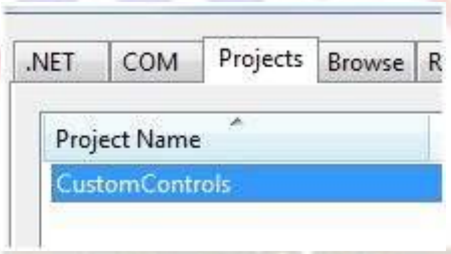
  Create a new website. Right click the solution (not the project) at the top of the tree in the Solution Explorer.



In the New Project dialog box, select ASP.NET Server Control from the project templates.


Visual Studio installed templates
ASP.NET Web Application
ASP.NET AJAX Server Control
ASP.NET Server Control
ASP.NET Web Service Application
ASP.NET AJAX Server Control Extender
WCF Service Application
My Templates
Search Online Templates...

The above step adds a new project and creates a complete custom control to the solution, called ServerControl1. In this example, let us name the project CustomControls. To use this control, this must be added as a reference to the web site before registering it on a page. To add a reference to the existing project, right click on the project (not the solution), and click Add Reference. Select the CustomControls project from the Projects tab of the Add Reference dialog box. The Solution Explorer should show the reference.



To use the control on a page, add the Register directive just below the @Page directive:

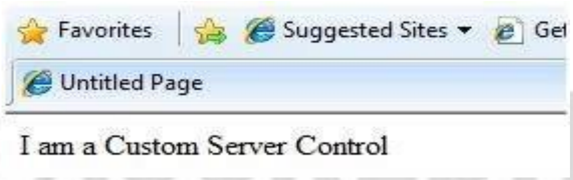
```
<%@ Register Assembly="CustomControls"  Namespace="CustomControls"  TagPrefix="ccs" %>
```

Further, you can use the control, similar to any other controls.

```
<form id="form1" runat="server">
   <div>
      <ccs:ServerControl1 runat="server" Text = "I am a Custom Server Control" />
   </div>
```

```
</form>
```

When executed, the Text property of the control is rendered on the browser as shown:



I am a Custom Server Control

Working with Custom Controls

In the previous example, the value for the Text property of the custom control was set. ASP.NET added this property by default, when the control was created. The following code behind file of the control reveals this.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace CustomControls
{
  [DefaultProperty("Text")]
  [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1 >")]
    public class ServerControl1 : WebControl
  {
    [Bindable(true)]
    [Category("Appearance")]
    [DefaultValue("")]
    [Localizable(true)]
        public string Text
    {
      get
      {
        String s = (String)ViewState["Text"];
        return ((s == null) ? "[" + this.ID + "]" : s);
      }
            set
        {
          ViewState["Text"] = value;
        }
      }
          protected override void RenderContents(HtmlTextWriter output)
      {
        output.Write(Text);
      }
   }
}
```

The above code is automatically generated for a custom control. Events and methods could be added to the custom control class.

Example

Let us expand the previous custom control named SeverControl1. Let us give it a method named checkpalindrome, which gives it a power to check for palindromes. Palindromes are words/literals that spell the same when reversed. For example, Malayalam, madam, saras, etc.

Extend the code for the custom control, which should look as:

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Linq;
using System.Text;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace CustomControls
{
   [DefaultProperty("Text")]
   [ToolboxData("<{0}:ServerControl1 runat=server></{0}:ServerControl1  >")]
      public class ServerControl1 : WebControl
   {
      [Bindable(true)]
      [Category("Appearance")]
      [DefaultValue("")]
```

```
[Localizable(true)]
    public string Text
{
  get
  {
    String s = (String)ViewState["Text"];
    return ((s == null) ? "[" + this.ID + "]" : s);
  }
        set
  {
    ViewState["Text"] = value;
  }
}
    protected override void RenderContents(HtmlTextWriter output)
{
  if (this.checkpanlindrome())
  {
    output.Write("This is a palindrome: <br />");
    output.Write("<FONT size=5 color=Blue>");
    output.Write("<B>");
    output.Write(Text);
    output.Write("</B>");
    output.Write("</FONT>");
  }
  else
  {
    output.Write("This is not a palindrome: <br />");
    output.Write("<FONT size=5 color=red>");
    output.Write("<B>");
    output.Write(Text);
    output.Write("</B>");
    output.Write("</FONT>");
  }
}
    protected bool checkpanlindrome()
{
        if (this.Text != null)
        {
          String str = this.Text;
          String strtoupper = Text.ToUpper();
          char[] rev = strtoupper.ToCharArray();
          Array.Reverse(rev);
          String strrev = new String(rev);
                  if (strtoupper == strrev)
          {
            return true;
          }
          else
          {
            return false;
          }
        }
        else
        {
          return false;
        }
      }
    }
}
```

When you change the code for the control, you must build the solution by clicking Build --> Build Solution, so that the changes are reflected in your project. Add a text box and a button control to the page, so that the user can provide a text, it is checked for palindrome, when the button is clicked.

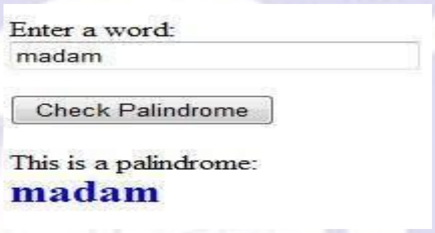
```
<form id="form1" runat="server">
  <div>
    Enter a word:
    <br />
    <asp:TextBox ID="TextBox1" runat="server" style="width:198px"> </asp:TextBox>
        <br /> <br />
        <asp:Button ID="Button1" runat="server onclick="Button1_Click" Text="Check
Palindrome" style="width:132px" />
```

```
         <br /> <br />
         <ccs:ServerControl1 ID="ServerControl11" runat="server" Text = "" />
   </div>
</form>
```

The Click event handler for the button simply copies the text from the text box to the text property of the custom control.

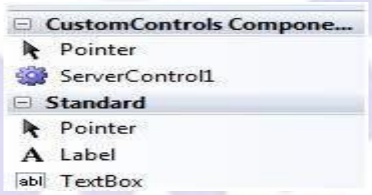
```
protected void Button1_Click(object sender, EventArgs e)
{
   this.ServerControl11.Text = this.TextBox1.Text;
}
```

When executed, the control successfully checks palindromes.



Observe the following:

(1) When you add a reference to the custom control, it is added to the toolbox and you can directly use it from the toolbox similar to any other control.



(2) The RenderContents method of the custom control class is overridden here, as you can add your own methods and events.

(3) The RenderContents method takes a parameter of HtmlTextWriter type, which is responsible for rendering on the browser.

ASP.NET - Personalization

Web sites are designed for repeated visits from the users. Personalization allows a site to remember the user identity and other information details, and it presents an individualistic environment to each user.

ASP.NET provides services for personalizing a web site to suit a particular client's taste and preference.

Understanding Profiles

ASP.NET personalization service is based on user profile. User profile defines the kind of information about the user that the site needs. For example, name, age, address, date of birth, and phone number.

This information is defined in the web.config file of the application and ASP.NET runtime reads and uses it. This job is done by the personalization providers.

The user profiles obtained from user data is stored in a default database created by ASP.NET. You can create your own database for storing profiles. The profile data definition is stored in the configuration file web.config.

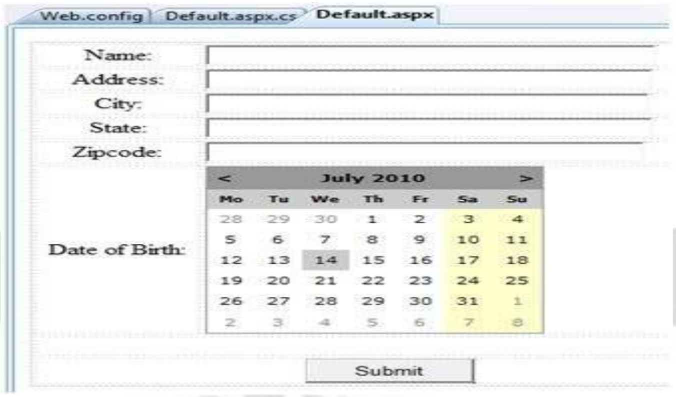
Example

Let us create a sample site, where we want our application to remember user details like name, address, date of birth etc. Add the profile details in the web.config file within the <system.web> element.

```
<configuration>
<system.web>
<profile>
  <properties>
    <add name="Name" type ="String"/>
    <add name="Birthday" type ="System.DateTime"/>
        <group name="Address">
      <add name="Street"/>
      <add name="City"/>
      <add name="State"/>
      <add name="Zipcode"/>
    </group>
     </properties>
</profile>
</system.web>
</configuration>
```

When the profile is defined in the web.config file, the profile could be used through the Profile property found in the current HttpContext and also available via page.

Add the text boxes to take the user input as defined in the profile and add a button for submitting the data:



Update Page_load to display profile information:

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    if (!this.IsPostBack)
    {
      ProfileCommon pc=this.Profile.GetProfile(Profile.UserName);
            if (pc != null)
      {
        this.txtname.Text = pc.Name;
        this.txtaddr.Text = pc.Address.Street;
        this.txtcity.Text = pc.Address.City;
        this.txtstate.Text = pc.Address.State;
        this.txtzip.Text = pc.Address.Zipcode;
        this.Calendar1.SelectedDate = pc.Birthday;
      }
    }
  }
}
```

Write the following handler for the Submit button, for saving the user data into the profile:

```
protected void btnsubmit_Click(object sender, EventArgs e)
{
  ProfileCommon pc=this.Profile.GetProfile(Profile.UserName);
    if (pc != null)
  {
    pc.Name = this.txtname.Text;
    pc.Address.Street = this.txtaddr.Text;
    pc.Address.City = this.txtcity.Text;
    pc.Address.State = this.txtstate.Text;
    pc.Address.Zipcode = this.txtzip.Text;
    pc.Birthday = this.Calendar1.SelectedDate;
        pc.Save();
  }
}
```

When the page is executed for the first time, the user needs to enter the information. However, next time the user details would be automatically loaded.

Attributes for the <add> Element

Apart from the name and type attributes that we have used, there are other attributes to the <add> element. Following table illustrates some of these attributes:

Attributes	**Description**
name	The name of the property.
type	By default the type is string but it allows any fully qualified class name as data type.
serializeAs	The format to use when serializing this value.
readOnly	A read only profile value cannot be changed, by default this property is false.
defaultValue	A default value that is used if the profile does not exist or does not have information.
allowAnonymous	A Boolean value indicating whether this property can be used with the anonymous profiles.
Provider	The profiles provider that should be used to manage just this property.

Anonymous Personalization

Anonymous personalization allows the user to personalize the site before identifying themselves. For example, Amazon.com allows the user to add items in the shopping cart before they log in. To enable this feature, the web.config file could be configured as:

```
<anonymousIdentification enabled ="true" cookieName=".ASPXANONYMOUSUSER"
   cookieTimeout="120000" cookiePath="/" cookieRequiresSSL="false"
   cookieSlidingExpiration="true" cookieprotection="Encryption"
   coolieless="UseDeviceProfile"/>
```

ASP.NET - Error Handling

Error handling in ASP.NET has three aspects:

- **Tracing -** tracing the program execution at page level or application level.
- **Error handling -** handling standard errors or custom errors at page level or application level.
- **Debugging -** stepping through the program, setting break points to analyze the code

In this chapter, we will discuss tracing and error handling and in this chapter, we will discuss debugging.

To understand the concepts, create the following sample application. It has a label control, a dropdown list, and a link. The dropdown list loads an array list of famous quotes and the selected quote is shown in the label below. It also has a hyperlink which has points to a nonexistent link.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="errorhandling._Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>
         Tracing, debugging and error handling
      </title>
   </head>
      <body>
      <form id="form1" runat="server">
            <div>
         <asp:Label ID="lblheading" runat="server" Text="Tracing, Debuggin  and Error
Handling">
         </asp:Label>
```

```
                <br /> <br />
                <asp:DropDownList    ID="ddlquotes"    runat="server"    AutoPostBack="True"
onselectedindexchanged="ddlquotes_SelectedIndexChanged">
        </asp:DropDownList>
                <br /> <br />
                <asp:Label ID="lblquotes" runat="server">
        </asp:Label>
                <br /> <br />
                <asp:HyperLink                    ID="HyperLink1"                    runat="server"
NavigateUrl="mylink.htm">Link to:</asp:HyperLink>
      </div>
          </form>
  </body>
  </html>
```

The code behind file:

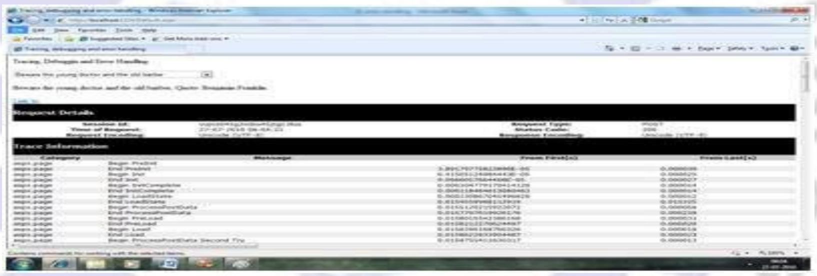
```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    if (!IsPostBack)
    {
      string[,] quotes =
      {
        {"Imagination is more important than Knowledge.", "Albert Einsten"},
        {"Assume a virtue, if you have it not" "Shakespeare"},
        {"A man cannot be comfortable without his own approval", "Mark Twain"},
        {"Beware the young doctor and the old barber", "Benjamin Franklin"},
        {"Whatever begun in anger ends in shame", "Benjamin Franklin"}
      };
            for (int i=0; i<quotes.GetLength(0); i++)
        ddlquotes.Items.Add(new ListItem(quotes[i,0], quotes[i,1]));
    }
  }
    protected void ddlquotes_SelectedIndexChanged(object sender, EventArgs e)
  {
      if (ddlquotes.SelectedIndex != -1)
      {
        lblquotes.Text = String.Format("{0}, Quote: {1}", ddlquotes.SelectedItem.Text,
ddlquotes.SelectedValue);
      }
   }
}
```

Tracing

To enable page level tracing, you need to modify the Page directive and add a Trace attribute as shown:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
   Inherits="errorhandling._Default" Trace ="true" %>
```

Now when you execute the file, you get the tracing information:



It provides the following information at the top:

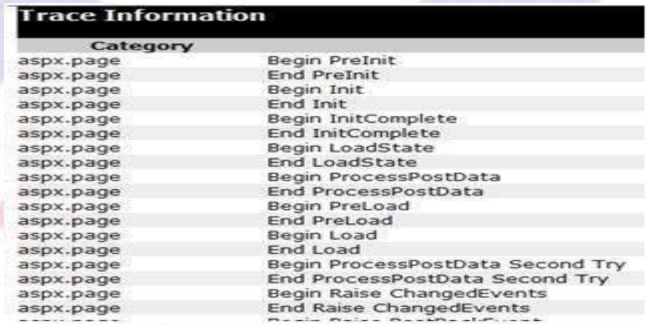
- Session ID
- Status Code
- Time of Request
- Type of Request
- Request and Response Encoding

The status code sent from the server, each time the page is requested shows the name and time of error if any. The following table shows the common HTTP status codes:

Number	**Description**
Informational (100 - 199)	
100	Continue
101	Switching protocols
Successful (200 - 299)	
200	OK
204	No content

Redirection (300 - 399)	
301	Moved permanently
305	Use proxy
307	Temporary redirect
Client Errors (400 - 499)	
400	Bad request
402	Payment required
404	Not found
408	Request timeout
417	Expectation failed
Server Errors (500 - 599)	
500	Internal server error
503	Service unavailable
505	HTTP version not supported

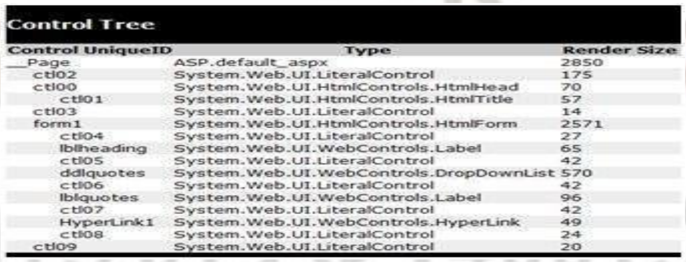
Under the top level information, there is Trace log, which provides details of page life cycle. It provides elapsed time in seconds since the page was initialized.



Trace Information

Category	
aspx.page	Begin PreInit
aspx.page	End PreInit
aspx.page	Begin Init
aspx.page	End Init
aspx.page	Begin InitComplete
aspx.page	End InitComplete
aspx.page	Begin LoadState
aspx.page	End LoadState
aspx.page	Begin ProcessPostData
aspx.page	End ProcessPostData
aspx.page	Begin PreLoad
aspx.page	End PreLoad
aspx.page	Begin Load
aspx.page	End Load
aspx.page	Begin ProcessPostData Second Try
aspx.page	End ProcessPostData Second Try
aspx.page	Begin Raise ChangedEvents
aspx.page	End Raise ChangedEvents

The next section is control tree, which lists all controls on the page in a hierarchical manner:



Control Tree

Control UniqueID	Type	Render Size
__Page	ASP.default_aspx	2850
ctl02	System.Web.UI.LiteralControl	175
ctl00	System.Web.UI.HtmlControls.HtmlHead	70
ctl01	System.Web.UI.HtmlControls.HtmlTitle	57
ctl03	System.Web.UI.LiteralControl	14
form1	System.Web.UI.HtmlControls.HtmlForm	2571
ctl04	System.Web.UI.LiteralControl	27
lblheading	System.Web.UI.WebControls.Label	65
ctl05	System.Web.UI.LiteralControl	42
ddlquotes	System.Web.UI.WebControls.DropDownList	570
ctl06	System.Web.UI.LiteralControl	42
lblquotes	System.Web.UI.WebControls.Label	96
ctl07	System.Web.UI.LiteralControl	42
HyperLink1	System.Web.UI.WebControls.HyperLink	49
ctl08	System.Web.UI.LiteralControl	24
ctl09	System.Web.UI.LiteralControl	20

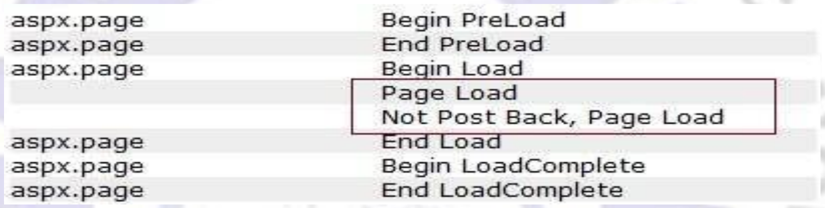
Last in the Session and Application state summaries, cookies, and headers collections followed by list of all server variables.

The Trace object allows you to add custom information to the trace output. It has two methods to accomplish this: the Write method and the Warn method.

Change the Page_Load event handler to check the Write method:

```
protected void Page_Load(object sender, EventArgs e)
{
   Trace.Write("Page Load");
      if (!IsPostBack)
   {
      Trace.Write("Not Post Back, Page Load");
      string[,] quotes =
      .......................
   }
}
```

Run to observe the effects:

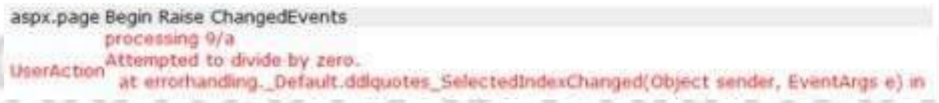


```
aspx.page                      Begin PreLoad
aspx.page                      End PreLoad
aspx.page                      Begin Load
                               Page Load
                               Not Post Back, Page Load
aspx.page                      End Load
aspx.page                      Begin LoadComplete
aspx.page                      End LoadComplete
```

To check the Warn method, let us forcibly enter some erroneous code in the selected index changed event handler:

```
try
{
   int a = 0;
   int b = 9 / a;
}catch (Exception e)
{
   Trace.Warn("UserAction", "processing 9/a", e);
}
```

Try-Catch is a C# programming construct. The try block holds any code that may or may not produce error and the catch block catches the error. When the program is run, it sends the warning in the trace log.



```
aspx.page Begin Raise ChangedEvents
           processing 9/a
UserAction Attempted to divide by zero.
              at errorhandling._Default.ddlquotes_SelectedIndexChanged(Object sender, EventArgs e) in
```

Application level tracing applies to all the pages in the web site. It is implemented by putting the following code lines in the web.config file:

```
<system.web>
   <trace enabled="true" />
```

```
</system.web>
```

Error Handling

Although ASP.NET can detect all runtime errors, still some subtle errors may still be there. Observing the errors by tracing is meant for the developers, not for the users. Hence, to intercept such occurrence, you can add error handing settings in the web.config file of the application. It is application-wide error handling. For example, you can add the following lines in the web.config file:

```
<configuration>
  <system.web>
      <customErrors mode="RemoteOnly" defaultRedirect="GenericErrorPage.htm">
      <error statusCode="403" redirect="NoAccess.htm"/>
      <error statusCode="404" redirect="FileNotFound.htm" />
    </customErrors>
      </system.web>
<configuration>
```

The <customErrors> section has the possible attributes:

- **Mode** It enables or disables custom error pages. It has the three possible values:
- **On**: displays the custom pages.
- **Off** : displays ASP.NET error pages (yellow pages)
- **Remote Only**: It displays custom errors to client, display ASP.NET errors locally.
- **Default Redirect**: It contains the URL of the page to be displayed in case of unhandled errors.

To put different custom error pages for different type of errors, the <error> sub tags are used, where different error pages are specified, based on the status code of the errors.

To implement page level error handling, the Page directive could be modified:

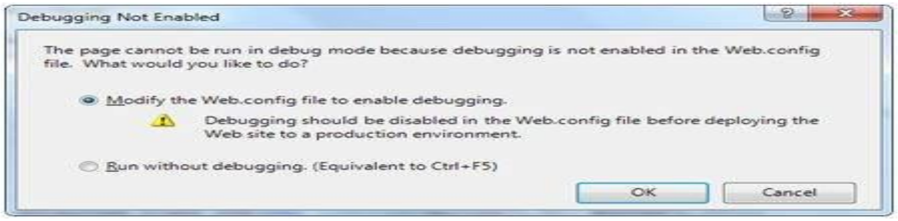
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
  Inherits="errorhandling._Default" Trace ="true" ErrorPage="PageError.htm" %>
```

Because ASP.NET Debugging is an important subject in itself, so we would discuss it in the next chapter separately.

ASP.NET - Debugging

Debugging allows the developers to see how the code works in a step-by-step manner, how the values of the variables change, how the objects are created and destroyed, etc.

When the site is executed for the first time, Visual Studio displays a prompt asking whether it should be enabled for debugging:


Debugging Not Enabled

The page cannot be run in debug mode because debugging is not enabled in the Web.config file. What would you like to do?

● Modify the Web.config file to enable debugging.

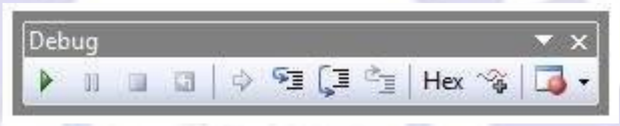
Debugging should be disabled in the Web.config file before deploying the Web site to a production environment.

○ Run without debugging. (Equivalent to Ctrl+F5)

When debugging is enabled, the following lines of codes are shown in the web.config:

```
<system.web>
  <compilation debug="true">
    <assemblies>
    </assemblies>
  </compilation>
</system.web>
```

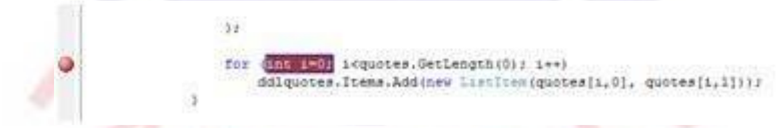
The Debug toolbar provides all the tools available for debugging:



Breakpoints

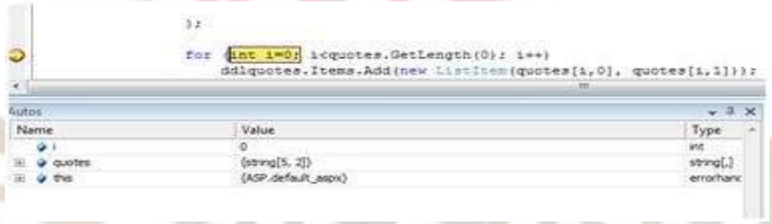
Breakpoints specifies the runtime to run a specific line of code and then stop execution so that the code could be examined and perform various debugging jobs such as, changing the value of the variables, step through the codes, moving in and out of functions and methods etc.

To set a breakpoint, right click on the code and choose insert break point. A red dot appears on the left margin and the line of code is highlighted as shown:



```
};

for (int i=0; i<quotes.GetLength(0); i++)
    ddlquotes.Items.Add(new ListItem(quotes[i,0], quotes[i,1]));
}
```

Next when you execute the code, you can observe its behavior.



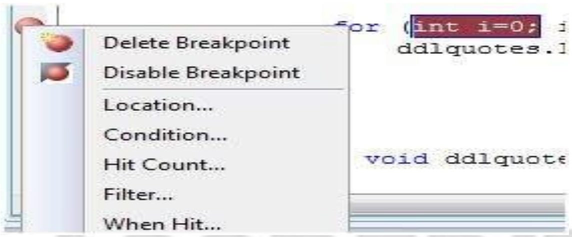
```
};

for (int i=0; i<quotes.GetLength(0); i++)
    ddlquotes.Items.Add(new ListItem(quotes[i,0], quotes[i,1]));
```

Autos

Name	Value	Type
i	0	int
quotes	{string[5, 2]}	string[,]
this	{ASP.default_aspx}	errorhanc

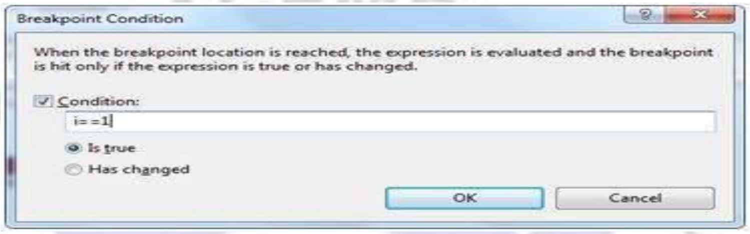
At this stage, you can step through the code, observe the execution flow and examine the value of the variables, properties, objects, etc.

You can modify the properties of the breakpoint from the Properties menu obtained by right clicking the breakpoint glyph:



```
for (int i=0; i
    ddlquotes.l

void ddlquote
```

The location dialog box shows the location of the file, line number and the character number of the selected code. The condition menu item allows you to enter a valid expression, which is evaluated when the program execution reaches the breakpoint:



Breakpoint Condition

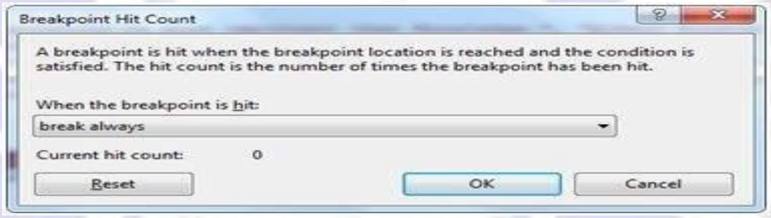
When the breakpoint location is reached, the expression is evaluated and the breakpoint is hit only if the expression is true or has changed.

[✓] Condition:

i==1

[✓ Is true]
Has changed

The Hit Count menu item displays a dialog box that shows the number of times the break point has been executed.



Breakpoint Hit Count

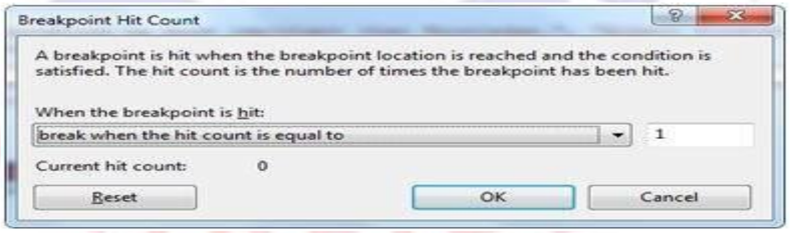
A breakpoint is hit when the breakpoint location is reached and the condition is satisfied. The hit count is the number of times the breakpoint has been hit.

When the breakpoint is hit:

break always

Current hit count: 0

Clicking on any option presented by the drop down list opens an edit field where a target hit count is entered. This is particularly helpful in analyzing loop constructs in code.



Breakpoint Hit Count

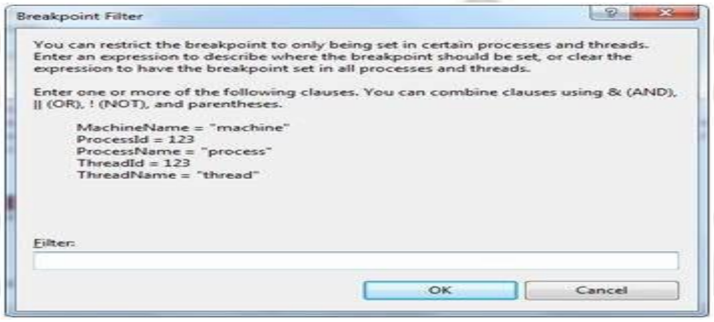
A breakpoint is hit when the breakpoint location is reached and the condition is satisfied. The hit count is the number of times the breakpoint has been hit.

When the breakpoint is hit:

break when the hit count is equal to 1

Current hit count: 0

The Filter menu item allows setting a filter for specifying machines, processes, or threads or any combination, for which the breakpoint will be effective.



Breakpoint Filter

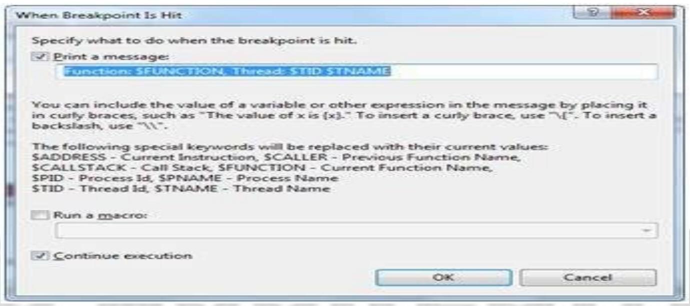
You can restrict the breakpoint to only being set in certain processes and threads. Enter an expression to describe where the breakpoint should be set, or clear the expression to have the breakpoint set in all processes and threads.

Enter one or more of the following clauses. You can combine clauses using & (AND), || (OR), ! (NOT), and parentheses.

```
MachineName = "machine"
ProcessId = 123
ProcessName = "process"
ThreadId = 123
ThreadName = "thread"
```

Filter:

The When Hit menu item allows you to specify what to do when the break point is hit.



When Breakpoint Is Hit

Specify what to do when the breakpoint is hit.

[✓] Print a message:

Function: $FUNCTION, Thread: $TID $TNAME

You can include the value of a variable or other expression in the message by placing it in curly braces, such as "The value of x is {x}." To insert a curly brace, use "\{". To insert a backslash, use "\\".

The following special keywords will be replaced with their current values:
$ADDRESS - Current Instruction, $CALLER - Previous Function Name,
$CALLSTACK - Call Stack, $FUNCTION - Current Function Name,
$PID - Process Id, $PNAME - Process Name
$TID - Thread Id, $TNAME - Thread Name

[ ] Run a macro:

[✓] Continue execution

The Debug Windows

Visual Studio provides the following debug windows, each of which shows some program information. The following table lists the windows:

Window	**Description**
Immediate	Displays variables and expressions.
Autos	Displays all variables in the current and previous statements.
Locals	Displays all variables in the current context.
Watch	Displays up to four different sets of variables.
Call Stack	Displays all methods in the call stack.
Threads	Displays and control threads.

ASP.NET - LINQ

Most applications are data-centric, however most of the data repositories are relational databases. Over the years, designers and developers have designed applications based on object models.

The objects are responsible for connecting to the data access components - called the Data Access Layer (DAL). Here we have three points to consider:

- All the data needed in an application are not stored in the same source. The source could be a relation database, some business object, XML file, or a web service.
- Accessing in-memory object is simpler and less expensive than accessing data from a database or XML file.
- The data accessed are not used directly, but needs to be sorted, ordered, grouped, altered etc.

Hence if there is one tool that makes all kind of data access easy that allows joining data from such disparate data sources and perform standard data processing operations, in few lines of codes, it would be of great help.

LINQ or Language-Integrated Query is such a tool. LINQ is set of extensions to the .Net Framework 3.5 and its managed languages that set the query as an object. It defines a common syntax and a programming model to query different types of data using a common language.

The relational operators like Select, Project, Join, Group, Partition, Set operations etc., are implemented in LINQ and the C# and VB compilers in the .Net framework 3.5, which support the LINQ syntax makes it possible to work with a configured data store without resorting to ADO.NET.

For example, querying the Customers table in the Northwind database, using LINQ query in C#, the code would be:

```
var data = from c in dataContext.Customers
where c.Country == "Spain"
select c;
```

Where:

- The 'from' keyword logically loops through the contents of the collection.
- The expression with the 'where' keyword is evaluated for each object in the collection.
- The 'select' statement selects the evaluated object to add to the list being returned.
- The 'var' keyword is for variable declaration. Since the exact type of the returned object is not known, it indicates that the information will be inferred dynamically.

LINQ query can be applied to any data-bearing class that inherits from IEnumerable<T>, here T is any data type, for example, List<Book>.

Let us look at an example to understand the concept. The example uses the following class: Books.cs

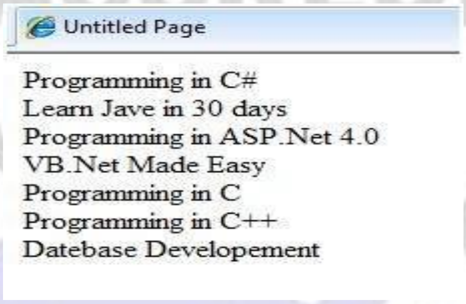
```
public class Books
{
  public string ID {get; set;}
  public string Title { get; set; }
  public decimal Price { get; set; }
  public DateTime DateOfRelease { get; set; }
  public static List<Books> GetBooks()
  {
    List<Books> list = new List<Books>();
    list.Add(new Books { ID = "001",
      Title = "Programming in C#",
      Price = 634.76m,
      DateOfRelease = Convert.ToDateTime("2010-02-05") });
```

```
         list.Add(new Books { ID = "002",
         Title = "Learn Java in 30 days",
         Price = 250.76m,
         DateOfRelease = Convert.ToDateTime("2011-08-15") });
           list.Add(new Books { ID = "003",
         Title = "Programming in ASP.Net 4.0",
         Price = 700.00m,
         DateOfRelease = Convert.ToDateTime("2011-02-05") });
           list.Add(new Books { ID = "004",
         Title = "VB.Net Made Easy",
         Price = 500.99m,
         DateOfRelease = Convert.ToDateTime("2011-12-31") });
           list.Add(new Books { ID = "005",
         Title = "Programming in C",
         Price = 314.76m,
         DateOfRelease = Convert.ToDateTime("2010-02-05") });
           list.Add(new Books { ID = "006",
         Title = "Programming in C++",
         Price = 456.76m,
         DateOfRelease = Convert.ToDateTime("2010-02-05") });
           list.Add(new Books { ID = "007",
         Title = "Datebase Developement",
         Price = 1000.76m,
         DateOfRelease = Convert.ToDateTime("2010-02-05") });
              return list;
   }
}
```

The web page using this class has a simple label control, which displays the titles of the books. The Page_Load event creates a list of books and returns the titles by using LINQ query:

```
public partial class simplequery : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      List<Books> books = Books.GetBooks();
      var booktitles = from b in books select b.Title;
      foreach (var title in booktitles)
        lblbooks.Text += String.Format("{0} <br />", title);
   }
}
```

When the page is executed, the label displays the results of the query:



```
Programming in C#
Learn Jave in 30 days
Programming in ASP.Net 4.0
VB.Net Made Easy
Programming in C
Programming in C++
Datebase Developement
```

The above LINQ expression:

```
var booktitles =
from b in books
select b.Title;
```

Is equivalent to the following SQL query:

```
SELECT Title from Books
```

LINQ Operators

Apart from the operators used so far, there are several other operators, which implement all query clauses. Let us look at some of the operators and clauses.

The Join clause

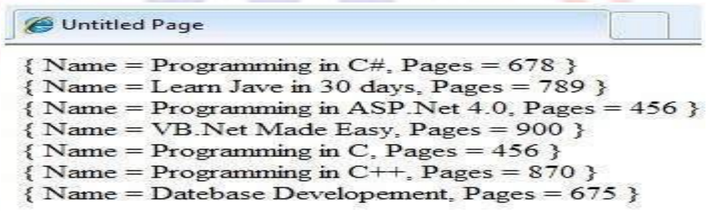
The 'join clause' in SQL is used for joining two data tables and displays a data set containing columns from both the tables. LINQ is also capable of that. To check this, add another class named Saledetails.cs in the previous project:

```
public class Salesdetails
{
  public int sales { get; set; }
  public int pages { get; set; }
  public string ID {get; set;}
  public static IEnumerable<Salesdetails> getsalesdetails()
  {
    Salesdetails[] sd =
    {
        new Salesdetails { ID = "001", pages=678, sales = 110000},
        new Salesdetails { ID = "002", pages=789, sales = 60000},
        new Salesdetails { ID = "003", pages=456, sales = 40000},
        new Salesdetails { ID = "004", pages=900, sales = 80000},
        new Salesdetails { ID = "005", pages=456, sales = 90000},
        new Salesdetails { ID = "006", pages=870, sales = 50000},
        new Salesdetails { ID = "007", pages=675, sales = 40000},
      };
          return sd.OfType<Salesdetails>();
  }
}
```

Add the codes in the Page_Load event handler to query on both the tables using the join clause:

```
protected void Page_Load(object sender, EventArgs e)
{
  IEnumerable<Books> books = Books.GetBooks();
  IEnumerable<Salesdetails> sales = Salesdetails.getsalesdetails();
    var booktitles = from b in books join s in sales on b.ID equals s.ID
    select new { Name = b.Title, Pages = s.pages };
      foreach (var title in booktitles)
    lblbooks.Text += String.Format("{0} <br />", title);
}
```

The resulting page is as shown:



Untitled Page

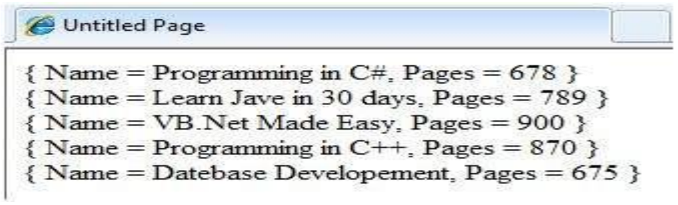
```
{ Name = Programming in C#, Pages = 678 }
{ Name = Learn Jave in 30 days, Pages = 789 }
{ Name = Programming in ASP.Net 4.0, Pages = 456 }
{ Name = VB.Net Made Easy, Pages = 900 }
{ Name = Programming in C, Pages = 456 }
{ Name = Programming in C++, Pages = 870 }
{ Name = Datebase Developement, Pages = 675 }
```

The Where clause

The 'where clause' allows adding some conditional filters to the query. For example, if you want to see the books, where the number of pages are more than 500, change the Page_Load event handler to:

```
var booktitles = from b in books join s in sales on b.ID equals s.ID
  where s.pages > 500 select new { Name = b.Title, Pages = s.pages };
```

The query returns only those rows, where the number of pages is more than 500:



Untitled Page

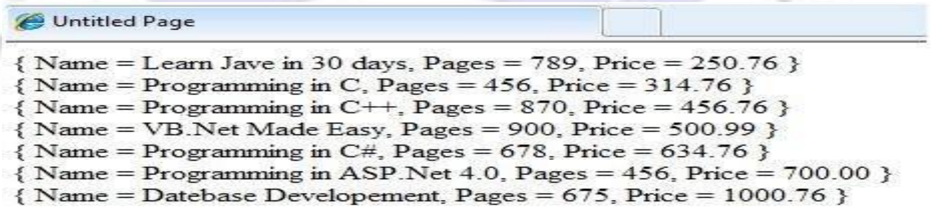
```
{ Name = Programming in C#, Pages = 678 }
{ Name = Learn Jave in 30 days, Pages = 789 }
{ Name = VB.Net Made Easy, Pages = 900 }
{ Name = Programming in C++, Pages = 870 }
{ Name = Datebase Developement, Pages = 675 }
```

Orderby and Orderbydescending Clauses

These clauses allow sorting the query results. To query the titles, number of pages and price of the book, sorted by the price, write the following code in the Page_Load event handler:

```
var booktitles = from b in books join s in sales on b.ID equals s.ID
   orderby b.Price select new { Name = b.Title,  Pages = s.pages, Price = b.Price};
```

The returned tuples are:



Untitled Page

```
{ Name = Learn Jave in 30 days, Pages = 789, Price = 250.76 }
{ Name = Programming in C, Pages = 456, Price = 314.76 }
{ Name = Programming in C++, Pages = 870, Price = 456.76 }
{ Name = VB.Net Made Easy, Pages = 900, Price = 500.99 }
{ Name = Programming in C#, Pages = 678, Price = 634.76 }
{ Name = Programming in ASP.Net 4.0, Pages = 456, Price = 700.00 }
{ Name = Datebase Developement, Pages = 675, Price = 1000.76 }
```

The Let clause

The let clause allows defining a variable and assigning it a value calculated from the data values. For example, to calculate the total sale from the above two sales, you need to calculate:

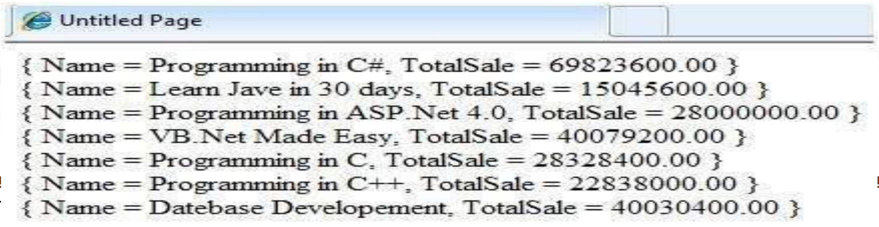
```
TotalSale = Price of the Book * Sales
```

To achieve this, add the following code snippets in the Page_Load event handler:

The let clause allows defining a variable and assigning it a value calculated from the data values. For example, to calculate the total sale from the above two sales, you need to calculate:

```
var booktitles = from b in book join s in sales on b.ID equals s.ID
   let totalprofit = (b.Price * s.sales)
   select new { Name = b.Title, TotalSale = totalprofit};
```

The resulting query page is as shown:



Untitled Page

```
{ Name = Programming in C#, TotalSale = 69823600.00 }
{ Name = Learn Jave in 30 days, TotalSale = 15045600.00 }
{ Name = Programming in ASP.Net 4.0, TotalSale = 28000000.00 }
{ Name = VB.Net Made Easy, TotalSale = 40079200.00 }
{ Name = Programming in C, TotalSale = 28328400.00 }
{ Name = Programming in C++, TotalSale = 22838000.00 }
{ Name = Datebase Developement, TotalSale = 40030400.00 }
```

ASP.NET - Security

Implementing security in a site has the following aspects:

- **Authentication** : It is the process of ensuring the user's identity and authenticity. ASP.NET allows four types of authentications:
  - Windows Authentication
  - Forms Authentication
  - Passport Authentication
  - Custom Authentication
- **Authorization** : It is the process of defining and allotting specific roles to specific users.
- **Confidentiality** : It involves encrypting the channel between the client browser and the web server.
- **Integrity** : It involves maintaining the integrity of data. For example, implementing digital signature.

Forms-Based Authentication

Traditionally, forms-based authentication involves editing the web.config file and adding a login page with appropriate authentication code.

The web.config file could be edited and the following codes written on it:

```
<configuration>
<system.web>
  <authentication mode="Forms">
    <forms loginUrl ="login.aspx"/>
  </authentication>
    <authorization>
    <deny users="?"/>
  </authorization>
</system.web>
...
...
</configuration>
```

The login.aspx page mentioned in the above code snippet could have the following code behind file with the usernames and passwords for authentication hard coded into it.

```
protected bool authenticate(String uname, String pass)
{
  if(uname == "Tom")
  {
    if(pass == "tom123")
```

```
        return true;
  }
      if(uname == "Dick")
  {
      if(pass == "dick123")
        return true;
  }
      if(uname == "Harry")
  {
      if(pass == "har123")
        return true;
  }
      return false;
}
public void OnLogin(Object src, EventArgs e)
{
   if (authenticate(txtuser.Text, txtpwd.Text))
   {
      FormsAuthentication.RedirectFromLoginPage(txtuser.Text, chkrem.Checked);
   }
   else
   {
      Response.Write("Invalid user name or password");
   }
}
```

Observe that the FormsAuthentication class is responsible for the process of authentication.

However, Visual Studio allows you to implement user creation, authentication, and authorization with seamless ease without writing any code, through the Web Site Administration tool. This tool allows creating users and roles.

Apart from this, ASP.NET comes with readymade login controls set, which has controls performing all the jobs for you.

Implementing Forms-Based Security

To set up forms-based authentication, you need the following:

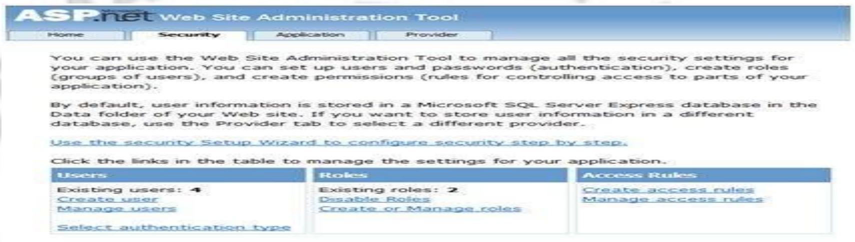
- A database of users to support the authentication process
- A website that uses the database

- User accounts
- Roles
- Restriction of users and group activities
- A default page, to display the login status of the users and other information.
- A login page, to allow users to log in, retrieve password, or change password

To create users, take the following steps:

Step (1) : Choose Website -> ASP.NET Configuration to open the Web Application Administration Tool.

Step (2) : Click on the Security tab.



You can use the Web Site Administration Tool to manage all the security settings for your application. You can set up users and passwords (authentication), create roles (groups of users), and create permissions (rules for controlling access to parts of your application).

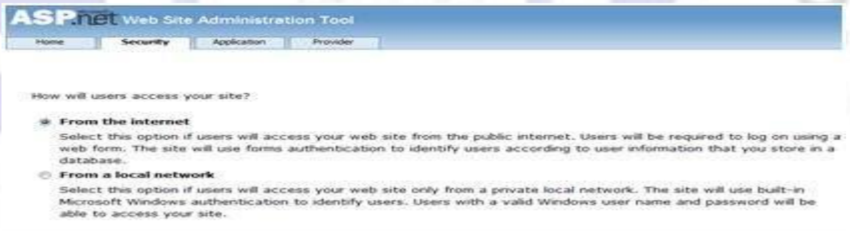
By default, user information is stored in a Microsoft SQL Server Express database in the Data folder of your Web site. If you want to store user information in a different database, use the Provider tab to select a different provider.

Use the security Setup Wizard to configure security step by step.

Click the links in the table to manage the settings for your application.

Users	Roles	Access Rules
Existing users: **4** Create user Manage users Select authentication type	Existing roles: **2** Disable Roles Create or Manage roles	Create access rules Manage access rules

Step (3) : Select the authentication type to 'Forms based authentication' by selecting the 'From the Internet' radio button.

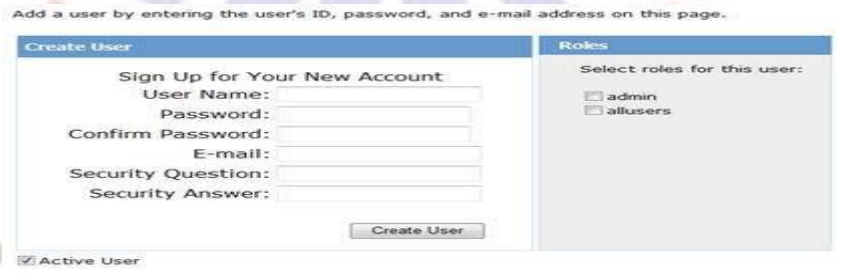


How will users access your site?

● **From the internet**
Select this option if users will access your web site from the public internet. Users will be required to log on using a web form. The site will use forms authentication to identify users according to user information that you store in a database.

○ **From a local network**
Select this option if users will access your web site only from a private local network. The site will use built-in Microsoft Windows authentication to identify users. Users with a valid Windows user name and password will be able to access your site.

Step (4) : Click on 'Create Users' link to create some users. If you already had created roles, you could assign roles to the user, right at this stage.



Add a user by entering the user's ID, password, and e-mail address on this page.

Create User

Sign Up for Your New Account
User Name:
Password:
Confirm Password:
E-mail:
Security Question:
Security Answer:

[✓] Active User

Roles

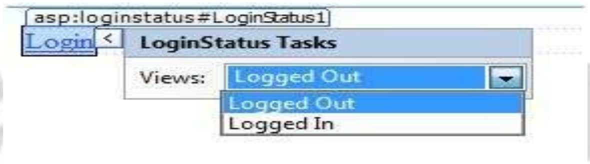
Select roles for this user:
☐ admin
☐ allusers

Step (5) : Create a web site and add the following pages:

- Welcome.aspx
- Login.aspx
- CreateAccount.aspx
- PasswordRecovery.aspx
- ChangePassword.aspx

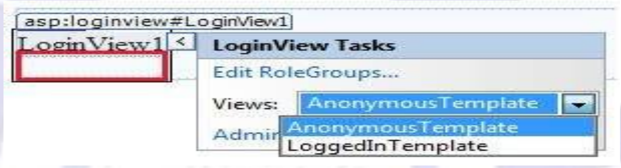
Step (6) : Place a LoginStatus control on the Welcome.aspx from the login section of the toolbox. It has two templates: LoggedIn and LoggedOut.

In LoggedOut template, there is a login link and in the LoggedIn template, there is a logout link on the control. You can change the login and logout text properties of the control from the Properties window.



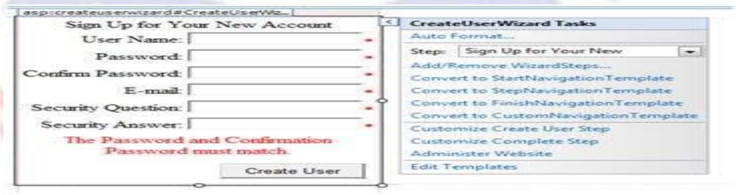
Step (7) : Place a LoginView control from the toolbox below the LoginStatus control. Here, you can put texts and other controls (hyperlinks, buttons etc.), which are displayed based on whether the user is logged in or not.

This control has two view templates: Anonymous template and LoggedIn template. Select each view and write some text for the users to be displayed for each template. The text should be placed on the area marked red.



Step (8) : The users for the application are created by the developer. You might want to allow a visitor to create a user account. For this, add a link beneath the LoginView control, which should link to the CreateAccount.aspx page.

Step (9) : Place a CreateUserWizard control on the create account page. Set the ContinueDestinationPageUrl property of this control to Welcome.aspx.


asp:createuserwizard#CreateUserWiz...
Sign Up for Your New Account
User Name:
Password:
Confirm Password:
E-mail:
Security Question:
Security Answer:
The Password and Confirmation Password must match.
Create User
CreateUserWizard Tasks
Auto Format...
Step: Sign Up for Your New
Add/Remove WizardSteps...
Convert to StartNavigationTemplate
Convert to StepNavigationTemplate
Convert to FinishNavigationTemplate
Convert to CustomNavigationTemplate
Customize Create User Step
Customize Complete Step
Administer Website
Edit Templates

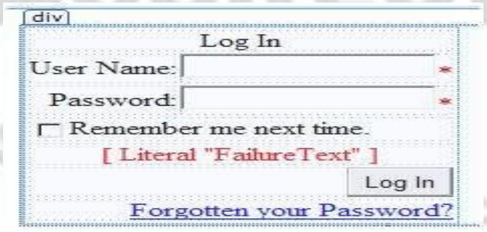
Step (10) : Create the Login page. Place a Login control on the page. The LoginStatus control automatically links to the Login.aspx. To change this default, make the following changes in the web.config file.

For example, if you want to name your log in page as signup.aspx, add the following lines to the <authentication> section of the web.config:

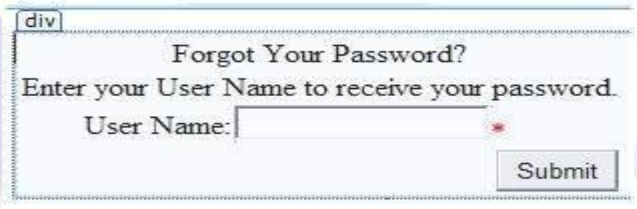
```
<configuration>
  <system.web>
    <authentication mode="Forms">
      <forms loginUrl ="signup.aspx" defaultUrl = â€œWelcome.aspxâ€ •  />
    </authentication>
  </system.web>
```

```
</configuration>
```

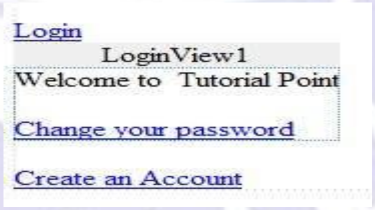
Step (11) : Users often forget passwords. The PasswordRecovery control helps the user gain access to the account. Select the Login control. Open its smart tag and click 'Convert to Template'.
Customize the UI of the control to place a hyperlink control under the login button, which should link to the PassWordRecovery.aspx



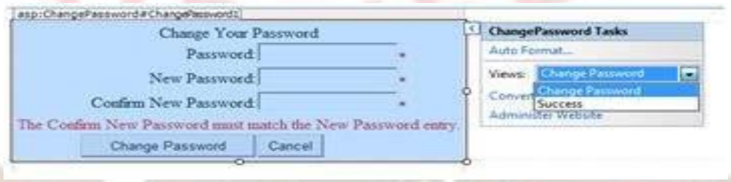
Step (12) : Place a PasswordRecovery control on the password recovery page. This control needs an email server to send the passwords to the users.



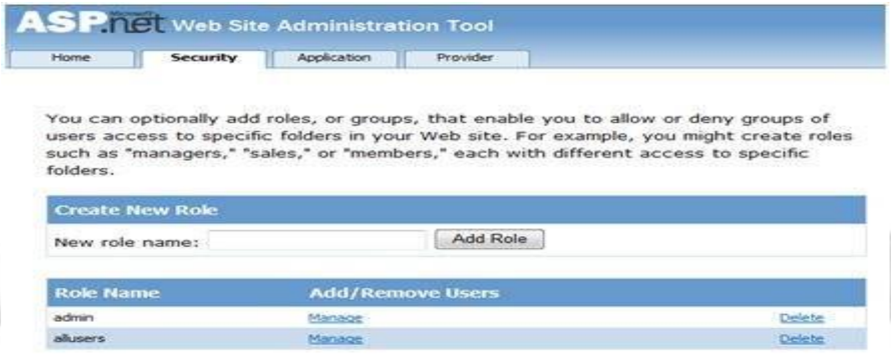
Step (13) : Create a link to the ChangePassword.aspx page in the LoggedIn template of the LoginView control in Welcome.aspx.



Step (14) : Place a ChangePassword control on the change password page. This control also has two views.



Now run the application and observe different security operations.
To create roles, go back to the Web Application Administration Tools and click on the Security tab. Click on 'Create Roles' and create some roles for the application.



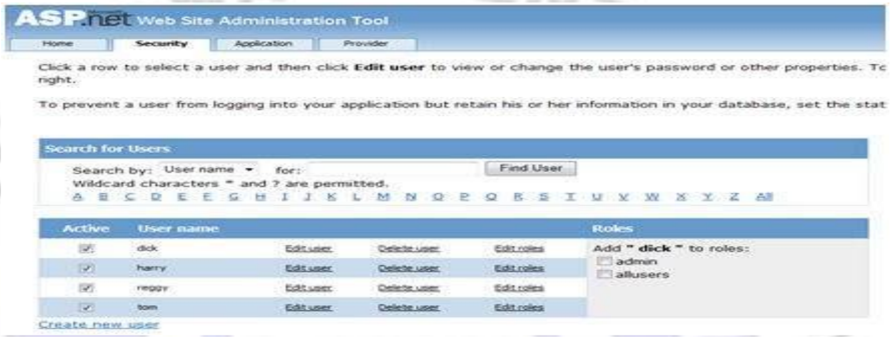
You can optionally add roles, or groups, that enable you to allow or deny groups of users access to specific folders in your Web site. For example, you might create roles such as "managers," "sales," or "members," each with different access to specific folders.

Create New Role

New role name: [Add Role]

Role Name	Add/Remove Users	
admin	Manage	Delete
allusers	Manage	Delete

Click on the 'Manage Users' link and assign roles to the users.



Click a row to select a user and then click **Edit user** to view or change the user's password or other properties. To right.

To prevent a user from logging into your application but retain his or her information in your database, set the stat

Search for Users

Search by: User name for: [Find User]
Wildcard characters * and ? are permitted.
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z All

Active	User name				Roles
✓	dick	Edit user	Delete user	Edit roles	Add " **dick** " to roles: ☐ admin ☐ allusers
✓	harry	Edit user	Delete user	Edit roles	
✓	reggy	Edit user	Delete user	Edit roles	
✓	tom	Edit user	Delete user	Edit roles	

Create new user

IIS Authentication: SSL

The Secure Socket Layer or SSL is the protocol used to ensure a secure connection. With SSL enabled, the browser encrypts all data sent to the server and decrypts all data coming from the server. At the same time, the server encrypts and decrypts all data to and from browser.

The URL for a secure connection starts with HTTPS instead of HTTP. A small lock is displayed by a browser using a secure connection. When a browser makes an initial attempt to communicate with a server over a secure connection using SSL, the server authenticates itself by sending its digital certificate.

To use the SSL, you need to buy a digital secure certificate from a trusted Certification Authority (CA) and install it in the web server. Following are some of the trusted and reputed certification authorities:

- www.verisign.com
- www.geotrust.com
- www.thawte.com

SSL is built into all major browsers and servers. To enable SSL, you need to install the digital certificate. The strength of various digital certificates varies depending upon the length of the key generated during encryption. More the length, more secure is the certificate, hence the connection.

Strength	Description
40 bit	Supported by most browsers but easy to break.
56 bit	Stronger than 40-bit.

128 bit	Extremely difficult to break but all the browsers do not support it.

ASP.NET - Data Caching

What is Caching?

Caching is a technique of storing frequently used data/information in memory, so that, when the same data/information is needed next time, it could be directly retrieved from the memory instead of being generated by the application.

Caching is extremely important for performance boosting in ASP.NET, as the pages and controls are dynamically generated here. It is especially important for data related transactions, as these are expensive in terms of response time.

Caching places frequently used data in quickly accessed media such as the random access memory of the computer. The ASP.NET runtime includes a key-value map of CLR objects called cache. This resides with the application and is available via the HttpContext and System.Web.UI.Page.

In some respect, caching is similar to storing the state objects. However, the storing information in state objects is deterministic, i.e., you can count on the data being stored there, and caching of data is nondeterministic.

The data will not be available in the following cases:

- If its lifetime expires,
- If the application releases its memory,
- If caching does not take place for some reason.

You can access items in the cache using an indexer and may control the lifetime of objects in the cache and set up links between the cached objects and their physical sources.

Caching in ASP.Net

ASP.NET provides the following different types of caching:

- **Output Caching** : Output cache stores a copy of the finally rendered HTML pages or part of pages sent to the client. When the next client requests for this page, instead of regenerating the page, a cached copy of the page is sent, thus saving time.
- **Data Caching** : Data caching means caching data from a data source. As long as the cache is not expired, a request for the data will be fulfilled from the cache. When the cache is expired, fresh data is obtained by the data source and the cache is refilled.
- **Object Caching** : Object caching is caching the objects on a page, such as data-bound controls. The cached data is stored in server memory.
- **Class Caching** : Web pages or web services are compiled into a page class in the assembly, when run for the first time. Then the assembly is cached in the server. Next time when a request is made

for the page or service, the cached assembly is referred to. When the source code is changed, the CLR recompiles the assembly.

- **Configuration Caching** : Application wide configuration information is stored in a configuration file. Configuration caching stores the configuration information in the server memory.

In this tutorial, we will consider output caching, data caching, and object caching.

Output Caching

Rendering a page may involve some complex processes such as, database access, rendering complex controls etc. Output caching allows bypassing the round trips to server by caching data in memory. Even the whole page could be cached.

The OutputCache directive is responsible of output caching. It enables output caching and provides certain control over its behaviour.

Syntax for OutputCache directive:

```
<%@ OutputCache Duration="15" VaryByParam="None" %>
```

Put this directive under the page directive. This tells the environment to cache the page for 15 seconds. The following event handler for page load would help in testing that the page was really cached.

```
protected void Page_Load(object sender, EventArgs e)
{
   Thread.Sleep(10000);
   Response.Write("This page was generated and cache at:" +
   DateTime.Now.ToString());
}
```

The **Thread.Sleep()** method stops the process thread for the specified time. In this example, the thread is stopped for 10 seconds, so when the page is loaded for first time, it takes 10 seconds. However, next time you refresh the page it does not take any time, as the page is retrieved from the cache without being loaded.

The OutputCache directive has the following attributes, which helps in controlling the behaviour of the output cache:

Attribute	**Values**	**Description**
DiskCacheable	true/false	Specifies that output could be written to a disk based cache.
NoStore	true/false	Specifies that the "no store" cache control header is sent or not.
CacheProfile	String name	Name of a cache profile as to be stored in web.config.

VaryByParam	None * Param- name	Semicolon delimited list of string specifies query string values in a GET request or variable in a POST request.
VaryByHeader	* Header names	Semicolon delimited list of strings specifies headers that might be submitted by a client.
VaryByCustom	Browser Custom string	Tells ASP.NET to vary the output cache by browser name and version or by a custom string.
Location	Any Client Downstream Server None	Any: page may be cached anywhere. Client: cached content remains at browser. Downstream: cached content stored in downstream and server both. Server: cached content saved only on server. None: disables caching.
Duration	Number	Number of seconds the page or control is cached.

Let us add a text box and a button to the previous example and add this event handler for the button.

```
protected void btnmagic_Click(object sender, EventArgs e)
{
   Response.Write("<br><br>");
   Response.Write("<h2> Hello, " + this.txtname.Text + "</h2>");
}
```

Change the OutputCache directive:

```
<%@ OutputCache Duration="60" VaryByParam="txtname" %>
```

When the program is executed, ASP.NET caches the page on the basis of the name in the text box.

Data Caching

The main aspect of data caching is caching the data source controls. We have already discussed that the data source controls represent data in a data source, like a database or an XML file. These controls derive from the abstract class DataSourceControl and have the following inherited properties for implementing caching:

- **CacheDuration** - It sets the number of seconds for which the data source will cache data.
- **CacheExpirationPolicy** - It defines the cache behavior when the data in cache has expired.
- **CacheKeyDependency** - It identifies a key for the controls that auto-expires the content of its cache when removed.
- **EnableCaching** - It specifies whether or not to cache the data.

Example

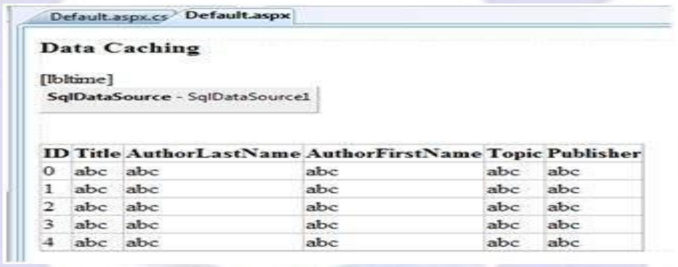
To demonstrate data caching, create a new website and add a new web form on it. Add a SqlDataSource control with the database connection already used in the data access tutorials.
For this example, add a label to the page, which would show the response time for the page.

```
<asp:Label ID="lbltime" runat="server"></asp:Label>
```

Apart from the label, the content page is same as in the data access tutorial. Add an event handler for the page load event:

```
protected void Page_Load(object sender, EventArgs e)
{
   lbltime.Text = String.Format("Page posted at: {0}", DateTime.Now.ToLongTimeString());
}
```

The designed page should look as shown:



Data Caching

[lbltime]

SqlDataSource - SqlDataSource1

ID	Title	AuthorLastName	AuthorFirstName	Topic	Publisher
0	abc	abc	abc	abc	abc
1	abc	abc	abc	abc	abc
2	abc	abc	abc	abc	abc
3	abc	abc	abc	abc	abc
4	abc	abc	abc	abc	abc

When you execute the page for the first time, nothing different happens, the label shows that, each time you refresh the page, the page is reloaded and the time shown on the label changes.
Next, set the EnableCaching attribute of the data source control to be 'true' and set the Cacheduration attribute to '60'. It will implement caching and the cache will expire every 60 seconds. The timestamp changes with every refresh, but if you change the data in the table within these 60 seconds, it is not shown before the cache expires.

```
<asp:SqlDataSource ID = "SqlDataSource1" runat = "server"
   ConnectionString = "<%$ ConnectionStrings: ASPDotNetStepByStepConnectionString %>"
   ProviderName = "<%$ ConnectionStrings:
ASPDotNetStepByStepConnectionString.ProviderName %>"
   SelectCommand = "SELECT * FROM [DotNetReferences]"
   EnableCaching = "true" CacheDuration = "60">
</asp:SqlDataSource>
```

Object Caching

Object caching provides more flexibility than other cache techniques. You can use object caching to place any object in the cache. The object can be of any type - a data type, a web control, a class, a dataset object, etc. The item is added to the cache simply by assigning a new key name, shown as follows Like:

```
Cache["key"] = item;
```

ASP.NET also provides the Insert() method for inserting an object to the cache. This method has four overloaded versions. Let us see them:

Overload	**Description**
Cache.Insert((key, value);	Inserts an item into the cache with the key name and value with default priority and expiration.
Cache.Insert(key, value, dependencies);	Inserts an item into the cache with key, value, default priority, expiration and a CacheDependency name that links to other files or items so that when these change the cache item remains no longer valid.
Cache.Insert(key, value, dependencies, absoluteExpiration, slidingExpiration);	This indicates an expiration policy along with the above issues.
Cache.Insert(key, value, dependencies, absoluteExpiration, slidingExpiration, priority, onRemoveCallback);	This along with the parameters also allows you to set a priority for the cache item and a delegate that, points to a method to be invoked when the item is removed.

Sliding expiration is used to remove an item from the cache when it is not used for the specified time span. The following code snippet stores an item with a sliding expiration of 10 minutes with no dependencies.

```
Cache.Insert("my_item", obj, null, DateTime.MaxValue, TimeSpan.FromMinutes(10));
```

Example

Create a page with just a button and a label. Write the following code in the page load event:

```
protected void Page_Load(object sender, EventArgs e)
{
   if (this.IsPostBack)
   {
      lblinfo.Text += "Page Posted Back.<br/>";
   }
   else
```

```
  {
     lblinfo.Text += "page Created.<br/>";
  }
     if (Cache["testitem"] == null)
  {
     lblinfo.Text += "Creating test item.<br/>";
     DateTime testItem = DateTime.Now;
     lblinfo.Text += "Storing test item in cache ";
     lblinfo.Text += "for 30 seconds.<br/>";
     Cache.Insert("testitem", testItem, null,
     DateTime.Now.AddSeconds(30), TimeSpan.Zero);
  }
  else
  {
     lblinfo.Text += "Retrieving test item.<br/>";
     DateTime testItem = (DateTime)Cache["testitem"];
     lblinfo.Text += "Test item is: " + testItem.ToString();
     lblinfo.Text += "<br/>";
  }
       lblinfo.Text += "<br/>";
}
```

When the page is loaded for the first time, it says:

```
Page Created.
Creating test item.
Storing test item in cache for 30 seconds.
```

If you click on the button again within 30 seconds, the page is posted back but the label control gets its information from the cache as shown:

```
Page Posted Back.
Retrieving test item.
Test item is: 14-07-2010 01:25:04
```

A web service is a web-based functionality accessed using the protocols of the web to be used by the web applications. There are three aspects of web service development:

- Creating the web service
- Creating a proxy
- Consuming the web service

Creating a Web Service

A web service is a web application which is basically a class consisting of methods that could be used by other applications. It also follows a code-behind architecture such as the ASP.NET web pages, although it does not have a user interface.

To understand the concept let us create a web service to provide stock price information. The clients can query about the name and price of a stock based on the stock symbol. To keep this example simple, the values are hardcoded in a two-dimensional array. This web service has three methods:

- A default HelloWorld method
- A GetName Method
- A GetPrice Method

Take the following steps to create the web service:

Step (1) : Select File -> New -> Web Site in Visual Studio, and then select ASP.NET Web Service.

Step (2) : A web service file called Service.asmx and its code behind file, Service.cs is created in the App_Code directory of the project.

Step (3) : Change the names of the files to StockService.asmx and StockService.cs.

Step (4) : The .asmx file has simply a WebService directive on it:

```
<%@ WebService Language="C#" CodeBehind="~/App_Code/StockService.cs"
Class="StockService" %>
```

Step (5) : Open the StockService.cs file, the code generated in it is the basic Hello World service. The default web service code behind file looks like the following:

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;
namespace StockService
{
   // <summary>
   // Summary description for Service1
   // <summary>
      [WebService(Namespace = "http://tempuri.org/")]
   [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
   [ToolboxItem(false)]
      // To allow this Web Service to be called from script,
   // using ASP.NET AJAX, uncomment the following line.
   // [System.Web.Script.Services.ScriptService]
      public class Service1 : System.Web.Services.WebService
   {
      [WebMethod]
            public string HelloWorld()
      {
         return "Hello World";
      }
   }
}
```

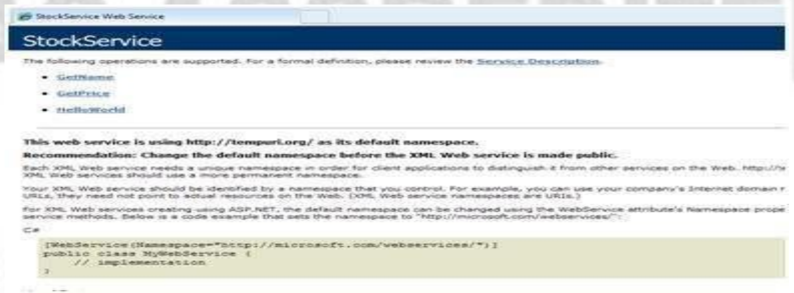
Step (6) : Change the code behind file to add the two dimensional array of strings for stock symbol, name and price and two web methods for getting the stock information.

```
using System;
using System.Linq;
using System.Web;
using System.Web.Services;
using System.Web.Services.Protocols;
using System.Xml.Linq;
[WebService(Namespace = "http://tempuri.org/")]
[WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
// To allow this Web Service to be called from script,
// using ASP.NET AJAX, uncomment the following line.
// [System.Web.Script.Services.ScriptService]
public class StockService : System.Web.Services.WebService
{
   public StockService () {
      //Uncomment the following if using designed components
```

```
  //InitializeComponent();
}
  string[,] stocks =
{
  {"RELIND", "Reliance Industries", "1060.15"},
  {"ICICI", "ICICI Bank", "911.55"},
  {"JSW", "JSW Steel", "1201.25"},
  {"WIPRO", "Wipro Limited", "1194.65"},
  {"SATYAM", "Satyam Computers", "91.10"}
};
[WebMethod]
public string HelloWorld() {
  return "Hello World";
}
  [WebMethod]
public double GetPrice(string symbol)
{
  //it takes the symbol as parameter and returns price
  for (int i = 0; i < stocks.GetLength(0); i++)
  {
    if (String.Compare(symbol, stocks[i, 0], true) == 0)
    return Convert.ToDouble(stocks[i, 2]);
  }
      return 0;
}
  [WebMethod]
public string GetName(string symbol)
{
  // It takes the symbol as parameter and
  // returns name of the stock
  for (int i = 0; i < stocks.GetLength(0); i++)
  {
    if (String.Compare(symbol, stocks[i, 0], true) == 0)
    return stocks[i, 1];
  }
      return "Stock Not Found";
```

```
   }
}
```

Step (7) : Running the web service application gives a web service test page, which allows testing the service methods.



StockService Web Service

StockService

The following operations are supported. For a formal definition, please review the Service Description.

- GetName
- GetPrice
- HelloWorld

This web service is using http://tempuri.org/ as its default namespace.

Recommendation: Change the default namespace before the XML Web service is made public.

Each XML Web service needs a unique namespace in order for client applications to distinguish it from other services on the Web. http://te XML Web services should use a more permanent namespace.

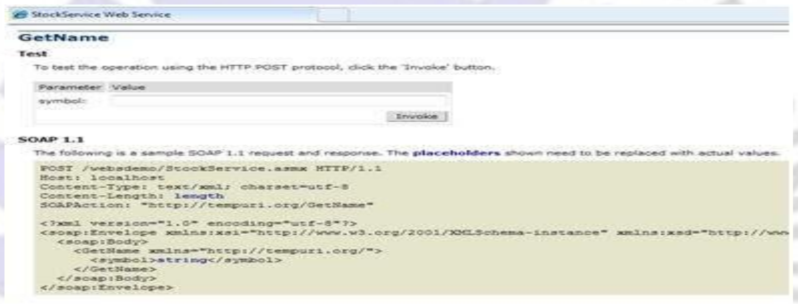
Your XML Web service should be identified by a namespace that you control. For example, you can use your company's Internet domain n URLs, they need not point to actual resources on the Web. (XML Web service namespaces are URIs.)

For XML Web services creating using ASP.NET, the default namespace can be changed using the WebService attribute's Namespace prope service methods. Below is a code example that sets the namespace to "http://microsoft.com/webservices/":

C#

```
[WebService(Namespace="http://microsoft.com/webservices/")]
public class MyWebService {
    // implementation
}
```

Step (8) : Click on a method name, and check whether it runs properly.



StockService Web Service

GetName

Test

To test the operation using the HTTP POST protocol, click the 'Invoke' button.

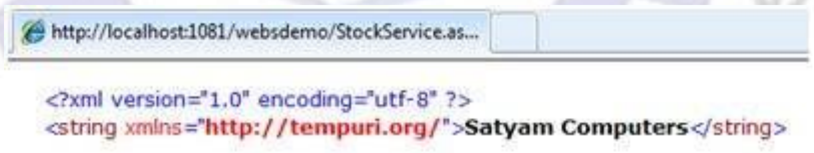
Parameter	Value
symbol:	

SOAP 1.1

The following is a sample SOAP 1.1 request and response. The **placeholders** shown need to be replaced with actual values.

```
POST /websdemo/StockService.asmx HTTP/1.1
Host: localhost
Content-Type: text/xml; charset=utf-8
Content-Length: length
SOAPAction: "http://tempuri.org/GetName"

<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www
  <soap:Body>
    <GetName xmlns="http://tempuri.org/">
      <symbol>string</symbol>
    </GetName>
  </soap:Body>
</soap:Envelope>
```

Step (9) : For testing the GetName method, provide one of the stock symbols, which are hard coded, it returns the name of the stock



```
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">Satyam Computers</string>
```

Consuming the Web Service

For using the web service, create a web site under the same solution. This could be done by right clicking on the Solution name in the Solution Explorer. The web page calling the web service should have a label control to display the returned results and two button controls one for post back and another for calling the service.

The content file for the web application is as follows:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="wsclient._Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
  <head runat="server">
    <title>
      Untitled Page
      </title>
   </head>
      <body>
         <form id="form1" runat="server">
         <div>


            <h3>Using the Stock Service</h3>
                        <br /> <br />
                        <asp:Label ID="lblmessage" runat="server"></asp:Label>
                        <br /> <br />
                        <asp:Button ID="btnpostback" runat="server" onclick="Button1_Click" Text="Post
Back" style="width:132px" />
                                    <asp:Button  ID="btnservice"  runat="server"  onclick="btnservice_Click"
Text="Get Stock" style="width:99px" />
                      </div>
      </form>
         </body>
</html>
```

The code behind file for the web application is as follows:

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
//this is the proxy
using localhost;
namespace wsclient
{
  public partial class _Default : System.Web.UI.Page
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      if (!IsPostBack)
      {
        lblmessage.Text = "First Loading Time: " +  DateTime.Now.ToLongTimeString
      }
      else
      {
        lblmessage.Text = "PostBack at: " + DateTime.Now.ToLongTimeString();
      }
    }
        protected void btnservice_Click(object sender, EventArgs e)
    {
      StockService proxy = new StockService();
      lblmessage.Text = String.Format("Current SATYAM Price:{0}",
      proxy.GetPrice("SATYAM").ToString());
    }
  }
}
```

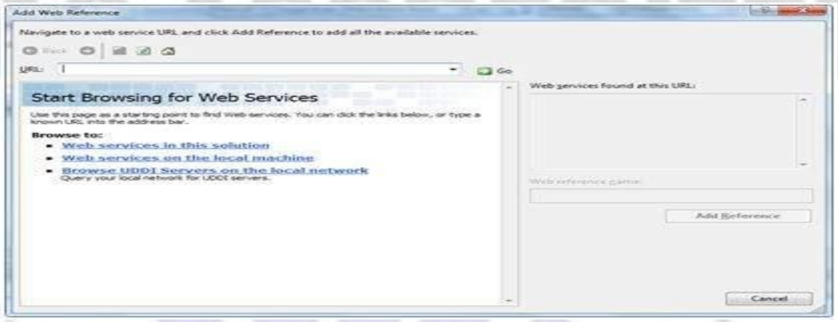
Creating the Proxy

A proxy is a stand-in for the web service codes. Before using the web service, a proxy must be created. The proxy is registered with the client application. Then the client application makes the calls to the web service as it were using a local method. The proxy takes the calls, wraps it in proper format and sends it as a SOAP request to the server. SOAP stands for Simple Object Access Protocol. This protocol is used for exchanging web service data. When the server returns the SOAP package to the client, the proxy decodes everything and presents it to the client application. Before calling the web service using the btnservice_Click, a web reference should be added to the application. This creates a proxy class transparently, which is used by the btnservice_Click event.

```
protected void btnservice_Click(object sender, EventArgs e)
{
  StockService proxy = new StockService();
  lblmessage.Text = String.Format("Current SATYAM Price: {0}",
  proxy.GetPrice("SATYAM").ToString());
}
```

Take the following steps for creating the proxy:

Step (1) : Right click on the web application entry in the Solution Explorer and click on 'Add Web Reference'.



Navigate to a web service URL and click Add Reference to add all the available services.

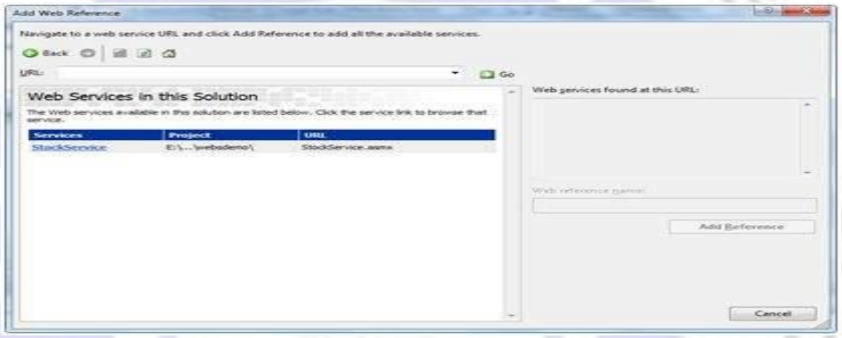
Start Browsing for Web Services

Use this page as a starting point to find Web services. You can click the links below, or type a known URL into the address bar.

Browse to:

- Web services in this solution
- Web services on the local machine
- Browse UDDI Servers on the local network
  Query your local network for UDDI servers.

Step (2) : Select 'Web Services in this solution'. It returns the StockService reference.



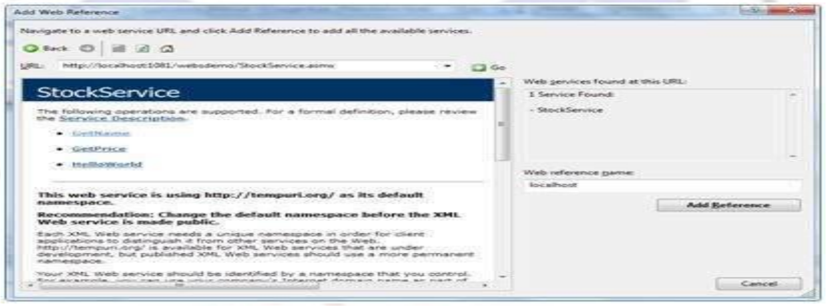
Navigate to a web service URL and click Add Reference to add all the available services.

Web Services in this Solution

The Web services available in this solution are listed below. Click the service link to browse that service.

Services	Project	URL
StockService	E:\...\webdemo\	StockService.asmx

Step (3) : Clicking on the service opens the test web page. By default the proxy created is called 'localhost', you can rename it. Click on 'Add Reference' to add the proxy to the client application.



Navigate to a web service URL and click Add Reference to add all the available services.

URL: http://localhost:1081/webdemo/StockService.asmx

StockService

The following operations are supported. For a formal definition, please review the Service Description.

- GetName
- GetPrice
- HelloWorld

This web service is using http://tempuri.org/ as its default namespace.

Recommendation: Change the default namespace before the XML Web service is made public.

Each XML Web service needs a unique namespace in order for client applications to distinguish it from other services on the Web. http://tempuri.org/ is available for XML Web services that are under development, but published XML Web services should use a more permanent namespace.

Your XML Web service should be identified by a namespace that you control.

1 Service Found:
- StockService

Web reference name: localhost

Include the proxy in the code behind file by adding:

```
using localhost;
```

ASP.NET - Multi Threading

A thread is defined as the execution path of a program. Each thread defines a unique flow of control. If your application involves complicated and time consuming operations such as database access or some intense I/O operations, then it is often helpful to set different execution paths or threads, with each thread performing a particular job.

Threads are lightweight processes. One common example of use of thread is implementation of concurrent programming by modern operating systems. Use of threads saves wastage of CPU cycle and increases efficiency of an application.

So far we compiled programs where a single thread runs as a single process which is the running instance of the application. However, this way the application can perform one job at a time. To make it execute multiple tasks at a time, it could be divided into smaller threads.

In .Net, the threading is handled through the 'System.Threading' namespace. Creating a variable of the *System.Threading.Thread* type allows you to create a new thread to start working with. It allows you to create and access individual threads in a program.

Creating Thread

A thread is created by creating a Thread object, giving its constructor a ThreadStart reference.

```
ThreadStart childthreat = new ThreadStart(childthreadcall);
```

Thread Life Cycle

The life cycle of a thread starts when an object of the System.Threading.Thread class is created and ends when the thread is terminated or completes execution.

Following are the various states in the life cycle of a thread:

- **The Unstarted State** : It is the situation when the instance of the thread is created but the Start method is not called.
- **The Ready State** : It is the situation when the thread is ready to execute and waiting CPU cycle.
- **The Not Runnable State** : a thread is not runnable, when:
  - Sleep method has been called
  - Wait method has been called
  - Blocked by I/O operations
- **The Dead State** : It is the situation when the thread has completed execution or has been aborted.

Thread Priority

The Priority property of the Thread class specifies the priority of one thread with respect to other. The .Net runtime selects the ready thread with the highest priority.

The priorities could be categorized as:

- Above normal
- Below normal
- Highest
- Lowest
- Normal

Once a thread is created, its priority is set using the Priority property of the thread class.

```
NewThread.Priority = ThreadPriority.Highest;
```

Thread Properties & Methods

The Thread class has the following important properties:

Property	Description
CurrentContext	Gets the current context in which the thread is executing.
CurrentCulture	Gets or sets the culture for the current thread.
CurrentPrinciple	Gets or sets the thread's current principal for role-based security.
CurrentThread	Gets the currently running thread.
CurrentUICulture	Gets or sets the current culture used by the Resource Manager to look up culture-specific resources at run time.
ExecutionContext	Gets an ExecutionContext object that contains information about the various contexts of the current thread.
IsAlive	Gets a value indicating the execution status of the current thread.
IsBackground	Gets or sets a value indicating whether or not a thread is a background thread.
IsThreadPoolThread	Gets a value indicating whether or not a thread belongs to the managed thread pool.
ManagedThreadId	Gets a unique identifier for the current managed thread.
Name	Gets or sets the name of the thread.
Priority	Gets or sets a value indicating the scheduling priority of a thread.
ThreadState	Gets a value containing the states of the current thread.

The Thread class has the following important methods:

Methods	Description
Abort	Raises a ThreadAbortException in the thread on which it is invoked, to begin the process of terminating the thread. Calling this method usually terminates the thread.
AllocateDataSlot	Allocates an unnamed data slot on all the threads. For better performance, use fields that are marked with the ThreadStaticAttribute

	attribute instead.
AllocateNamedDataSlot	Allocates a named data slot on all threads. For better performance, use fields that are marked with the ThreadStaticAttribute attribute instead.
BeginCriticalRegion	Notifies a host that execution is about to enter a region of code in which the effects of a thread abort or unhandled exception might endanger other tasks in the application domain.
BeginThreadAffinity	Notifies a host that managed code is about to execute instructions that depend on the identity of the current physical operating system thread.
EndCriticalRegion	Notifies a host that execution is about to enter a region of code in which the effects of a thread abort or unhandled exception are limited to the current task.
EndThreadAffinity	Notifies a host that managed code has finished executing instructions that depend on the identity of the current physical operating system thread.
FreeNamedDataSlot	Eliminates the association between a name and a slot, for all threads in the process. For better performance, use fields that are marked with the ThreadStaticAttribute attribute instead.
GetData	Retrieves the value from the specified slot on the current thread, within the current thread's current domain. For better performance, use fields that are marked with the ThreadStaticAttribute attribute instead.
GetDomain	Returns the current domain in which the current thread is running.
GetDomainID	Returns a unique application domain identifier.
GetNamedDataSlot	Looks up a named data slot. For better performance, use fields that are marked with the ThreadStaticAttribute attribute instead.
Interrupt	Interrupts a thread that is in the WaitSleepJoin thread state.
Join	Blocks the calling thread until a thread terminates, while continuing to perform standard COM and SendMessage pumping. This method has different overloaded forms.
MemoryBarrier	Synchronizes memory access as follows: The

	processor executing the current thread cannot reorder instructions in such a way that memory accesses prior to the call to MemoryBarrier execute after memory accesses that follow the call to MemoryBarrier.
ResetAbort	Cancels an Abort requested for the current thread.
SetData	Sets the data in the specified slot on the currently running thread, for that thread's current domain. For better performance, use fields marked with the ThreadStaticAttribute attribute instead.
Start	Starts a thread.
Sleep	Makes the thread pause for a period of time.
SpinWait	Causes a thread to wait the number of times defined by the iterations parameter.
VolatileRead()	Reads the value of a field. The value is the latest written by any processor in a computer, regardless of the number of processors or the state of processor cache. This method has different overloaded forms.
VolatileWrite()	Writes a value to a field immediately, so that the value is visible to all processors in the computer. This method has different overloaded forms.
Yield	Causes the calling thread to yield execution to another thread that is ready to run on the current processor. The operating system selects the thread to yield to.

Example

The following example illustrates the uses of the Thread class. The page has a label control for displaying messages from the child thread. The messages from the main program are directly displayed using the Response.Write() method. Hence they appear on the top of the page.

The source file is as follows:

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="threaddemo._Default" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" >
```

```
  <head runat="server">
    <title>
      Untitled Page
    </title>
  </head>
    <body>
    <form id="form1" runat="server">
      <div>
        <h3>Thread Example</h3>
      </div>



     <asp:Label ID="lblmessage" runat="server" Text="Label">
      </asp:Label>
    </form>
  </body>
  </html>
```

The code behind file is as follows:

```
using System;
using System.Collections;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.HtmlControls;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Xml.Linq;
using System.Threading;
namespace threaddemo
{
  public partial class _Default : System.Web.UI.Page
  {
      protected void Page_Load(object sender, EventArgs e)
      {
        ThreadStart childthreat = new ThreadStart(childthreadcall);
        Response.Write("Child Thread Started <br/>");
        Thread child = new Thread(childthreat);
              child.Start();
              Response.Write("Main sleeping  for 2 seconds.......<br/>");
        Thread.Sleep(2000);
        Response.Write("<br/>Main aborting child thread<br/>");
              child.Abort();
      }

          public void childthreadcall()
      {
        try{
          lblmessage.Text = "<br />Child thread started <br/>";
          lblmessage.Text += "Child Thread: Coiunting to 10";
                  for( int i =0; i<10; i++)
          {
            Thread.Sleep(500);
            lblmessage.Text += "<br/> in Child thread </br>";
          }
                  lblmessage.Text += "<br/> child thread finished";
                }catch(ThreadAbortException e){
                lblmessage.Text += "<br /> child thread - exception";
                }finally{
          lblmessage.Text += "<br /> child thread - unable to catch the  exception";
        }
      }
    }
}
```

Observe the following

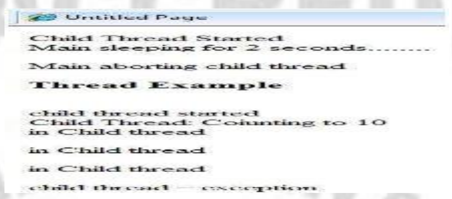
- ~~When the page is loaded, a new thread is started with the reference of the method childthreadcall().~~

The main thread activities are displayed directly on the web page.

- ~~The second thread runs and sends messages to the label control.~~

- The main thread sleeps for 2000 ms, during which the child thread executes.
- The child thread runs till it is aborted by the main thread. It raises the ThreadAbortException and is terminated.
- Control returns to the main thread.

When executed the program sends the following messages:



```
Untitled Page

Child Thread Started
Main sleeping for 2 seconds.........
Main aborting child thread
Thread Example

child thread started
Child Thread: Coiunting to 10
in Child thread
in Child thread
in Child thread
child thread -- exception
```

ASP.NET - Configuration

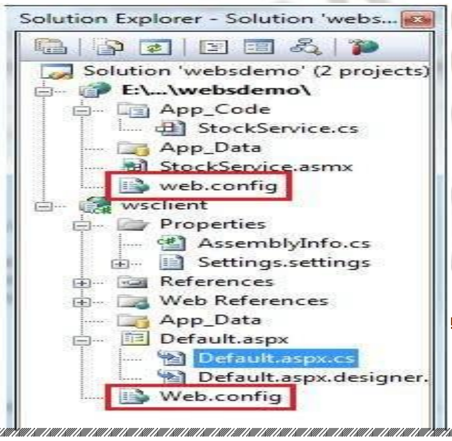
The behavior of an ASP.NET application is affected by different settings in the configuration files:

- machine.config
- web.config

The machine.config file contains default and the machine-specific value for all supported settings. The machine settings are controlled by the system administrator and applications are generally not given access to this file. An application however, can override the default values by creating web.config files in its roots folder. The web.config file is a subset of the machine.config file. If the application contains child directories, it can define a web.config file for each folder. Scope of each configuration file is determined in a hierarchical top-down manner.

Any web.config file can locally extend, restrict, or override any settings defined on the upper level. Visual Studio generates a default web.config file for each project. An application can execute without a web.config file, however, you cannot debug an application without a web.config file.

The following figure shows the Solution Explorer for the sample example used in the web services tutorial:



```
Solution Explorer - Solution 'webs...
Solution 'websdemo' (2 projects)
  E:\...\websdemo\
    App_Code
      StockService.cs
    App_Data
    StockService.asmx
    web.config
  wsclient
    Properties
      AssemblyInfo.cs
      Settings.settings
    References
    Web References
    App_Data
    Default.aspx
      Default.aspx.cs
      Default.aspx.designer.
    Web.config
```

In this application, there are two web.config files for two projects i.e., the web service and the web site calling the web service. The web.config file has the configuration element as the root node. Information inside this element is grouped into two main areas: the configuration section-handler declaration area, and the configuration section settings area.

The following code snippet shows the basic syntax of a configuration file:

```
<configuration>
  <!-- Configuration section-handler declaration area. -->
    <configSections>
      <section name="section1" type="section1Handler" />
      <section name="section2" type="section2Handler" />
    </configSections>
  <!-- Configuration section settings area. -->
    <section1>
    <s1Setting1 attribute1="attr1" />
  </section1>
    <section2>
    <s2Setting1 attribute1="attr1" />
  </section2>
    <system.web>
    <authentication mode="Windows" />
  </system.web>
  </configuration>
```

Configuration Section Handler declarations

The configuration section handlers are contained within the <configSections> tags. Each configuration handler specifies name of a configuration section, contained within the file, which provides some configuration data. It has the following basic syntax:

```
<configSections>
  <section />
  <sectionGroup />
  <remove />
  <clear/>
</configSections>
```

It has the following elements:

- **Clear** - It removes all references to inherited sections and section groups.
- **Remove** - It removes a reference to an inherited section and section group.

- **Section** - It defines an association between a configuration section handler and a configuration element.
- **Section group** - It defines an association between a configuration section handler and a configuration section.

Application Settings

The application settings allow storing application-wide name-value pairs for read-only access. For example, you can define a custom application setting as:

```
<configuration>
   <appSettings>
      <add key="Application Name" value="MyApplication" />
   </appSettings>
</configuration>
```

For example, you can also store the name of a book and its ISBN number:

```
<configuration>
   <appSettings>
      <add key="appISBN" value="0-273-68726-3" />
      <add key="appBook" value="Corporate Finance" />
   </appSettings>
</configuration>
```

Connection Strings

The connection strings show which database connection strings are available to the website. For example:

```
<connectionStrings>
   <add name="ASPDotNetStepByStepConnectionString"
      connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
      Data Source=E:\\projects\datacaching\ /
      datacaching\App_Data\ASPDotNetStepByStep.mdb"
      providerName="System.Data.OleDb" />
         <add name="booksConnectionString"
      connectionString="Provider=Microsoft.Jet.OLEDB.4.0;
      Data Source=C:\ \databinding\App_Data\books.mdb"
      providerName="System.Data.OleDb" />
</connectionStrings>
```

System.Web Element

The system.web element specifies the root element for the ASP.NET configuration section and contains configuration elements that configure ASP.NET Web applications and control how the applications behave.

It holds most of the configuration elements needed to be adjusted in common applications. The basic syntax for the element is as given:

```
<system.web>
   <anonymousIdentification>
   <authentication>
   <authorization>
   <browserCaps>
   <caching>



 <clientTarget>
   <compilation>
   <customErrors>
   <deployment>
   <deviceFilters>
   <globalization>
   <healthMonitoring>
   <hostingEnvironment>
   <httpCookies>
   <httpHandlers>
   <httpModules>
   <httpRuntime>
   <identity>
   <machineKey>
   <membership>
   <mobileControls>
   <pages>
   <processModel>
   <profile>
   <roleManager>
   <securityPolicy>
   <sessionPageState>
```

```
  <sessionState>
  <siteMap>
  <trace>
  <trust>
  <urlMappings>
  <webControls>
  <webParts>
  <webServices>
  <xhtmlConformance>
</system.web>
```

The following table provides brief description of some of common sub elements of the **system.web** element:

AnonymousIdentification

This is required to identify users who are not authenticated when authorization is required.

Authentication

It configures the authentication support. The basic syntax is as given:

```
<authentication mode="[Windows|Forms|Passport|None]">
  <forms>...</forms>
  <passport/>
</authentication>
```

Authorization

It configures the authorization support. The basic syntax is as given:

```
<authorization>
  <allow .../>
  <deny .../>
</authorization>
```

Caching

It Configures the cache settings. The basic syntax is as given:

```
<caching>
```

```
   <cache>...</cache>
   <outputCache>...</outputCache>
   <outputCacheSettings>...</outputCacheSettings>
   <sqlCacheDependency>...</sqlCacheDependency>
</caching>
```

CustomErrors

It defines custom error messages. The basic syntax is as given:

```
<customErrors defaultRedirect="url" mode="On|Off|RemoteOnly">
   <error. . ./>
</customErrors>
```

Deployment

It defines configuration settings used for deployment. The basic syntax is as follows:

```
<deployment retail="true|false" />
```

HostingEnvironment

It defines configuration settings for hosting environment. The basic syntax is as follows:

```
<hostingEnvironment idleTimeout="HH:MM:SS" shadowCopyBinAssemblies="true|false"
   shutdownTimeout="number" urlMetadataSlidingExpiration="HH:MM:SS" />
```

Identity

It configures the identity of the application. The basic syntax is as given:

```
<identity impersonate="true|false" userName="domain\username"
   password="<secure password>"/>
```

MachineKey

It configures keys to use for encryption and decryption of Forms authentication cookie data. It also allows configuring a validation key that performs message authentication checks on view-state data and forms authentication tickets. The basic syntax is:

```
<machineKey validationKey="AutoGenerate,IsolateApps" [String]
   decryptionKey="AutoGenerate,IsolateApps" [String]
   validation="HMACSHA256" [SHA1 | MD5 | 3DES | AES | HMACSHA256 |
   HMACSHA384 | HMACSHA512 | alg:algorithm_name]
   decryption="Auto" [Auto | DES | 3DES | AES | alg:algorithm_name]
/>
```

Membership

This configures parameters of managing and authenticating user accounts. The basic syntax is:

```
<membership defaultProvider="provider name"
   userIsOnlineTimeWindow="number of minutes" hashAlgorithmType="SHA1">
   <providers>...</providers>
</membership>
```

Pages

It provides page-specific configurations. The basic syntax is:

```
<pages asyncTimeout="number" autoEventWireup="[True|False]"
     buffer="[True|False]" clientIDMode="[AutoID|Predictable|Static]"
     compilationMode="[Always|Auto|Never]"
     controlRenderingCompatibilityVersion="[3.5|4.0]"
     enableEventValidation="[True|False]"
     enableSessionState="[True|False|ReadOnly]"
     enableViewState="[True|False]"
     enableViewStateMac="[True|False]"
     maintainScrollPositionOnPostBack="[True|False]"
     masterPageFile="file path"
     maxPageStateFieldLength="number"
     pageBaseType="typename, assembly"
     pageParserFilterType="string"
     smartNavigation="[True|False]"
     styleSheetTheme="string"
     theme="string"
     userControlBaseType="typename"
     validateRequest="[True|False]"
     viewStateEncryptionMode="[Always|Auto|Never]" >
     <controls>...</controls>
```

```
  <namespaces>...</namespaces>
  <tagMapping>...</tagMapping>
  <ignoreDeviceFilters>...</ignoreDeviceFilters>
</pages>
```

Profile

It configures user profile parameters. The basic syntax is:

```
<profile enabled="true|false" inherits="fully qualified type reference"
  automaticSaveEnabled="true|false" defaultProvider="provider name">
    <properties>...</properties>
  <providers>...</providers>
  </profile>
```

RoleManager

It configures settings for user roles. The basic syntax is:

```
<roleManager cacheRolesInCookie="true|false" cookieName="name"
  cookiePath="/" cookieProtection="All|Encryption|Validation|None"
  cookieRequireSSL="true|false " cookieSlidingExpiration="true|false "
  cookieTimeout="number of minutes" createPersistentCookie="true|false"
  defaultProvider="provider name" domain="cookie domain">
  enabled="true|false"
  maxCachedResults="maximum number of role names cached"
    <providers>...</providers>
</roleManager>
```

SecurityPolicy

It configures the security policy. The basic syntax is:

```
<securityPolicy>
  <trustLevel />
</securityPolicy>
```

UrlMappings

It defines mappings to hide the original URL and provide a more user friendly URL. The basic syntax is:

```
<urlMappings enabled="true|false">
   <add.../>
   <clear />
   <remove.../>
</urlMappings>
```

WebControls

It provides the name of shared location for client scripts. The basic syntax is:

```
<webControls clientScriptsLocation="String" />
```

WebServices

This configures the web services.

ASP.NET - Deployment

There are two categories of ASP.NET deployment:

- **Local deployment** : In this case, the entire application is contained within a virtual directory and all the contents and assemblies are contained within it and available to the application.
- **Global deployment** : In this case, assemblies are available to every application running on the server.

  There are different techniques used for deployment, however, we will discuss the following most common and easiest ways of deployment:
- XCOPY deployment
- Copying a Website
- Creating a set up project

  XCOPY Deployment

  XCOPY deployment means making recursive copies of all the files to the target folder on the target machine. You can use any of the commonly used techniques:
- FTP transfer
- Using Server management tools that provide replication on a remote site
- MSI installer application

XCOPY deployment simply copies the application file to the production server and sets a virtual directory there. You need to set a virtual directory using the Internet Information Manager Microsoft Management Console (MMC snap-in).

Copying a Website

The Copy Web Site option is available in Visual Studio. It is available from the Website -> Copy Web Site menu option. This menu item allows copying the current web site to another local or remote location. It is a sort of integrated FTP tool.

Using this option, you connect to the target destination, select the desired copy mode:

- Overwrite
- Source to Target Files
- Sync UP Source And Target Projects

Then proceed with copying the files physically. Unlike the XCOPY deployment, this process of deployment is done from Visual Studio environment. However, there are following problems with both the above deployment methods:

- You pass on your source code.
- There is no pre-compilation and related error checking for the files.
- The initial page load will be slow.

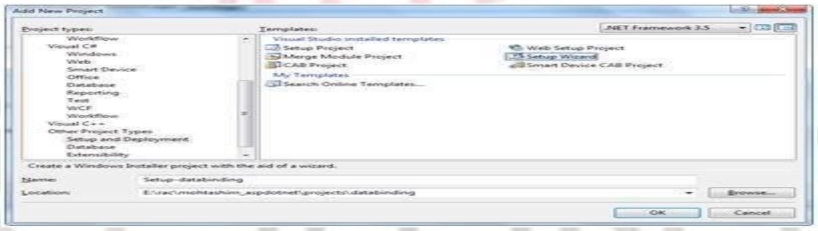
Creating a Setup Project

In this method, you use Windows Installer and package your web applications so it is ready to deploy on the production server. Visual Studio allows you to build deployment packages. Let us test this on one of our existing project, say the data binding project.

Open the project and take the following steps:

Step (1) : Select File -> Add -> New Project with the website root directory highlighted in the Solution Explorer.

Step (2) : Select Setup and Deployment, under Other Project Types. Select Setup Wizard.



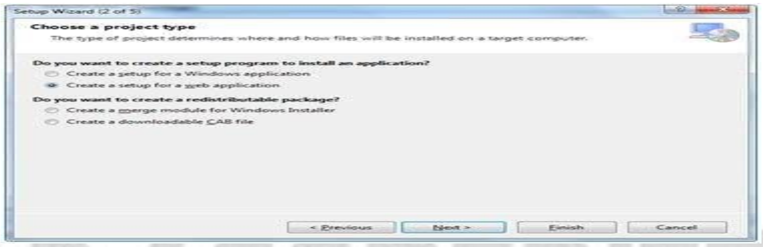
Create a Windows Installer project with the aid of a wizard.

Name: Setup-databinding

Location: E:\rac\mohtashim_aspdotnet\projects\databinding

Step (3) : Choosing the default location ensures that the set up project will be located in its own folder under the root directory of the site. Click on okay to get the first splash screen of the wizard.

Step (4) : Choose a project type. Select 'Create a setup for a web application'.



Choose a project type

The type of project determines where and how files will be installed on a target computer.

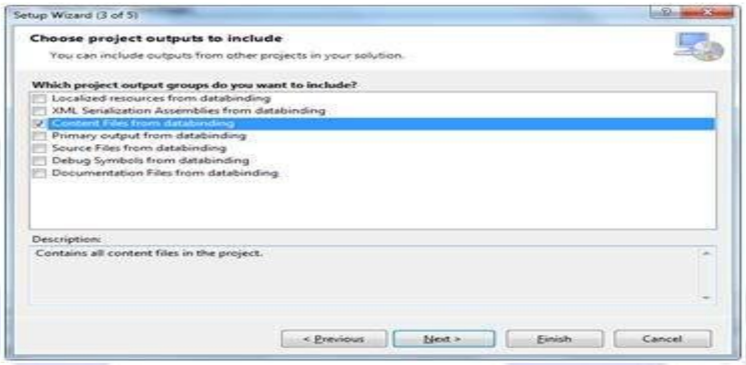
Do you want to create a setup program to install an application?

- Create a setup for a Windows application
- [✓ Create a setup for a web application]

Do you want to create a redistributable package?

- Create a merge module for Windows Installer
- Create a downloadable CAB file

Step (5) : Next, the third screen asks to choose project outputs from all the projects in the solution. Check the check box next to 'Content Files from...'



Choose project outputs to include

You can include outputs from other projects in your solution.

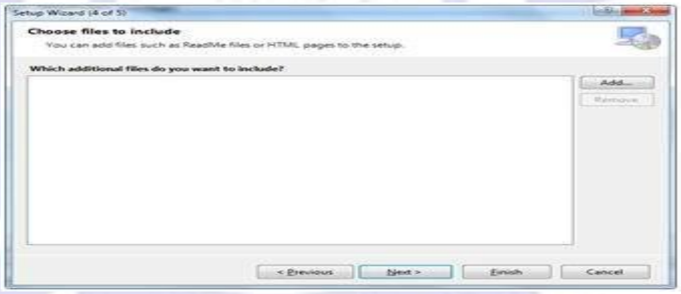
Which project output groups do you want to include?

- Localized resources from databinding
- XML Serialization Assemblies from databinding
- [✓ Content Files from databinding]
- Primary output from databinding
- Source Files from databinding
- Debug Symbols from databinding
- Documentation Files from databinding

Description:

Contains all content files in the project.

Step (6) : The fourth screen allows including other files like ReadMe. However, in our case there is no such file. Click on finish.

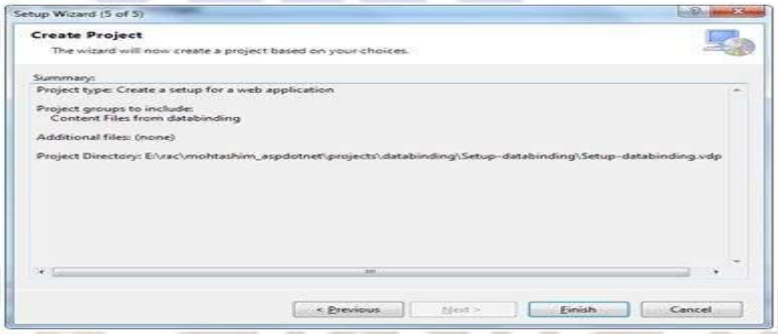


Choose files to include

You can add files such as ReadMe files or HTML pages to the setup.

Which additional files do you want to include?

Step (7) : The final screen displays a summary of settings for the set up project.



Create Project

The wizard will now create a project based on your choices.

Summary:

Project type: Create a setup for a web application

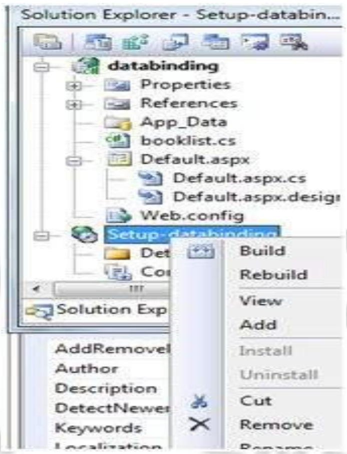
Project groups to include:
  Content Files from databinding

Additional files: (none)

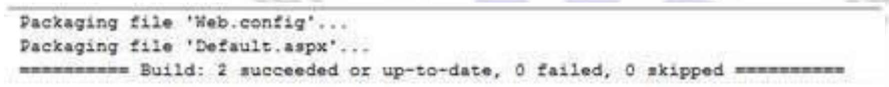
Project Directory: E:\rac\mohtashim_aspdotnet\projects\databinding\Setup-databinding\Setup-databinding.vdp

Step (8) : The Set up project is added to the Solution Explorer and the main design window shows a file system editor.

Step (9) : Next step is to build the setup project. Right click on the project name in the Solution Explorer and select Build.



Step (10) : When build is completed, you get the following message in the Output window:



```
Packaging file 'Web.config'...
Packaging file 'Default.aspx'...
========== Build: 2 succeeded or up-to-date, 0 failed, 0 skipped ==========
```

Two files are created by the build process:

- Setup.exe
- Setup-databinding.msi

You need to copy these files to the server. Double-click the setup file to install the content of the .msi file on the local machin

PRINCIPAL OF ACCOUNTING (207)

MEANING AND SCOPE OF ACCOUNTING

DEFINITION

Financial accountancy (or financial accounting) is the field of accountancy concerned with the preparation of financial statements for decision makers, such as stockholders, suppliers, banks, employees, government agencies, owners and other stakeholders. In 1966, American accounting Association (AAA) defined accounting as "Accounting is the process of identifying, measuring and communicating economic information to permit the informed judgements ands decisions by the users of the information."

In 1941, American Institute of Certified public Accountants (AICPA) defined accounting as "Accounting is the art of recording, classifying and summarizing in significant manner and in terms of money, transactions and events which are, in part, at least of a financial character and interpreting the results thereof." Financial capital maintenance can be measured in either nominal monetary units or units of constant purchasing power. The central need for financial accounting is

to reduce the various principal-agent problems, by measuring and monitoring the agents' performance and thereafter reporting the results to interested users.

SCOPE OF ACCOUNTING:

Accounting as compared to book-keeping has a very wide and huge role to play in the businesses whether there is a small firm engaged in few transactions to a large MNC with multiple transactions daily, Accounting is required everywhere. Accounting not only maintains the records but also analyzes and interprets the results also. Accounting is being done in any nature and size of the firm, example: schools, hospitals, banks, retail shops. Nowadays Accountants are serving the accounting requirements of various businesses and organizations for which they require expertise in the same field. Accounting has grown in its importance today as a discipline that provides results online in a quick and accessible form that can be used by the management for decision-making. It has been recognized as a tool for mastering the economic problems that a business organization may have to face.

There are several parties now connected with the accounting information they are as follows: · Proprietors

· Managers · Suppliers · Investors

· Government · Public

Thus accounting is being used anywhere and everywhere as it is the need of an hour to give rise to any business event that occurs daily many times as it accounts for the profitability and returns of the business.

NATURE OF ACCOUNTING:

Accounting as a discipline has been originated to serve the purpose of the organization in maintaining and updating the records and transactions of the various business events on a day-to-day basis.

It functions as a means from which various parties connected to the business can get the accounting information and the management is able to do the decision-making for various business plans and policies. The accountants are hired for maintaining and preparing the records and financial statements, provide the results and also conduct analysis and conclusions drawn from them.

From the above on e can derive the basic nature as follows:

1. It keeps and maintains the record of the business.
2. It also helps to analyze and interpret the financial data.
3. It is useful in decision-making.
4. It is useful in preparation and compilation of financial results.
5. It also serves as a means to depict the financial position of the business.

6. It ignores the qualitatitive aspects of the business.

7. The information provided in accounting is based on estimates.

FUNCTIONS OF ACCOUNTING

1. Record Keeping
2. Protecting of properties
3. Communication of results
4. Meeting legal requirements

OBJECTIVES OF ACCOUNTING ARE

1. To keep systematic records: Its main objective is to keep complete record of Business transactions. It avoids the possibility of omission and fraud.

2. To calculate profit or loss: Accounting helps to ascertain the net profit earned or loss suffered on account of business transactions during a particular period. To ascertain profit or loss at the end of each accounting period Trading and Profit& Loss is prepared.

3. To facilitate decision making: The information collected from various financial statements is useful in decision-making and is of use to the parties connected to the business.

4. To ascertain the financial position of the business:

The businessman is able to know the financial position of the firm through the profits made or losses incurred by the firm during the accounting period as well as the balance-sheet serve as a basis of serving the financial status of the firm at a particular date.

RELATIONSHIP OF ACCOUNTING WITH ECONOMICS

Prof. Robbins has defined term 'economics' as follows-

"Economics is the science, which studies human behavior as a relationship between ends and scarce means which have alternatives uses."However when a person is to take any economical decision, he has to depend mainly on the accounting information. Generally an accountant is concerned with the economic problems of an only one enterprise only, but an economist is concerned with the problems of an industry as a whole. Micro levels data, arranged by the accounting system, is summed up to get macro level data base. Thus at the macro level, accounting provides the basic data, upon which the economic models are developed.

However, there exists a wide gulf between economists' and accountants' concept of income and capital For example, the profit according to an economist is not same thing as the profit according to an accountant. No doubt, accountants have derived the ideas of value, income and capital maintenance from economists, but suitably modified to make them usable in practical circumstances. Thus, Economics and Accounting are close related subjects.

Relationship of Accounting with Mathematics

Knowledge of arithmetic and algebra is a pre-requisite for accounting computations and measurements. Calculations of interest and annuity etc. are some examples of fundamental uses of mathematics in accounting.

Presently graphs and charts are being widely used for communicating accounting information to the users. Thus the knowledge in geometry and trigonometry has become essential to have a better understanding about the accounting communication system.

Relationship of Accounting with Statistics

Collection, Tabulation, Analysis and presentation of data are some primary functions, which are performed by both Accountants and statisticians.The accountant is mainly concerned with the monetary data, although to some extent, he is also concerned with the quantitative data. But a statistician is concerned equally with the monetary and quantitative data.The use of statistics is accounting can be appreciated better in the context of the nature of accounting records. Accounting information is very precise; it is exact to the last paisa. But for decision-making purposes, such precision is not necessary and hence the statistical approximations are sought.

Accounting records generally confined to one year, while statistical analysis is more useful if a longer period is taken. For example, a longer period will be required to fit the trend line. Statistical methods are helpful in developing accounting data and in their interrelation. Therefore the study and application of statistical methods will add extra edge to the accounting data.

Relationship of Accounting with Law

Every business house has to work within legal environment. All the transactions with suppliers and customers are governed by the Contract Act, Sale of Goods Act, Negotiable Instruments Act, etc. The entity, itself, created and controlled by laws. For example a partnership business is controlled by Partnership Act, a company is created and controlled by the Companies Act. Very often the accounting system to be followed has been prescribed by the law. For example the Companies Act has prescribed the format of financial statements. However legal prescription about the accounting system is the product of development in accounting knowledge. That is to say legislation about accounting system cannot be enacted unless there is a corresponding development in the accounting discipline. In what way, accounting influences law and is also influenced by law.

Relationship of Accounting with Management

Management is a broad occupational field, which comprises many functions and application of many disciplines including statistics, mathematics, economics, etc. Accountants are well placed in the management and play a key role in the management them. A large portion of accounting information is prepared for management decision-making. In the management team, an accountant is in a better position to understand and use such data. In other words, since an accountant plays an active role in management, he understands the data requirement. So, accounting system can be molded to serve the management purpose.

Advantages of Financial Accounting:

• It provides legal information to stakeholders such as financial accounts in the form of trading, profit and loss account and balance sheet.

• It shows the mode of investment for shareholders. • It provides business trade credit for suppliers.

• It notifies the risks of loan in business for banks and lenders.

Limitations of Financial Accounting:

One of the major limitations of financial accounting is that it does not take into
Account the non-monetary facts of the business like the competition in the market, change in the value for money etc.

The following limitations of financial accounting have led to the development of cost Accounting:

1. **No clear idea of operating efficiency**: You will agree that, at times, profits may be more or less, not because of efficiency or inefficiency but because of inflation or trade depression. Financial accounting will not give you a clear picture of operating efficiency when prices are rising or decreasing because of inflation or trade depression.
2. **Weakness not spotted out by collective results**: Financial accounting discloses only the net result of the collective activities of a business as a whole. It does not indicate profit or loss of each department, job, process or contract. It does not disclose the exact cause of inefficiency i.e.it does not tell where the weakness is because it discloses the net profit of all the activities of a business as a whole. Say, for instance, it can be compared with a reading on a thermometer. A reading of more than 98.4° or less than 98.4º discloses that something is wrong with the human body but the exact disease is not disclosed. Similarly, loss or less profit disclosed by the profit and loss account is a signal of bad performance of the business in whole, but the exact cause of such performance is not identified.
3. **Not helpful in price fixation:** In financial accounting, costs are not available as an aid in determining prices of the products, services, production order and lines of products.
4. **No classification of expenses and accounts**: In financial accounting, there is no such

system by which accounts are classified so as to give relevant data regarding costs by departments, processes, products in the manufacturing divisions, by units of product lines and sales territories, by departments, services and functions in the administrative division. Further expenses are not attributed as to direct and indirect items. They are not assigned to the products at each stage of production to show the controllable and uncontrollable items of overhead costs.

5. **No data for comparison and decision-making**: It will not provide you with useful data for comparison with a previous period. It also does not facilitate taking various financial decisions like introduction of new products, replacement of labour by machines, price in normal or special circumstances, producing a part in the factory or sourcing it from the market, production of a product to be continued or given up, priority accorded to different products and whether investment should be made in new products etc.
6. **No control on cost**: It does not provide for a proper control of materials and supplies, wages, labour and overheads.

7. **No standards to assess the performance**: In financial accounting, there is no such well developed system of standards, which would enable you to appraise the efficiency of the organization in using materials, labour and overhead costs. Again, it does not provide you any such information, which would help you to assess the performance of various persons and departments in order that costs do not exceed a reasonable limit for a given quantum of work of the requisite quality.

Basis of Accounting:

There are basically two systems of accounting: 1. **Cash System of Accounting.**

2. **Accrual System of Accounting.**

Cash system of Accounting:

It is a system in which accounting entries are made only when cash is received or paid. No entry is made when a payment or receipt is merely due. For example, the rent for December 2009 has not been paid till January 10the 2010. Under cash basis, rent expense for the month of December will not be recorded as payment has not been made. Government system of accounting is mostly on the cash system.

Accrual System of Accounting:

It is a system in which accounting entries are made on the basis of amount having become due for payment or receipt. This system recognizes the fact that if a transaction or an event occurred, its consequences cannot be avoided and therefore, should be brought into book in order to present a meaningful picture of profit earned or loss suffered.

Accounting Principles-Concepts and Conventions

Meaning of Accounting Principles:

Financial accounting is information that must be processed and reported objectively. Third parties, who must rely on such information, have a right to be assured that the data is free from bias and inconsistency, whether deliberate or not. For this reason, financial accounting relies on certain standards or guides that are called 'Generally Accepted Accounting Principles' (GAAP). Principles derived from tradition, such as the concept of matching. In any report of
financial statements (audit, compilation, review, etc.), the preparer/auditor must indicate to the reader whether or not the information contained within the statements complies with GAAP.

ACCOUNTING PRINCIPLES

• **Principle of regularity**: Regularity can be defined as conformity to enforced rules and laws.

• Principle of consistency: This principle states that when a business has fixed a specific method for the accounting treatment of an item, it will enter all similar items that follow, in exactly the same way.

• Principle of sincerity: According to this principle, the accounting unit should reflect in good faith the reality of the company's financial status.

• **Principle of the permanence of methods**: This principle aims at maintaining the coherence and comparison of the financial information published by the company.

Principle of non-compensation: One should show the full details of the financial information and not seek to compensate a debt with an asset, revenue with an expense etc.

• **Principle of prudence**: This principle aims at showing the reality 'as is': one should not try to make things look rosier than they are. Typically, revenue should be recorded only when it is certain and a provision should be entered for an expense, which is probable.

• Principle of continuity: When stating financial information, one assumes that business will not be interrupted. This principle mitigates the principle of prudence: assets do not have to be accounted at their disposable value, but it is accepted that they are at their historical value.

• **Principle of periodicity**: Each accounting entry should be allocated to a given period and split accordingly if it covers several periods. If a client pre-pays a subscription (or lease, etc.), the given revenue should be split to the entire time-span and not accounted for entirely on the date of the transaction.

• **Principle of full disclosure/materiality**: All information and values pertaining to the financial position of a business must be disclosed in the records.

Accounting Concepts and Conventions:

An accounting convention is a modus operandi of universally accepted system of recording and presenting accounting information to the concerned parties. They are followed judiciously and rarely ignored. Accounting conventions are evolved through the regular and consistent practice over the years to aid unvarying recording in the books of accounts. Accounting conventions help in comparing accounting data of different business units or of the same unit for different periods. These have been developed over the years.

1. **Convention of relevance**: The convention of relevance emphasizes the fact that only such information should be made available by accounting that is pertinent and helpful for achieving its objectives. The relevance of the items to be recorded depends on its nature and the amount involved. It includes information, which will influence the decision of its client. This is also known as convention of materiality.

For example, business is interested in knowing as to what has been the total labor cost. It is neither interested in knowing the amount employees spend nor what they save.

2. **Convention of feasibility**: The convention of feasibility emphasizes that the time, labor and cost of analyzing accounting information should be comparable to the benefits arising out of it. For example, the cost of 'oiling and greasing' the machinery is so small that its break-up per unit produced will be meaningless and will amount to wastage of labor and time of the accounting staff.

3. **Convention of consistency**: The convention of consistency means that the same accounting principles should be used for preparing financial statements year on year. An evocative conclusion can be drawn from financial statements of the same enterprise when there is similarity between them over a period of time. However, these are possible only when accounting policies and practices followed by the enterprise are uniform and consistent over a period. If dissimilar accounting procedures and practices are followed for preparing financial statements of different accounting years, then the result will not be analogous. Generally, a business follows the above-mentioned general practices or methods year after year. For example, while charging depreciation on fixed assets or valuing unsold stock, if a particular method is used it should be followed year after year, so that the financial statements can be

analyzed and a comparison made.

4. **<u>Convention of objectivity</u>**: The convention of objectivity highlights that accounting information should be measured and expressed by the standards which are universally acceptable. For example, unsold stock of goods at the end of the year should be valued at cost price or market price, whichever is less and not at a higher price even if it is likely to be sold at a higher price in the future.

5. **<u>Convention of full disclosure</u>**: Convention of full disclosure states that all material and relevant facts concerning financial statements should be fully disclosed. Full disclosure means that there should be complete, reasonable and sufficient disclosure of accounting information. Full refers to complete and detailed presentation of information. Thus, the convention of full disclosure suggests that every financial statement should disclose all pertinent information. For example, the business provides financial information to all interested parties like investors, lenders, creditors, shareholders etc. The shareholder would like to know the profitability of the firm while the creditors would like to know the solvency of the business. This is only possible if the financial statement discloses all relevant information in a complete, fair and an unprejudiced manner.

6. **<u>Convention of conservatism</u>**: This concept accentuates that profits should never be overstated or anticipated. However, if the business anticipates any loss in the near future, provision should be made for it in the books of accounts, for the same. For example, creating provision for doubtful debts, discount on debtors, writing off intangible assets like goodwill, patent and so on should be taken in to consideration. Traditionally, accounting follows the rule 'anticipate no profit and provide for all possible losses. 'For example, the closing stock is valued at cost price or market price, whichever is lower. The effect of the above is that in case market price has come down then provides for the 'anticipated loss', but if the market price has increased then ignore 'anticipated profits'. The convention of conservatism is a valuable tool in situation of ambiguity and qualms.

<u>Introduction To Accounting Standards:</u> ACCOUNTING STANDARDS ISSUED BY ICAI

AS-1 DISCLOSURE OF ACCOUNTING POLICIES

AS-2 VALUATION OF INVENTORIES

AS-3 CASH FLOW STATEMENTS

AS-4 CONTINGENCIES AND EVENTS OCCURING AFTER THE BALANCE SHEET DATE

AS-5 NET PROFIT OR LOSS FOR THE PERIOD, PRIOR PERIOD ITEMS AND CHANGES IN ACCOUNTING POLICIES

AS-6 DEPRECIATION ACCOUNTING

AS-7 ACCOUNTING FOR CONSTRUCTION CONTRACTS

AS-8 ACCOUNTING FOR RESEARCH AND DEVELOPMENT

AS-9 REVENUE RECOGNITION

AS-10 ACCOUNTING FOR FIXED ASSETS

AS-11 ACCOUNTING FOR THE EFFECTS OF CHANGES IN FOREIGN EXCHANGE RATES

AS-12 ACCOUNTING FOR GOVERNMENT GRANTS

AS-13 ACCOUNTING FOR INVESTMENTS

AS-14 ACCOUNTING FOR AMALGAMATIONS

AS-15 ACCOUNTING FOR RETIREMENT BENEFITS IN THE FINANCIAL STATEMENTS OF EMPLOYERS

AS-16 BORROWING COSTS

Basic Concepts of Accounting Standards:

Basic concepts of (GAAP) Accounting Standards :

In order to achieve the aforesaid objectives of GAAP and implement fundamental qualities, the set accounting standards feature four basic assumptions as listed below:

- **Going concern**

This assumption of GAAP accounting standards presumes that the business stays in operation indefinitely thus validating the techniques of asset capitalization, amortization, and depreciation. This assumption is, however, not applicable in case of liquidation. The business is believed to continue in the unforeseeable future.

- **Monetary Unit Principle**

This assumption presumes a stable currency going to be the unit of record.

- **Accounting Entity**

This assumption presumes the business to individually exist from its owners or other business entities. Also, revenue and expense need to be kept separate from personal expenses.

- **Time-period principle**

This assumption states that an entity's economic activities can be divided into simulated time periods.

Benefits of Accounting Standards:

As per the accounting standards presented by GAAP, the financial reports should provide info which is:

- Useful in being presented to potential creditors and investors in addition to other users in making cogent investment, credit, and similar financial decisions.
- Helpful for the potential creditors and investors in addition to other users in evaluating the timing, amounts, and uncertainty of probable cash receipts.
- Related to economic resources, the claims to these resources, and the changes occurring in them.
- Helpful in taking financial decisions. · Helpful in taking long-term decisions.
- Helpful in improving the business' performance. · Useful in maintaining records.

<u>Procedure for Issuing Accounting Standards in India</u>

A summarized extract of the text of the "Preface to the Statements of Accounting Standards (Revised 2004)," issued by the council of the Institute of Chartered Accountants of India, explains the procedure of issuing Accounting Standards. They are:

1. The ASB determines the broad areas requiring formulation of Accounting Standards and lists them according to priority.
2. In the preparation of Accounting Standards, the ASB is assisted by a Study Group, constituted for this purpose. Views of government, public sector undertakings, industry and other organizations are obtained before formulating the Exposure Draft.
3. The Exposure Draft comprises the following:

- Objective and scope of the standard.
- Definition of the terms used in the standard.
- The manner in which the accounting principles have been applied for formulating the standard.
- The presentations and disclosure requirements of it comply with the standard. · Class of enterprises to which the standard will apply.
- Date from which the standard will be effective.

<u>Need and Significance of IFRS:</u>

<u>International Financial Reporting Standards (IFRS)</u> are principles-based Standards, Interpretations and the Framework adopted by the International Accounting Standards Board (IASB). IFRS represent a set of internationally accepted accounting principles used by companies to prepare financial statements.

The goal with IFRS is to make international comparisons as easy as possible. More than 100 countries around the world currently require or permit IFRS reporting. Approximately 85 of those countries require IFRS reporting for all domestic, listed companies. All member states of the EU are required to use IFRS as adopted by the EU for listed companies since 2005. The US is also gearing towards IFRS. While some countries require all companies to adhere to IFRS, others merely allow it or try to coordinate their own country's standards to be similar.

IFRS include the following Standards:

IFRS 1 First-time Adoption of International Financial Reporting Standards. ·

IFRS 2 Share-based Payment.

IFRS 3 Business Combinations. ·

IFRS 4 Insurance Contracts.

IFRS 5 Non-current Assets Held for Sale and Discontinued Operations. ·

IFRS 6 Exploration for and Evaluation of Mineral Resources.

IFRS 7 Financial Instruments: Disclosures. ·

IFRS 8 Operating Segments.

IFRS 9 Financial Instruments.

IAS 1 Presentation of Financial Statements.

IAS 2 Inventories.

IAS 7 Statement of Cash Flows.

IAS 8 Accounting Policies, Changes in Accounting Estimates and Errors. ·

IAS 10 Events after the Reporting Period.

IAS 11 Construction Contracts. ·

IAS 12 Income Taxes.

IAS 16 Properties, Plant and Equipment.

IAS 17 Leases.

IAS 18 Revenue.

IAS 19 Employee Benefits.

IAS 20 Accounting for Government Grants and Disclosure of Government Assistance.

IAS 21 The Effects of Changes in Foreign Exchange Rates.

IAS 23 Borrowing Costs.

IAS 24 Related Party Disclosures.

IAS 26 Accounting and Reporting by Retirement Benefit Plans. ·

IAS 27 Consolidated and Separate Financial Statements.

IAS 28 Investments in Associates.

·IAS 29 Financial Reporting in Hyperinflationary Economies. ·

IAS 31 Interests in Joint Ventures.

IAS 32 Financial Instruments: Presentation. ·

IAS 33 Earnings Per Share.

IAS 34 Interim Financial Reporting. ·

IAS 36 Impairment of Assets.

IAS 37 Provisions, Contingent Liabilities and Contingent Assets. ·

IAS 38 Intangible Assets.

IAS 39 Financial Instruments: Recognition and Measurement.

IAS 40 Investment Property.

IAS 41 Agriculture.

International Accounting Standards (IAS) are the older standards that IFRS are gradually replacing (IAS were issued from 1973 to 2000).

XBRL: XBRL(Extensive Business Reporting Language) is a freely available and global standard for exchanging business information. XBRL allows the expression of semantic meaning commonly required in business reporting. The language is XML-based and uses the XML syntax and related XML technologies such as XML Schema, XLink, XPath, and Namespaces. One use of XBRL is to define and exchange financial information, such as a financial statement. The XBRL Specification is developed and published by XBRL International, Inc. (**XII**).

XBRL is a standards-based way to communicate and exchange business information between business systems. These communications are defined by metadata set out in taxonomies, which capture the definition of individual reporting concepts as well as the relationships between concepts and other semantic meaning. Information being communicated or exchanged is provided within an XBRL instance. Early users of XBRL included regulators such as the U.S. Federal Deposit Insurance Corporation· and the Committee of European Banking

Supervisors (CEBS)·. Common functions in many countries that make use of XBRL include regulators of stock exchanges and securities, banking regulators, business registrars, revenue reporting and tax-filing agencies, and national statistical agencies.

UNIT-2 JOURNALISING TRANSACTIONS

Rules of Debit and Credit

Any account that obtains a benefit is Debit. OR

Anything that will provide benefit to the business is Debit.

Both these statements may look different but in fact if we consider that whenever an account benefits as a result of a transaction, it will have to return that benefit to the business then both the statements will look like different sides of the same picture. For credit, Any account that provides a benefit is **Credit.**

OR

Anything to which the business has a responsibility to return a benefit in future is **credit**. As explained in the case of Debit, whenever an account provides benefit to the business the business will have a responsibility to return that benefit at some time in future and so it is Credit.

***Rules of Debit and Credit for Assets**

Similarly we have established that whenever a business transfers a value / benefit to an account and as a result creates something that will provide future benefit; the `thing' is termed as **Asset**.

By combining both these rules we can devise following rules of Debit and Credit for Assets: When an asset is created or purchased, value / benefit is transferred to that account, so it is debited.

I. Increase in Asset is Debit

Reversing the above situation if the asset is sold, which is termed as disposing off, for say cash, the asset account provides benefit to the cash account. Therefore, the asset account is credited.

II. Decrease in Asset is Credit

***Rules of Debit and Credit for Liabilities**

Anything that transfers value to the business, and in turn creates a responsibility on part of the business to return a benefit, is a **Liability**. Therefore, liabilities are the exact opposite of the assets.

When a liability is created the benefit is provided to business by that account so it is Credited

III. Increase in Liability is Credit

When the business returns the benefit or repays the liability, the liability account benefits from the business. So it is debited

IV. Decrease in Liability is Debit

***Rules of Debit and Credit for Expenses**

Just like assets, we have to pay for expenses. From assets, we draw benefit for a long time whereas the benefit from expenses is for a short run. Therefore, Expenditure is just like Asset but for a short run. Using our rule for Debit and Credit, when we pay cash for any expense that expense account benefits from cash, therefore, it is debited. Now we can lay down our rule for Expenditure:

V. Increase in Expenditure is Debit

Reversing the above situation, if we return any item that we had purchased, we will receive cash in return. Cash account will receive benefit from that Expenditure account. Therefore, Expenditure account will be credited

VI. Decrease in Expenditure is Credit *Rules of Debit and Credit for Income

Income accounts are exactly opposite to expense accounts just as liabilities are opposite to that of assets.

Therefore, using the same principle we can draw our rules of Debit and Credit for Income.

VII. Increase in Income is Credit VIII. Decrease in Income is Debit

<u>Posting and Preparation of Trial Balance:</u>

<u>Trial Balance:</u>

Trial Balance is a list of closing balances of ledger accounts on a certain date and is the first step towards the preparation of financial statements. It is usually prepared at the end of an accounting period to assist in the drafting of financial statements. Ledger balances are segregated into debit balances and credit balances. Asset and expense accounts appear on the debit side of the trial balance whereas liabilities, capital and income accounts appear on the credit side. If all accounting entries are recorded correctly and all the ledger balances are accurately extracted, the total of all debit balances appearing in the trial balance must equal to the sum of all credit balances.

<u>How to prepare a Trial Balance:</u>

Following Steps are involved in the preparation of a Trial Balance:

1. All Ledger Accounts are closed at the end of an accounting period.
2. Ledger balances are posted into the trial balance.
3. Trial Balance is cast and errors are identified.
4. Suspense account is created to agree the trial balance totals temporarily until corrections

are accounted for.

5. Errors identified earlier are rectified by posting corrective entries.
6. Any adjustments required at the period end not previously accounted for are incorporated into the trial balance.

Closing Ledger Accounts:

Ledger accounts are closed at the end of each accounting period by calculating the totals of debit and credit sides of a ledger. The difference between the sum of debits and credits is known as the closing balance. This is the amount which is posted in the trial balance.

How closing balances are presented in the ledger depends on whether the account is related to income statement (income and expenses) or balance sheet (assets, liabilities and equity). Balance sheet ledger accounts are closed by writing 'Balance c/d' next to the balancing figure since these are to be rolled forward in the next accounting period. Income statement ledger accounts on the other hand are closed by writing 'Income Statement' next to the residual amount because it is being transferred to the income statement as revenue or expense incurred for the period.

The steps involved in closing a ledger account may be summarized as below:

1. Add the totals of both sides of a ledger
2. The higher of the totals among the debit side and credit side must be inserted at the end of **BOTH** sides. Closing balance is the balancing figure on the side with the lower balance.
3. In case of ledger accounts of assets, liabilities and equity, 'balance c/d' is written next to the closing balance whereas in case of income and expenses ledger accounts, 'Income Statement' is written next to the closing balance.
4. The closing balances of all ledger accounts are posted into the trial balance.

Closing Balance of all ledger accounts are posted into the trial balance. It is important to remember that a debit closing balance in the ledger account appears on the credit side but in the trial balance it is presented in the debit column and vice versa.

Posting of closing balances should be done carefully as many errors may occur during the posting process such as Posting Error, Transposition Errors and Slide error.

Following is an example of a trial balance prepared from the closing balances of the ledgers detailed above.

ABC	**LTD**
Trial Balance as at 31 December 2011	

Account Title	Debit (in rupees)	Credit (in rupees)
Share Capital		10,000
Bank Loan		10,000
Cash	30,000	
Salaries Expense	5,000	
Sales Revenue		15,000
Total	**35,000**	**35,000**

Capital and Revenue:

Capital: The amount invested in the firm by the owner is referred to as the Capital. The expenditure borne/incurred in purchase of fixed assets, investments is known as Capital Expenditure.

Revenue: The profit/gain arising from the business from the sale proceeds is referred to as the revenue. Revenue includes the total cost/expenses (whether direct or indirect) and the total income earned during the accounting period by the firm.

Classification of Income:

- Gain on sale of Investments. ·
- Capital Income.
- Dividends Received
- Interest Received.
- Commission Received.

Classification of Expenditure:

in Fixed Assets.

· Purchase of Land & Building.

· Repairs and Installation of Tools & Equipments. · Payment of direct and indirect expenses.

· Deferred Revenue Expenditure. · Depreciation on Fixed Assets.

<u>**Capital and Revenue Expenditure**</u>

Expenditure on fixed assets may be classified into Capital Expenditure **and** Revenue Expenditure. **The distinction between the nature of capital and revenue expenditure is important as only capital expenditure is included in the cost of fixed asset.**

<u>**Capital Expenditure:**</u>

Capital expenditure includes costs incurred on the acquisition of a fixed asset and any subsequent expenditure that increases the earning capacity of an existing fixed asset. The cost of acquisition not only includes the cost of purchases but also any additional costs incurred in bringing the fixed asset into its present location and condition (e.g. delivery costs).

Capital expenditure, as opposed to revenue expenditure, is generally of a one-off kind and its benefit is derived over several accounting periods. Capital Expenditure may include the following:

- Purchase costs (less any discount received)
  Delivery costs
- Legal charges
- Installation costs
- Up gradation costs
- Replacement costs

<u>**Revenue Expenditure**</u>

Revenue expenditure incurred on fixed assets include costs that are aimed at 'maintaining' rather than enhancing the earning capacity of the assets. These are costs that are incurred on a regular basis and the benefit from these costs is obtained over a relatively short period of time. For example, a company buys a machine for the production of biscuits. Whereas the initial purchase and installation costs would be classified as capital expenditure, any subsequent repair and maintenance charges incurred in the future will be classified as revenue expenditure. This is so because repair and maintenance costs do not increase the earning capacity of the machine but only maintains it (i.e. machine will produce the same quantity of biscuits as it did when it was first purchased).

Revenue costs therefore comprise of the following:

- Repair costs

- Maintenance charges
- Repainting costs
- Renewal expenses

As revenue costs do not form part of the fixed asset cost, they are expensed in the income statement in the period in which they are incurred.

Journal and subsidiary Books: **JOURNAL**

The word 'Journal' has been derived from the French word 'JOUR' means daily records. **Journal** is a book of original entry in which transactions are recorded as and when they occur in chronological order (in order of date) from source documents. Recording in journal is made showing the accounts to be debited and credited in a systematic manner. Thus, the journal provides a date-wise record of all the transactions with details of the accounts and amounts debited and credited for each transaction with a short explanation, which is known as narration. Firms having limited number of transactions record those in journal and from there post these to the concerned ledger accounts. Firms having large number of transactions, maintain some special purpose journals such as, Purchase Book, Sales Books, Returns books, Bills Book, Cash Book, Journal proper etc.

COMPOUND JOURNAL ENTRY

A compound journal entry is an accounting entry which effects more than two account heads. A simple journal entry has one debit and one credit whereas a compound journal entries includes one or more debits and/or credits than a simple journal entry. A compound journal entry may combine two or more debits and a credit, or a debit and two or more credits, or two or more of both debits and credits.

OPENING ENTRY

In the case of continuing business we are required to pass an entry in the journal for bringing in the new books all assets and liabilities as appearing in the books on the last day of the previous year. This entry is known as 'opening entry'. Rule of passing opening entry is to debit each asset account; credit each liability account; excess of debits over credits represents capital balance.

SUB DIVISION OF JOURNAL

CASH JOURNAL

The number of cash transactions in a firm is generally larger, therefore, it becomes inconvenient to record all cash transactions in the journal. Since all cash transactions are recorded for the first time in the cash book, it is therefore called a book of original entry. Only cash transactions are recorded in the cash book.

PETTY CASH BOOK

It saves the time of chief cashier. Maintenance of petty cash book does not require any specialized knowledge of accounting. It provides control over small payments. It minimizes the chances of fraud.

PURCHASE JOURNAL

It is a subsidiary book which records transactions of credit purchases of goods. Cash purchases are not recorded in the purchases book since they are recorded in the cash book.

SALES JOURNAL

It is a subsidiary book which records transactions of credit sales of goods. Cash sales will be recorded in the cash book and not in the sales book. Sale of assets is not recorded in the sales books.

SALES RETURN JOURNAL

Sales returns journal is a subsidiary book in which seller records all the sales that have been returned to him by his customers. Sales returns journal is also known as returns inwards book and sales returns day book.

VOUCHER SYSTEM

Type of internal system used to control the cash (checks) being spent (written). The voucher system consists of vouchers, voucher files (paid and unpaid), voucher register that takes the place of the purchase journal, cash register that takes the place of the cash disbursement journal, and the general journal.

Unit-3 Depreciation

Depreciation is systematic allocation the cost of a fixed asset over its useful life. It is a way of matching the cost of a fixed asset with the revenue (or other economic benefits) it generates over its useful life. Without depreciation accounting, the entire cost of a fixed asset will be recognized in the year of purchase. Depreciation is the measure of wearing out of a fixed asset. All fixed assets are expected to be less efficient as time goes on.

Depreciation is calculated as the estimate of this measure of wearing out and is charged to the Profit & Loss account either on a monthly or annual basis. The cost of the asset less the total depreciation will give you the Net Book Value of the asset. This will give a misleading view of the profitability of the entity. The observation may be explained by way of an example.

Example

ABC LTD purchased a machine costing $1000 on 1st January 2001. It had a useful life of three years over which it generated annual sales of $800. ABC LTD's annual costs during the three years were $300.

Causes for Depreciation:

The major causes of depreciation are as follows:

1. **Wear And Tear** wear and tear refer to a decline in the efficiency of asset due to its constant use. When an asset losses its efficiency, its value goes down and depreciation arises. This is true in case of tangible assets like plant and machinery, building, furniture, tools and equipment used in the factory.

2. **Effusion of Time** The value of asset may decrease due to the passage of time even if it is not in use. There are some intangible fixed assets like copyright, patent right, and lease hold premises which decrease its value as time elapse.

3. **Exhaustion** An asset may loss its value because of exhaustion too. This is the case with wasting assets such as mines, quarries, oil-wells and forest-stand. On account of continuous extraction, a stage will come where mines and oil-wells get completely exhausted.

Objectives for providing Depreciation:

1. **Ascertainment of True Profits**
When an asset is purchased, it is nothing more than a payment in advance for for the use of asset. Depreciation is the cost of using a fixed asset. To determine true and correct amount of profit or loss, depreciation must be treated as revenue expenses and debited to profit and loss account.

2. **Reporting of True And Fair Financial Position Of A Business**
The value of assets decrease over a period of time on account of various factors. In order to present a true state of affairs of the business, the assets should be shown in the balance sheet, at their true and fair values. If the depreciation is not provided then the asset will appear in the balance sheet at the original value. So, in order to show the true financial position of a business, depreciation is required to be charged on the assets.

3.**Replacement of Assets** Assets used in the business need to be replaced after the expiry of their useful life. Depreciation can be taken as a source of fund for replacing worn out asset by a new asset. Thus, depreciation charges help in accumulating funds for the replacement of an asset.

4. Saving In Taxes

The profit and loss account will show more profits if depreciation is not charged on asset. So, the business needs to pay more income tax to the government. Depreciation charges on assets save the amount of tax equivalent to tax rate. Since it is shown as expense in the profit and loss account, it reduces the amount of the profit.

4. Obsolescence

Changes in fashion are external factors which are responsible for throwing out of assets even if those are in good condition. For example black and white televisions have become obsolete with the introduction of color TVs, the users have discarded black and white TVs although they are in good condition. Such as loss on account of new invention or changed fashions is termed as obsolescence.

Methods of Calculating Depreciation:

There are two basic methods of depreciation to choose from when depreciating an asset. These methods include

Straight-line and Written Down Value Method..

The **Straight-Line** method is generally the most commonly used method due to its simplicity and consistency of allocating depreciation evenly over the useful life of the asset. To calculate depreciation under this method, the Cost of the Asset is reduced by the salvage or residual value to arrive at the depreciable basis. The resulting depreciable basis is then divided by the estimated useful life.

Straight Line Method (SLM)

This is the simple method of depreciation.

It charges equal amount of depreciation each year over useful life of asset.

It first add up all the costs incurred to bring the asset in use and then it divides that by the useful life of asset in years to calculate the depreciation expense.

E.g.: Say a Computer costs Rs. 30,000 and Rs. 11,000 (as additional set-up/installation/maintenance expenses) = Rs 41,000 and it is anticipated that its scrap value will be Rs. 1,000 at the end of its useful life, of say, 5 yrs.

Total Cost = Cost of Computer + Installation Exp. + Other Direct Costs

Depreciable Amount over No. of years = Total Cost - Salvage Value (At end of useful life)

30,000 +11,000 =41,000 (Total cost)

41,000 – 1,000 = 40,000 as the **Depreciable Amount**

Depreciable Amount = Rs. 40,000, Spread out over 5 years = Rs. 40,000/5(Yrs) = Rs. 8000/- depreciation per annum

Advantages:

· The straight-line method offers simplicity

· Write off the same amount each year and don’t have to keep recalculating.

· Easy to determine profits for future years easily; at least as far as how much you will save because of depreciation. In other words, as your profits grow, your depreciation costs stay the same.

· This allows you to make financial forecasts for several years.

· You receive the benefit of depreciation evenly over the life of the asset.

Disadvantages:

· Assets tend to lose value more quickly in their early years.

· Straight-line depreciation does not take this fact into account. If you use assets as collateral for loans, the lender will assume your assets lose more of their value in their first years, so your straight-line method will make the asset look more valuable on your books than it really is for the lender.

· The efficiency of a machine declines in its later years and you will be writing off the same value in a later year as you did in the first year. The machine actually will have less value for you than your depreciation amount indicates.

Written Down Value Method (WDV)

- This method involves applying the depreciation rate on the Net Book Value (NBV) of asset. In this method, depreciation of the asset is done at a constant rate.
- In this method depreciation charges reduces each successive period.
- This method should be used in those assets, where high depreciation should be charged in initial years.

Assume the price of a depreciable asset i.e. computer is Rs. 40,000 and its salvage value after 10 years is 0. In this method NBV will never be zero.

Depreciation Per year = (1/N) Previous year's value, Where N= No. of years

So in our example, the depreciation amount during the first year is

[Rs. 40,000*1/10] =Rs. 4,000

NBV of computer after 1st year= Rs 40,000- 4,000 = Rs. 36,000

Depreciation for 2nd year is

[Rs. 36,000*1/10] =Rs. 3,600

Advantages:

1. Since under this method higher depreciation is charged in early years it takes into account that asset is more efficient in early years and therefore it is more realistic way of depreciation.

2. Since in early years machines requires less repairs and as the year passes by repair cost began to rise, therefore this method by charging more depreciation in early years and less in later years make sure that total cost of repairs and depreciation is same every year.

Disadvantages:

1. Since the rate of depreciation is fixed by not following formula chances of subjectivity in fixing rate of depreciation becomes high.

2. The value of asset will never be zero in books of account under this even if asset is of no use to company.

Change in method of charging Depreciation:

CHOICE OF DEPRECIATION METHOD

Depreciation expenses differ from method to method. Choice of selecting a suitable depreciation method is not easy. The decision is based on the inherent characteristic features of an asset. Accelerated depreciation methods may be of much use in case of the following: Quality of the asset decreases with its age years roll, assets may loose its effective working capacity – Maintenance costs grow.

· Introduction of new equipment due to Research and Development may adversely affect the effective usage of existing equipment.

· The other Straight Line Method may be suitable for assets like buildings, furniture, patents, leases, etc. and for assets which do not warrant frequent repairs and renewals.

· Choice of a method of depreciation affects the amount of net income because quite often the management employs depreciation as an instrument of financial policy of the entity. Hence, selection of a method depends on the management too.

Change in depreciation amount due to change in method is to be given retrospective effect but in all other cases (like Change in Cost, Life, Revaluation etc.) Change in depreciation

is given prospective effect

1. The depreciation method selected should be applied consistently from period to period.
2. A change from one method of providing depreciation to another should be made only if the adoption of the new method is required by statute or for compliance with an accounting standard or if it is considered that the change would result in a more appropriate preparation or presentation of the financial statements of the enterprise.
3. When a change in the method of depreciation is made, depreciation should be recalculated in accordance with the new method from the date of the asset coming into use. The deficiency or surplus arising from the retrospective recomputation of depreciation in accordance with the new method would be adjusted in the accounts in the year in which the method of depreciation is changed.
4. In case the change in the method results in deficiency in depreciation in respect of past years, the deficiency should be charged to the profit and loss account.

In case the change in the method results in surplus, it is recommended that the surplus be initially transferred to the 'Appropriations' part of the profit and loss account and thence to General Reserve through the same part of the profit and loss account. Such a change should be treated as a change in accounting policy and its effects should be quantified and disclosed.

Salient Features of Accounting Standards(AS-6)(ICAI)Revised:

The Institute of Chartered Accountants of India, keeping in view with international accounting principles, revised (AS)–6.This standard AS–6 deals with the concept: Depreciation is defined as "a measure of the wearing out, consumption or other loss of value of a depreciable asset arising from use, effusion of time or obsolescence through technology and market changes. Depreciation is allocated so as to charge a fair proportion of the depreciable amount in each accounting period during the Expected Useful Life of the Asset. Depreciation includes amortization of assets whose useful life is predetermined".

Salient features of AS–6 (Revised) Accounting for Depreciation:

i. Existing Assets: The depreciable amount of existing assets = Cost of the asset (historical not market value) – salvage (scrap value) value.

ii. Revision of estimate useful life of an asset: In case, if there is a necessity to revise the estimated life of an asset, the unamortized depreciable amount will have to be charged over the remaining useful life.

Addition (or) extension to an existing asset of capital nature: In such cases, two factors will have to be considered:

a) such an addition should retain separate identity,
b) It can still be used after the disposal of existing assets.

Then, depreciation is to be determined independently on the basis of an estimate of its own useful life. In other cases, the depreciation has to be determined on the basis of remaining useful life of the existing asset plus addition or extension as an integral part.

INVENTORIES:

The raw materials, work-in-process goods and completely finished goods that are considered to be the portion of a business's assets that is ready or will be ready for sale. Inventory represents one of the most important assets that most businesses possess, because the turnover of inventory represents one of the primary sources of revenue generation and subsequent earnings for the company's shareholders/owners.

Inventories are assets:

(a) held for sale in the ordinary course of business; (b) In the process of production for such sale; or (c) In the form of materials or supplies to be consumed in the production process or in the rendering of services.

Inventories encompass goods purchased and held for resale, for example, merchandise purchased by a retailer and held for resale, computer software held for resale, or land and other property held for resale. Inventories also encompass finished goods produced, or work in progress being produced, by the enterprise and include materials, maintenance supplies, consumables and loose tools awaiting use in the production process. Inventories do notinclude machinery spares which can be used only in connection with an item of fixed asset and whose use is expected to be irregular; such machinery spares are accounted for in accordance with Accounting Standard (AS) 10, Accounting for Fixed Assets.

Valuation of Inventories:

Inventory is valued on the basis of the following factors:

1. **Measurement**

Inventories should be valued at the lower of cost and net realizable value.

The cost of inventories should comprise all costs of purchase, costs of conversion and other costs incurred in bringing the inventories to their present location and condition.

2. **Costs of Purchase**

The costs of purchase consist of the purchase price including duties and taxes (other than those subsequently recoverable by the enterprise from the taxing authorities), freight inwards and other expenditure directly attributable to the acquisition. Trade discounts, rebates, duty drawbacks and other similar items are deducted in determining the costs of purchase.

3. **Costs of Conversion**

The costs of conversion of inventories include costs directly related to the units of production, such as direct labour. They also include a systematic allocation of fixed and variable production overheads that are incurred in converting materials into finished goods. Fixed production overheads are those indirect costs of production that remain relatively constant regardless of the volume of production, such as depreciation and maintenance of factory buildings and the cost of factory management and administration. Variable production overheads are those indirect costs of production that vary directly, or nearly directly, with the volume of production, such as indirect materials.

4. **Other Costs**

Other costs are included in the cost of inventories only to the extent that they are incurred in bringing the inventories to their present location and condition. For example, it may be appropriate to include overheads other than production overheads or the costs of designing products for specific customers in the cost of inventories. Interest and other borrowing costs are usually considered as not relating to bringing the inventories to their present location and condition and are, therefore, usually not included in the cost of inventories.

Method of valuation of Inventories:

There are three basis approaches to valuing inventory that are allowed by GAAP -
(a) **First-in, First-out (FIFO):** Under FIFO, the cost of goods sold is based upon the cost of material bought earliest in the period, while the cost of inventory is based upon the cost of material bought later in the year. This results in inventory being valued close to current replacement cost. During periods of inflation, the use of FIFO will result in the lowest estimate of cost of goods sold among the three approaches, and the highest net income.

(b) **Last-in, First-out (LIFO):** Under LIFO, the cost of goods sold is based upon the cost of material bought towards the end of the period, resulting in costs that closely approximate current costs. The inventory, however, is valued on the basis of the cost of materials bought earlier in the year. During periods of inflation, the use of LIFO will result in the highest estimate of cost of goods sold among the three approaches, and the lowest net income.

(c) **Weighted Average:** Under the weighted average approach, both inventory and the cost of goods sold are based upon the average cost of all units bought during the period. When inventory turns over rapidly this approach will more closely resemble FIFO than LIFO.

Firms often adopt the LIFO approach for the tax benefits during periods of high inflation, and studies indicate that firms with the following characteristics are more likely to adopt LIFO - rising prices for raw materials and labor, more variable inventory growth, an absence of other tax loss carry forwards, and large size. When firms switch from FIFO to LIFO in valuing inventory, there is likely to be a drop in net income and a concurrent increase in cash flows (because of the tax savings). The reverse will apply when firms switch from LIFO to FIFO.

Given the income and cash flow effects of inventory valuation methods, it is often difficult to compare firms that use different methods. There is, however, one way of adjusting for these differences.

Firms that choose to use the LIFO approach to value inventories have to specify in a footnote the difference in inventory valuation between FIFO and LIFO, and this difference is termed the LIFO reserve. This can be used to adjust the beginning and ending inventories, and consequently the cost of goods sold, and to restate income based upon FIFO valuation.

Periodic Inventory System:

Periodic inventory is a system of inventory in which updates are made on a periodic basis. This differs from perpetual inventory systems, where updates are made as seen fit.In a periodic inventory system no effort is made to keep up-to-date records of either the inventory or the cost of goods sold. Instead, these amounts are determined only periodically - usually at the end of each year. This physical count determines the amount of inventory appearing in the balance sheet. The cost of goods sold for the entire year then is determined by a short computation.

Perpetual Inventory System:

Under perpetual inventory system, inventory and cost of goods sold are updated for each sale/purchase and return transaction. We have already discussed the basic concept of perpetual inventory system in the comparison of perpetual-periodic inventory.

The perpetual inventory system is intended as an aid to material control. It is a system of stock control followed by stores department. The system follows a method of recording stores by which information about each receipt, issue and current balance of stock is always available.

Perpetual inventory system may be defined as a method of recording stores balances after every receipt and issue to facilitate regular checking and to obviate closing down for stock taking." So perpetual inventory system implies continuous maintenance of stock records and in its broad sense it covers both continuous stock taking as well as up to date recording of stores books. The balance of the same item of store in bin card should correspond with that shown in the materials or store ledger card and a frequent checking of these two records should be made and compared with the actual or physical quantity of materials in stock.

Final Accounts:

The financial statements of an organization made up at the end of an accounting period, usually the fiscal year. For a manufacturer, the final accounts consist of (1) manufacturing account, (2) trading account, (3) profit and loss account, and (4) profit and loss appropriation account. A commercial company's final accounts will include all of the above except the manufacturing account. Together, these accounts show the gross profit, net income, and distribution of net income figures of the company.

Form of Final Accounts: There is a standard format of final accounts only in the case of a limited company. There is no fixed prescribed format of financial accounts in the case of a proprietary concern and partnership firm.

- Transactions · Journal
- Ledger
- Trial Balance
- Trading & Profit & Loss Account · Balance Sheet

MEANING

The Trading and Profit & loss account and Balance Sheet prepared at the end of a year is known as **<u>Final accounts</u>**. While preparing the final accounts, there may be some items so far not adjusted. These items are to be adjusted in the final accounts for calculating the correct profit or loss of the business. The usual adjustments in the final accounts are:-

a. <u>Expenses owing: -</u> These are the expenses incurred during the year but not paid in cash. This will be paid in the near future (next year). The owing expense is to be added with the amount of same expense already paid given in the trial balance and it should be shown in the balance sheet as a current liability.

The double entry for recording the expenses owing is

Debit Expenses account

Credit Expenses owing account

This expense is also known as outstanding expenses, expenses payable or expense payable.

b. Prepaid expense. :- This is the expense paid during the year for the benefit of the next year. The portion of the expense which is prepaid is to be deducted from the total expenses already paid during the year (given in the trial balance) and shown as current asset in the balance sheet.

The double entry for recording the prepaid expense is

Debit Prepaid expense account and

Credit Expense account

This expense is also known as expense paid in advance or unexpired expense

c. Accrued income: - The income earned during the year but not received in cash is known as accrued income. The amount of accrued income is to be considered as current year's income and added

With the concerned income received during the year (given in the trial balance) and shown as a current asset in the balance sheet.

The double entry for recording the accrued income is:

Debit Accrued income account and

Credit Income account

The accrued income is also known as outstanding income.

d. Income received in advance: - This is the income received during the year for the services to be rendered during the next year. Since this income is not related to the current year, it should be deducted from the concerned income (given in the trial balance) and shown as a current liability in the balance sheet.

The double entry for recording the income received in advance is:

Debit Income account and

Credit Income received in advance

This is also known as unexpired income.

e. Depreciation: - The part of the cost of a fixed asset that is consumed by a business during the period of its use is known as depreciation. It is considered as an expense in the business therefore shown as an expense in the profit & loss account and deducted from the cost price of the concerned fixed asset in the balance sheet.

The double entry for recording depreciation is:

Debit Profit & loss account and

Credit Depreciation account

f. Bad debt: - The part of the amount of debtors which cannot be recovered is known as bad debt. It is an expense to be shown in the profit & loss account. If the bad debt appears in the trial balance, it is known as bad debt written off and shown in the profit & loss account only. If bad debt information appears among the adjustment points below the trial balance, then it should be shown as an expense in the profit & loss account and shown as a deduction from the debtors in the balance sheet under the heading "current assets".

The double entry for recording the bad debt is:

Debit Bad debt account and

Credit Debtors account

g. Goods drawings by the owner for his personal use:-

The amount of goods withdrawn by the owner for his personal use is to be considered as drawing. The double entry for recording the goods drawings is:

Debit Drawings account and

Credit Purchase account or sales account

The amount of goods drawings should be deducted from purchases and capital in the Balance Sheet.

From the following Trial Balance and additional information, you are required to prepare profit and loss account and Balance Sheet.

TRIAL BALANCE as on 31st March 2012

Particulars	**Debit**	**Credit**
Capital		**2,90,000**
Sundry Debtors	**65,000**	
Drawings	**7,600**	
Building	**2,20,000**	
Sundry Creditors		**12,000**
Wages	**8,000**	
Purchases	**89,000**	
Opening Stock	**12,000**	
Cash in Hand	**1,900**	
Cash at Bank	**12,000**	
Carriage Charges	**20,000**	
Salaries	**8,000**	

Rent, Taxes & Insurance	**1300**	
Sales		**1,50,000**
Purchase Returns		**4,500**
Sales Returns	**2,800**	
Bills Receivable	**15,000**	
Bils Payable		**7,000**
Interest		**3,500**
Advertisement	**2,400**	
Trade Expenses	**2,000**	
	4,67,000	**4,67,000**

Additional Information:

i. Closing Stock as on 31st March2012 was Rs.15,000.

ii. Prepaid Insurance Rs.400

iii. Outstanding Salaries Rs.2,000;Outstanding Rent & Taxes Rs.1,300 iv. Depreciation charged on Building @2%p.a.

Solution:

Dr. Trading and Profit & Loss Account as on 31st March2012 Cr.

Particulars	**Amount**	Particulars	**Amount**
To Opening Stock	12,000	By Closing Stock	15,000
To Purchases 89,000 (-)Returns 4,500	84,500	By Sales 1,50,000 (-)Returns 2,800	147200
To Wages	8,000		
To Carriage	20,000		
To Gross profit c/d	37,700		
	1,62,200		**1,62,200**
To Trade Expenses	2,000	By Gross Profit b/d	**37,700**
To Advertisement	2,400	By Interest	**3,500**
To Salaries 8,000	10,000		
(+)Outstanding 2,000			
To Rent,Taxes & Insurance 1,300 (-)Prepaid 400	2,200		
To Depreciation on Building	4,400		
To Net Profit transferred to Capital A/C	20,200		

	41,200		**41,200**

Balance-Sheet as on 31st March2012

Liabilities	**Amount**	**Assets**	**Amount**
Capital 2,90,000 (+)Net Profit 20,200 (-)Drawings 7,600	3,02,600	Building 2,20,000 (-)Depreciation 4,400	2,15,600
Creditors	12,000	Cash at Bank	12,000
Bills Payable	7,000	Cash in Hand	1,900
Outstanding Rent & Taxes	1,300	Debtors	65,000
Outstanding Salaries	2,000	Bills Receivable	15,000
		Prepaid Insurance	400
		Closing Stock	15,000
	3,24,900		**3,24,900**

PREPARATION OF FINAL ACCOUNTS OF NON-PROFIT ORGANIZATION

By preparing Receipts & Payments Account, Income and Expenditure Account and a Balance sheet. Being a "non-profit" organization does not mean it doesn't make any profit, but rather that the profits are not distributed to investors as dividends. Many non-profit organizations make billions of dollars of profits per year, and often donate these corporate profits to charitable causes.

BANK RECONCILIATION STATEMENT

A **Bank reconciliation** is a process that explains the difference between the bank balance shown in an organisation's bank statement, as supplied by the bank, and the corresponding amount shown in the organization's own accounting records at a particular point in time. Such differences may occur, for example, because a cheque or a list of cheques issued by the organization has not been presented to the bank, a banking transaction, such as a credit received, or a charge made by the bank, has not yet been recorded in the organisation's books, or either the bank or the organization itself has made an error. It may be easy to reconcile the difference by looking at very recent transactions in either the bank statement or the organisation's own accounting records (cash book) and seeing if some combination of them tallies with the difference to be explained. Otherwise it may be necessary to go through and match every single transaction in both sets of records since the last reconciliation, and see what transactions remain unmatched. The necessary adjustments should then be made in the cash book, or any timing differences recorded to assist with future reconciliations. A bank reconciliation statement is prepared to reconcile the two balances of Cash Book and Pass Book. So, when you will prepare a bank reconciliation statement you will start it with one balance make adjustments and then you will reach to the other balance. This way both the balances will agree.

Unit-IV

CONSIGNMENT:

An arrangement whereby goods are left in the possession of another party to sell. Typically, the consignor receives a percentage of the sale (sometimes a very large percentage). Consignment deals are made on a variety of products - from artwork, to clothing, to books. In recent years, consignment shops have become rather trendy, especially those offering specialty products, infant wear and high-end fashion items. It is also defined as a quantity of goods that are sent to a person or place to be sold the act or process of sending goods to a person or place to be sold.

Features of consignment are:

· The relation between the two parties is that of consignor and consignee and not that of buyer and seller

· The consignor is entitled to receive all the expenses in connection with consignment · The consignee is not responsible for damage of goods during transport or any other procedure

·

Goods are sold at the risk of consignor. The profit or loss belongs to consignor only

A consignor who consigns goods to a consignee transfers possession but not ownership of the goods to the consignee. The consignor retains title to the goods. The consignee takes possession of the goods subject to a trust. If the consignee converts the goods to a use not contemplated in the consignment agreement, for example selling them and keeping the proceeds of the sale for himself, then the consignee commits the crime of embezzlement.

Accounting Entries in the Books of Consignor: (1) **On dispatch of goods:-**

Consignment account

To Goods sent on consignment account (With the cost of goods)

(2)**On payment of expenses on dispatch:-**

Consignment account

To Bank account (With the amount spent as expenses)

(3) **On receiving advance:**

Cash or bills receivable account

To Consignee's personal account (With the amount cash or bill)

(4)**On the consignee reporting sale (as per A/S):-**

Consignee's personal account

To Consignment account (With gross proceeds of sales)

(5) **For expenses incurred by the consignee (as per A/S):-**

Consignment account

To Consignee's personal account (With the amount of expenses)

(6) **For commission payable to the consignee:-**

Consignment account

To Consignee's personal account (With the amount of expenses)

Difference between Consignment and Joint Venture

The main differences between joint venture and consignment are as under:

1. Nature

Joint venture: It is a temporary partnership business without a firm name. **Consignment:** It is an extension of business by principal through agent.

2. Parties

Joint venture: The parties involving in joint venture are known as co-ventures. **Consignment:** Consignor and consignee are involving parties in the consignment.

3. Relation

Joint venture: The relation between co-ventures is just like the partners in partnership firm. **Consignment:** The relation between the consignor and consignee is 'principal and agent'.

4. Sharing Profit

Joint venture: The profits ans losses of joint venture are shared among the co-ventures in their agreed proportion.

Consignment: The profits and losses are not shared between the consignor and consignee. Consignee gets only the commission.

5. Rights

Joint venture: The co-ventures in a joint venture have equal rights.

Consignment: In consignment, the consignor enjoys principal's right whereas consignee enjoys the right of agent.

6. Exchange Of Information

Joint venture: The co-ventures exchange the required information among them regularly. **Consignment:** The consignee prepares an account sale which contains a details of business activities carried on and is being sent to the consignor.

7. Ownership

Joint Venture: All the co-ventures are the owners of the joint venture. **Consignment:** The consignor is the owner of the business.

8. Method Of Maintaining Accounts

Joint venture: There are different methods of maintaining accounts in joint venture.As per agreement the co-ventures maintain their account.

Consignment: In consignment, there is only one method of maintaining account.

9. Basis of Account

Joint venture: Cash basis of accounting is applicable in joint venture. **Consignment:** Actual basis is adopted in consignment.

10. Continuity

Joint venture: As soon as the particular venture is completed, the joint venture is terminated. **Consignment:** The continuity of business exists according to the willingness of both consignor and consignee.

<u>Loss of Goods on Consignment</u>

The goods are consigned from one place to another. After receiving the goods by consignee, the goods are stored by the consignee before selling them to customers. It is natural that some loss to the goods may take place within that period. The goods may be lost, destroyed or damaged either in transit or in consignee's store. Such loss can be divided into two parts.

1. Normal Loss

The loss which is caused by unavoidable reasons is known as normal loss. For example shrinkage, evaporation, leakage and pilferage. Such losses form part of cost of goods and no additional adjustment is required for this purpose. The normal loss is borne by goods units. The quantity of such loss is to be deducted from the total quantity sent by the consignor. The following formula may be used for the valuation of unsold stock.

Value of closing stock= (Total value of goods sent/Net quantity received by consignee) X unsold quantity

Net quantity received = Goods consigned quantity - Normal loss quantity.

2. Abnormal Loss

The loss which could be avoided by proper planning and care are abnormal loss. They are like theft, riots, accidents, fire, earthquake etc. These losses could occur in transit or in consignee's store and solely to be borne by consignor. The abnormal loss should be adjusted before ascertaining the result of the consignment. The valuation of abnormal loss is done on the same basis as the unsold stock is valued. The journal entries for abnormal loss in different cases are as under:

If goods are not insured For recording abnormal loss:

Abnormal loss A/C ....Dr. To consignment A/C

For abnormal loss transferred: Profit and loss A/C........Dr.

To abnormal loss A/C

If goods are insured and claim admitted in full Bank/Consignee's/Insurance company A/C...........Dr.

To consignment A/C

If goods are insured and claim admitted in partial Profit and loss A/C...........Dr. (Net loss amount)

Insurance Co./bank/consignee's A/c.......Dr. (Claim admitted) To consignment A/c (total loss amount)

The following method should be followed while valuing abnormal loss:

A) Goods sent on consignment (at cost price)…............$ XXX B) Add: Non-recurring expenses: Consignor’s expenses…..$ XXX Consignee’s expenses…...$ XXX Total cost before abnormal loss A+B….........................$ XXX

Value of abnormal loss = (Total cost/Total units consigned) X abnormal loss units.

Question: A& Co. of Kolkata sent on consignment account goods to B& Co. of Mumbai at an invoice price of Rs.29675 and paid for freight Rs.762,Cartage rs.232 and insurance rs.700.Half the goods were sold by agents for Rs.17,500,subject to the agent’s commission of Rs.875,storage expenses of Rs.200 and other selling expenses of Rs.350.One-fourth of the consignment was lost by fire and a claim of Rs.5000 was recovered. Draw up the necessary accounts in the books of A & Co. and ascertain the profit or loss made on the consignment. The consignor received a two months bill of exchange from the agents in satisfaction of the dues.

Solution:

Consignment of Mumbai Account

Particulars	Amount	Particulars	Amount
To Goods sent on consignment	29,675	By B& Co.(Sale proceeds)	17500
To Cash: Freight 762 Cartage 232 Insurance 700	1694	By Abnormal Loss A/C	7843
To B & Co. Commission 875 Storage Expenses 200	1425	By Consignment Stock A/C	7843
Other Selling Expenses 350			
To Net Profit (transferred to P&L A/C)	392		
	33186		**33186**

<u>B & Co. Mumbai</u>

Particulars	Amount	Particulars	Amount
To Consignment to Mumbai sale proceeds)	17500	By Consignment to Mumbai(expense and commission)	1425
		By Bills Receiveble A/C	16075
	17500		**17500**

<u>Abnormal Loss Account</u>

Particulars	Amount	Particulars	Amount
To Consignment A/C	7843	By Bank(received from Insurance Co.)	5000
		By P&L A/C	2843
	7843		**7843**

Working Notes:

Calculation of Abnormal Loss

Particulars	**Rupees**
¼ of invoice Price of goods	**7419(approx)**
Add :¼ of Freight, Cartage and Insurance	**424**
Total Abnormal Loss	**7843**
Less: Recovered from insurance Co.	**5000**
Net Abnormal Loss	**2843**

Joint Venture:

An association of two or more individuals or companies engaged in a solitary business enterprise for profit without actual partnership or incorporation; also called a joint adventure. A joint venture is a contractual business undertaking between two or more parties. It is similar to a business partnership, with one key difference: a partnership generally involves an ongoing, long-term business relationship, whereas a joint venture is based on a single business transaction.

Individuals or companies choose to enter joint ventures in order to share strengths, minimize risks, and increase competitive advantages in the marketplace. Joint ventures can be distinct business units (a new business entity may be created for the joint venture) or collaborations between businesses. In a collaboration, for example, a high-technology firm may contract with a manufacturer to bring its idea for a product to market; the former provides the know-how, the latter the means.

Difference between a Joint venture and Partnership

Partnership

A partnership is a legal arrangement where two or more people own a business together. This means that the entire business is shared for as long as the business exists. Both partners contribute money, time and expertise to making a profitable enterprise, and that enterprise lasts until the partnership is dissolved.

Joint Venture

You enter a joint venture for a specific project. There is a time limit on joint ventures, and they have clearly stated limits on their purposes. You might enter a joint venture in order to make a product that neither partner can afford to make on her own. An example is developing new software. You do not give up half of your business in a joint venture; you share the profits and expenses for a particular venture.

Difference between Consignment and Joint Venture

The main differences between joint venture and consignment are as under:

1. Nature

Joint venture: It is a temporary partnership business without a firm name. **Consignment:** It is an extension of business by principal through agent.

2. Parties

Joint venture: The parties involving in joint venture are known as co-ventures. **Consignment:** Consignor and consignee are involving parties in the consignment.

3. Relation

Joint venture: The relation between co-ventures is just like the partners in partnership firm. **Consignment:** The relation between the consignor and consignee is 'principal and agent'.

4. Sharing Profit

Joint venture: The profits ans losses of joint venture are shared among the co-ventures in their agreed proportion.

Consignment: The profits and losses are not shared between the consignor and consignee. Consignee gets only the commission.

5. Rights

Joint venture: The co-ventures in a joint venture have equal rights.

Consignment: In consignment, the consignor enjoys principal's right whereas consignee enjoys the right of agent.

6. Exchange Of Information

Joint venture: The co-ventures exchange the required information among them regularly. **Consignment:** The consignee prepares an account sale which contains a details of business activities carried on and is being sent to the consignor.

7. Ownership

Joint Venture: All the co-ventures are the owners of the joint venture. **Consignment:** The consignor is the owner of the business.

8. Method Of Maintaining Accounts

Joint venture: There are different methods of maintaining accounts in joint venture. As per agreement the co-ventures maintain their account.

Consignment: In consignment, there is only one method of maintaining account.

9. Basis of Account

Joint venture: Cash basis of accounting is applicable in joint venture. **Consignment:** Actual basis is adopted in consignment.

10. Continuity

Joint venture: As soon as the particular venture is completed, the joint venture is terminated. **Consignment:** The continuity of business exists according to the willingness of both consignor and consignee.

FEATURES OF A JOINT VENTURE

The main features of a joint venture are specifically made clear. Two or more person are needed.

·It is an agreement to execute a particular venture or a project. · The joint venture business may not have a specific name.

· It is of temporary nature. So the agreement regarding the venture automatically stands terminated as soon as the venture is complete.

·The co-ventures share profit and loss in an agreed ratio. The profits and losses are to be shared equally if not agreed otherwise.

·

The co-ventures are free to continue with their own business unless agreed otherwise during the life of joint venture.

Accounting Entries in a Joint Venture

(1)When venture contributes cash to joint funds	Joint Bank A/C	Venturer's A/C(individual contribution)
(2) When amount is spent on account of expenses, for purchasing goods for the venture	Joint venture A/C	Joint Bank A/C
(3)If any expenses are paid by the	Joint Venture A/C	Venturer's A/C
(4)For Sales: i. Cash ii. Credit	Joint Bank A/C Sundry Debtors A/C	Joint Venture A/C Joint Venture A/C
(7) Balance of the Joint venture A/C will be either profit or Loss.	Joint Venture A/C(if profit)	Venturer's A/C

(8)Joint Bank Account and personal account of the Venturer's A/C will be automatically closed by the introduction and the withdrawal of cash.		

Hire Purchase:

Hire Purchase is defined asa system for purchasing merchandise, such as cars or furniture, in which the buyer takes possession of the merchandise on payment of a deposit and completes the purchase by paying a series of regular installments while the seller retains ownership until the final installment is paid

A method of buying goods through making installment payments over time. The term hire purchase originated in the U.K., and is similar to what are called "rent-to-own" arrangements in the United States. Under a hire purchase contract, the buyer is leasing the goods and does not obtain ownership until the full amount of the contract is paid.

Parties in a Hire-Purchase System:

1. **Hirer-**who buy the goods at any time by giving notice to the owner and paying the balance of the HP price less a rebate (each jurisdiction has a different formula for calculating the amount of this rebate).

He also to pay the hire installments and to take reasonable care of the goods (if the hirer damages the goods by using them in a non-standard way, he or she must continue to pay the installments and, if appropriate, recompense the owner for any loss in asset value).

Seller- a person who has the resources and the legal right to sell the goods on credit (which usually depends on a licensing system in most countries), the seller and the owner will be the same person. But most sellers prefer to receive a cash payment immediately He also has the right to terminate the agreement where the hirer defaults in paying the installments or breaches any of the other terms in the agreement. Hire Purchase agreements must be in writing and signed by both [parties].They must clearly lay out the following information in a print that all can read without effort:

1. A clear description of the goods 2. The cash price for the goods
3. The Hire Purchase price, i.e., the total sum that must be paid to hire and then purchase the goods

4. The deposit
5. The monthly installments (most states require that the applicable interest rate is disclosed

and regulate the rates and charges that can be applied in HP transactions) and

6. A reasonably comprehensive statement of the parties' rights (sometimes including the right to cancel the agreement during a "cooling-off" period).
7. The right of the hirer to terminate the contract when he feels like doing so with a valid reason.

<u>Characteristics of Hire-Purchase System</u>

The characteristics of hire-purchase system are as under

- Hire-purchase is a credit purchase.
- The price under hire-purchase system is paid in installments.
- The goods are delivered in the possession of the purchaser at the time of commencement of the agreement.
- Hire vendor continues to be the owner of the goods till the payment of last installment. · The hire-purchaser has a right to use the goods as a bailer.
- The hire-purchaser has a right to terminate the agreement at any time in the capacity of a hirer.

- The hire-purchaser becomes the owner of the goods after the payment of all installments as per the agreement.
- If there is a default in the payment of any installment, the hire vendor will take away the goods from the possession of the purchaser without refunding him any amount.

Thus," **<u>hire- purchase agreement</u>**" means an agreement under which goods are let on hire and under which the hirer has an option to purchase them in accordance with the terms of the agreement and includes an agreement under which-

· possession of goods is delivered by the owner thereof to a person on condition that such person pays the agreed amount in periodical installments, and

· the property in the goods is to pass to such person on the payment of the last of such installments, and (iii) such person has a right to terminate the agreement at any time before the property so passes;

· " hirer" means the person who obtains or has obtained possession of goods from an under hire- purchase agreement, and includes a person to whom the hirer' s rights or liabilities under the agreement have passed by assignment or by operation of law;

· “owner" means the person who lets or has let, delivers or has delivered possession of goods, to a hirer under a hire- purchase agreement and includes a person to whom the owner' s property in the goods or any of the owner' s rights or liabilities under the agreement has passed by assignment or by operation of law.

Difference between Hire-Purchase System and Installment System

Installment Payment System is system of purchase and sale of goods in which title of goods is immediately transferred to the purchaser at the time of sale of goods and the sale price of the goods is paid in installments. In the event of default in payment of any installment, the seller has no right to take back goods from the possession of the purchaser. He can file a suit for the recovery of the outstanding balance of the price of goods sold.

The followings are the differences between Hire-purchase system and Installment payment system:

· In **Hire-purchase system**, the transfer of ownership takes place after the payment of all installments while in case of **Installment payment system**, the ownership is transferred immediately at the time of agreement.

·

In **Hire-purchase system**, the hire-purchase agreement is like a contract of hire though later on it may become a purchase after the payment of last installment while in **Installment payment system**, the agreement is like a contract of credit purchase.

· In case of default in payment, in **Hire-purchase system** the vendor has a right to back goods from the possession of the hire-purchaser while in case of **Installment payment system**, the vendor has no right to take back the goods from the possession of the purchaser; he can simply sue for the balance due.

· In **Hire-purchase system**, if the purchaser sells the goods to a third party before the payment of last installment, the third party does not get a better title on the goods purchased. But in case of **Installment payment system**, the third party gets a better title on the goods purchased.

· In **Hire-purchase system** the provisions of the Hire-purchase Act apply to the transaction while in case of **Installment payment system**, the provisions of Sale of Goods Act apply to the transaction.

Accounting Entries in the Books of Hire purchaser

Journal Entries in the Books Of Hire Purchaser

There are two methods of recording hire purchase transactions in the books of the hire purchaser: i. When the asset is recorded in full cash price-full cash price method

ii. When the asset is recorded at cash price actually paid in each installment-: Actual cash price method.

1. **For the purchase of asset**: First Method

Asset A/C (full cash price)...........Dr. To vendor A/C

Second Method

No entry

2. **For the payment made for 'down payment'** First Method

Vendor A/C.............Dr. To bank A/C

Second Method Asset A/C............Dr. To Bank A/C

3**. For installment due**

First Method

Interest A/C............Dr. To vendor A/C

Second Method

Asset A/C (part of cash value)............Dr. To Interest A/C

4. **For the payment of installment (both method)** Vendor A/C............Dr.

To Bank A/C

5. **For charging depreciation(on the basis of cash value) (both methods)** Depreciation A/C...................Dr.

To Asset A/C

6. **For transfer of interest and depreciation (both methods)** Profit and loss A/C............Dr.

To depreciation A/C To interest A/C

Journal Entries In The Books Of Vendor

1. **For selling goods on hire purchase**

Hire purchase A/C...........Dr. (full cash price) To sales/hire purchase sales A/C

2. **For receiving down payment** Cash/bank A/C.................Dr.

To hire purchaser A/C

3. **For installment due**

Hire purchaser A/C...........Dr. To Interest A/C

4. **For receiving the installment** Cash/bank A/C ....Dr.

To hire purchaser A/C

5. **For transferring interest** Interest A/C............Dr.

To profit and loss A/C

Posting in Ledger Accounts: After passing journal entries under any of the methods discussed above, the following ledger accounts are opened in the ledger and the postings are made accordingly.

(i) Asset A/c. (e.g. Trucks A/c, Machinery A/c. etc.) (ii) Vendor's A/c. (iii)Interest A/c. (iv)Depreciation A/c.

Note: Before recording the entries the amounts of interest and depreciation will be calculated in two separate tables showing the calculations of interest and depreciation.

Calculation of Interest

The total payment made under hire-purchase system is more than cash price. In fact, this excess of payment over the cash price is interest. It is very essential to calculate interest because the amount paid for interest is charged to revenue and the asset is capitalized at cash price. Thus normally all installments will include a part of cash price and a part of interest on the outstanding balance. However the amount paid at the time of agreement (down payment) will not include any interest.

The calculation of interest is made under two conditions:

(a) When interest is included in amount of installment: Where the hire-purchase price i.e. payment made in the form of down payment and all installments is more than the cash price, it is regarded that the interest is included in installments.

Illustration: On Ist April, 2005 Mr. X purchased from M/s Y & Co. one 'Motor Truck' under hire-purchase system, Rs. 5,000 being paid on delivery and the balance in five annual installments of Rs. 7,500 each payable on 31st March each year. The cash price of the motor truck is Rs. 37,500 and vendors charge interest at the rate of 5 per cent per annum on yearly balances. Find out the amounts of principal and interest included in each installment.

(b) When interest is not included in installments: Where the total amount paid in the form of down payment and all installments is exactly equal to the cash price, it is regarded that the interest is not included in installments. It means that interest is payable in addition to the agreed amount of installment.

Question: On April 1,2005, A Transport Company purchased a Motor Lorry from Motor Supply Co. Ltd. on hire-purchase basis, the cash price being Rs. 60,000. Rs. 15,000 on signing of the contract and balance in three annual installments of Rs. 15,000 each on 31st

March every year. In addition to it, interest at 5 per cent per annum was also payable to vendors on outstanding balances.

Branch Accounting

An accounting system in which separate accounts are maintained for each branch of a corporate entity or organization. The primary objectives of branch accounting are better accountability and control, since profitability and efficiency can be closely tracked at the branch level. Branch accounting may involve added expenses for an organization in terms of accounting and infrastructure. This is because it may be necessary to appoint branch accountants to ensure accurate financial reporting and compliance with head office procedures and processes.

Types of Branches

The branches opened in the different parts of the nation, where the original undertaking being registered are called inland branches. These types of branches are also called home branches or national branches. There are two types of inland branches, which are:

a) Dependent branch b) Independent branch

a) Dependent Branch:

Dependent branches are the branches that do not keep their records but all the records are maintained by head office. They are not authorized to act solely without the prior permission of the head office. All the plans, policies, rules and regulations of these branches are totally formulated and executed by the head office. In other words, all the functions of dependent branch are totally controlled by head office.

b). Independent Branch:

The branches that can keep their accounts themselves and sell goods that are sent by the head office as well as those purchased by themselves are known as independent branches. These are the branches which can sell the goods to head office too. They can pay their own expenses and can deposit their collection in their own name in the bank. These branches record separately and independently all the transactions which are even recorded by the head office.

Systems of Accounting:

Stock and Debtors system is generally used when the goods are sent to the branch at pro-forma invoice price and the size of the branch is large. Under this system, the branch maintains a few central accounts to exercise greater control over the branch stock and other related expenses.

These accounts are as follows:

1.Branch Stock Account

2.Branch Debtors Account

3.Branch Expenses Account

4.Branch Adjustment Account

5.Goods Sent to Branch Account

6.Branch Stock Reserve Account

Branch Stock Account

This account is on the pattern of a stock account. The account helps the Head Office in maintaining an effective control over the Branch Stock and tells about shortage and surplus in the branch stock because of the difference between the pro-forma invoice price and the selling price. Unlike traditional accounting practice, branch stock a/c is always maintained on the selling price or pro-forma invoice price. Selling price is used to record the goods sold by the branch to its customer and goods returned by the branch customers.

Branch Debtors Account

Branch debtors' a/c is maintained in the traditional manner to record transactions in between branch and its credit customers

Branch Expense Account

The purpose of maintaining this account is nothing but the compile all branch expenses at one place. This will include all types of expenses i.e. cash based expenses and receivables based expenses

Branch Adjustment Account

Branch adjustment a/c replaces the branch income statement (profit & loss a/c). This is the account in which all expenses and losses are closed along with the margin that is a difference between cost and the selling price. This difference is split into two; one is termed as "surplus" that comes from the branch stock a/c representing the difference between selling price and pro-forma invoice price, the second is termed as "loading" that represents the difference between pro-forma invoice price and cost. This loading is calculated on opening and closing stock balances and also on the net of the goods sent branch.

Goods Sent to Branch Account

This is a supporting account, which is maintained to show second effects of the goods sent to branch and the goods returned from branch at pro-forma invoice price. Although the goods sent to and returned form the branch should be adjusted in the purchases a/c of the head office, but as we know that the branch stock a/c is not maintained at cost price, therefore, second effect of goods sent to and returned from branch is not recorded directly into the purchases a/c instead this second effect is recorded into the goods sent to branch a/c which after adjustment of the loading is finally closed into the purchases a/c.

Branch Stock Reserve Account

This is contra to branch stock account. In this account opening and closing balance of loading on branch stock is maintained.

<u>**Accounting Entries in books of head office under Debtors system of Branch Accounting**</u>

1) For Goods sent by Head Office to the Branch

   Branch A/C Dr.

   To Goods sent to the Branch A/C

2) For Goods returned by the Branch to H.O.

   Goods sent to the Branch A/C Dr.

   To Branch A/C

3) For Goods sent by the Branch to another Branch at instructions from the H.O.

   Goods sent to Branch A/C Dr.

   To Branch A/C

4) For Goods returned by the Branch Debtors to H.O. directly

   Goods sent to Branch A/C Dr.

   To Branch A/C

5) For Expenses at the Branch met by the H.O.

   Branch A/C Dr.

   To Bank A/C

6) For Assets at the Branch at end of accounting period

   Branch Assets A/C Dr.

   To Branch A/C

7) For Liabilities at the end of the accounting period

   Branch A/C Dr.

   To Branch Liabilities/C

8) For Profit/Loss

   If Profit: Branch A/C Dr.

   To General P&L A/C

   If Loss: General P&L A/c

To Branch A/C

9) For transfer of Balance in Goods sent to Branch account

Goods sent to Branch A/C Dr.

To Purchases/Trading A/C

10) For remittance of cash or cheque to the branch

Branch A/C Dr.

To Cash/Bank A/C

Question: Excellent Garments of Multan has a branch at Lahore. Goods are supplied to the branch at cost. The expenses of the branch are paid from Multan and the branch keeps a sales journal and the debtors' ledger only. From the following information supplied by the branch, prepare a Branch Account in the books of the head office. Goods are sent to branch at pro-forma invoice price which is cost plus 20%.(All figures in rupees).

Opening Stock (at Pro-forma invoice) 28,800, Closing Debtors9,150

Closing Stock (at Pro-forma invoice) 21,600

Opening Debtors 6200

Goods received from HO(at Pro-forma invoice)

Bad Debt 140

Credit Sales 41,000

Expenses paid by Head office 10,400

Cash Sales 17,500

Cash received from Debtors 37,900

Pilferage of goods by the employees(Normal Loss) 2,000

<u>Solution:</u>

(Debtors System)

In the books of H.O. (Multan)

Lahore Branch Account

Particulars	Amount	Particulars	Amount
Opening Stock	24,000	Cash Recd.fromBranch	17500
Opening Debtors	6200	Cash Recd. From Debtors	37900
Cash sent to Branch	10400	Goods sent to Branch	6720
Goods sent to Branch	40,320	Closing Stock	18000

To general P&L A/C	8360	Closing Debtors	9160
	89280		**89280**

Debtors Account

Particulars	Amount	Particulars	Amount
Opening		Cash recd.from	37900
Credit Sales	41000	Bad Debt	140
		Closing Debtors (c/.f.)	9160
	47200		**47200**

<u>Question:</u> On 1st January, 2008 goods costing Rs. 132,000 were invoiced by Multan head office to its new branch at Lahore and charged at selling price to produce a gross profit of 25% on the selling price. At the end of the year, the return from Lahore Branch showed that the credit sales were Rs. 150,000. Goods invoiced at Rs. 2,000 to Lahore branch have been returned to Multan head office. The closing stock at Lahore branch was Rs. 24,000 at selling price. Record the above transactions in the books of

(i)Lahore Branch Stock Account; (ii) Goods Sent to Lahore Branch Account; (iii)Lahore Branch Adjustment Account; and (iv) Lahore Branch Debtors Account in the head office book and close the said accounts on 31st December 2008.

In the Books of the Head Office, Multan

Lahore Branch Stock Account

Particulars	Amount	Particulars	Amount
To Goods sent to Lahore Branch	176000	By goods sent to Lahore	2000
		By a/c returns	150000
		By Branch Debtors(Cr.Sales)	24,000
	176000		**176000**

Goods Sent to Lahore Branch Account

Particulars	Amount	Particulars	Amount

To Lahore Branch Stock A/c(Returns)	2000	By Lahore Branch stock a/c	176000
To Lahore branch adj.a/c	43500		
Topurchases(bal.fig.)	130500		
	176000		**176000**

Lahore Branch Debtors Account

Particulars	**Amount**	**Particulars**	**Amount**
To Lahore Branch Stock a/c	150000	By balance c/d	150000
	150000		150000

Lahore Branch adjustment Account

Particulars	**Amount**	**Particulars**	**Amount**
To Stock Reserve	6000	By goods sent to Delhi Branch	43500
To general P&L a/c	37500		
	43500		43500

Independent Branch Accounting:

Accounting Entries under Independent Branch:

A. For goods supplied by head office to branch:

Branch book:

Goods supplied by head office A/C.........Dr. To Head office A/C

(Being receipt of goods)

Head office book: Branch A/C..............Dr.

To goods supplied to branch A/C (Being goods sent to branch)

B. **For cash remitted by head office to branch: Branch book:**

Cash A/C..............Dr. To head office A/C

(Being cash received)

Head office book: Branch A/C ....Dr. To cash A/C

(Being cash sent to branch)

C. **For goods returned by branch: Branch book:**

Head office A/C............Dr.

To goods supplied to head office A/C (Being goods return to head office)

Head office book:

Goods supplied from branch A/C............Dr. To Branch A/C

(Being goods returned from branch)

D. **For cash remitted by branch to head office: Branch book:**

Head office A/C.............Dr. To cash

(Being cash sent to head office)

Head office book: Cash A/C.................Dr. To Branch A/C

(Being cash received from branch)

E. **For assets purchased by branch on behalf of head office: Branch book:**

Head office A/C ....Dr. To cash A/C

(Being purchase of assets)

Head office book:

Branch assets A/C.............Dr. To branch A/C

(Being assets purchased by branch)

F. **For depreciation charged: Branch book:**

Depreciation A/C ....Dr.

To Head office A/C

(Being depreciation on branch fixed assets)

Adjustment in Profit sharing Ratio

When a new partner is admitted he/she acquires his/her share in profit from the existing partners.

As a result, the profit sharing ratio in the new firm is decided mutually between the existing partners and the new partner. The incoming partner acquires his/her share of future profits either incoming from one or more existing partner. The existing partners sacrifice a share of their profit in the favour of new partner, hence the calculation of new profit sharing ratio becomes necessary.

Sacrificing Ratio

At the time of admission of a partner, existing partners have to surrender some of their share in favour of the new partner. The ratio in which they agree to sacrifice their share of profits in favour of incoming partner is called sacrificing ratio. Some amount is paid to the existing partners for their sacrifice. The amount of compensation is paid by the new partner to the existing partner for acquiring the share of profit which they have surrendered in the favour of the new partner.

Sacrificing Ratio is calculated as follows:

Sacrificing Ratio = Existing Ratio – New Ratio

Following cases may arise for the calculation of new profit sharing ratio and sacrificing ratio:

Only the new partner's share is given

In this case, it is presumed that the existing partners continue to share the remaining profit in the same ratio in which they were sharing before the admission of the new partner. Then, existing partner's new ratio is calculated by dividing remaining share of the profit in their existing ratio. Sacrificing ratio is calculated by deducting new ratio from the existing ratio.

Question:

Deepak and Vivek are partners sharing profit in the ratio of 3 : 2. They admit Ashu as a new partner for 1/5 share in profit. Calculate the new profit
sharing ratio and sacrificing ratio. **Solution:**

Calculation of new profit sharing ratio: Let total Profit = 1

New partner's share = 1/5 Remaining share = 1 – 1/5 = 4/5

Deepak's new share = 3/5 of 4/5 i.e. 12/25

Vivek's new share = 2/5 of 4/5 i.e. 8/25 Ashu's Share = 1/5

The new profit sharing ratio of Deepak, Vivek and Ashu is : = 12/25 : 8/25 : 1/5 = 12 : 8 : 5/25 = 12 : 8 : 5

So Deepak Sacrificed = 3/5 – 12/25 = 15 – 12/25 = 3/25 Vivek Sacrificed = 2/5 – 8/25 = 10 – 8/25 = 2/25 Sacrificing Ratio = 3 : 2

Sacrificing ratio of the existing partners is same as their existing ratio.

(ii) The new partner purchases his/her share of the profit from the Existing partner in a particular ratio.

In this case : the new profit sharing ratio of the existing partners is to be ascertained after deducting the sacrifice agreed from his share. It means the incoming partner has purchased some share of profit in a particular ratio from the existing partners.

Only the new partner's share is given

In this case, it is presumed that the existing partners continue to share the remaining profit in the same ratio in which they were sharing before the admission of the new partner. Then, existing partner's new ratio is calculated by dividing remaining share of the profit in their existing ratio. Sacrificing ratio is calculated by deducting new ratio from the existing ratio.

When a new partner is admitted in the firm, the existing/old partners have to sacrifice, what is given to the new partner, from their future profits, the reputation they have gained in their past efforts and the side of capital they have taken before. The new partner when admitted, has to compensate for all these sacrifices made by the old ones. The compensation for such sacrifice can be termed as 'goodwill'. Hence, at the time of admission of the new partner, it is necessary to account the valuation of goodwill in the firm. If the new partner brings in cash for his share of goodwill, in addition to his capital, it is known as premium method. When the new partner brings nothing but only the capital, and the value of goodwill is erected or raised, this method of treatment is called Revaluation Method.

However, once creating the value of goodwill and writing of the same after admission is done, it can be said to be Memorandum Revaluation Method. Thus, keeping in mind, all these methods, the various ways of treating goodwill in the books of the firm at the time of admission of the new partner, are as follows:

1. Share of goodwill brought by the new partner in cash. 2. Share of goodwill brought by the new partner in kind.

3. Nothing is brought by the new partner as his share of goodwill.

4. Share of goodwill brought by the new partner in cash only a portion not as a whole. 5. Hidden goodwill

1. When the new partner brings his share of goodwill in cash

When the new partner brings his share of goodwill in cash, the payment ma be made to the old partners as if outside/private transaction. It may be retained in the business or after recording the same in the firm, the old partners may withdraw the whole amount or some portion only,

a. When the amount of goodwill brought by the new partner is not recorded in the books and the payment is made to the old partners as outside or private transaction, it does not affect in the transaction of the firm and hence no entry is passed in the books of the firm

b.When the amount of goodwill brought in by the new partner is retained in the business to increase cash resources, and if there exists already no-goodwill:

i) Cash/Bank A/C.......................Dr. To Goodwill A/c

(Being goodwill brought in by the new partner) ii) Goodwill A/C...........................Dr.

To old partners' capital A/C

(Being goodwill credited to old partners in the sacrificing ratio)

c. When there is no-goodwill already appeared in the books and the amount of goodwill brought in by the new partner, is fully or partially withdrawn by the old partners:

i) Cash A/C.......................Dr. To Goodwill A/C

(Being goodwill brought by the new partner)

ii) Goodwill A/C................Dr. To old partners' capital A/C

(Being goodwill divided among old partners) iii) Old partners' capital A/C...............Dr.

To Cash/Bank A/C

(Being the amount withdrawn)

d. When there is goodwill already appeared in the books and even then if the new partner brings his share of goodwill in cash, the amount may be retained or withdrawn by the old partners. If the amount of goodwill brought in by the new partner is retained in the business:

i) Old partners' capital A/C..............................Dr. To Goodwill A/C

(Being goodwill appearing in the book written off in the old ratio) ii) Cash/Bank A/C.......................Dr.

To Goodwill A/C

(Being goodwill brought in by the new partner) iii) Goodwill A/C.......................Dr.

To old partners' capital A/C

(Being goodwill brought in by new partner shared by the old partners)

If the goodwill amount brought by the new partner is withdrawn by the old partners, the following extra entry should also be passed:

Old partners' capital A/C.............Dr. To Cash/Bank

(Being amount withdrawn)

If they agree to show the original value of goodwill in the books, it is raised by passing the entry: Goodwill A/C ....Dr.

To All partners capital A/C (Being goodwill raised)

2. When the new partner brings his share of goodwill in kind

The new partner may bring his share of goodwill and capital in kind i.e. the form of assets instead of cash. Again, new partner may have an established name in the market among the customers. In such case, he may be recognized for his goodwill. As a result he will bring a lesser amount of assets than the amount of credited to him. This requires two journal entries:

i) All assets A/C............................Dr. Goodwill A/C/New partner's capital A/C

(Being goodwill brought in kind by the new partner)

ii) Goodwill A/C/New partner's capital A/C................Dr. To old partners' capital A/C

(Being goodwill shared by the old partners)

3. When the new partner is unable to bring his share of goodwill in cash or kind

When the new partner cannot bring anything for his share of goodwill, first of all we have to see if there exists goodwill already or not. If there is no-goodwill already appearing in the books of the firm, goodwill is raised at its full value. If goodwill already appears in the books, it is compared to the full value of goodwill raised or created and the adjustment is done accordingly. **a**.

When the new partner is unable to bring his share of goodwill and if there is no-goodwill already appearing in the books, goodwill is raised at its full value:

i) Goodwill A/C.................Dr. To old partners' capital A/C

(Being goodwill is created at its full value and credited to the old partners in old ratio) * By this entry, goodwill A/C then appears as an asset in the balance sheet of the firm.

b. If the new partner cannot bring his share of goodwill and there appears goodwill already in the books, even then goodwill is raised at its full value. If the raised value of goodwill is equal to the existing value of goodwill, no entry what so ever is needed. If the raised goodwill is more than the existing goodwill, then goodwill will be credited to the old partner's capital A/C by the excess amount only:

Goodwill A/C......................Dr. (excess value) To old partners' capital A/C

(Being the value of goodwill increased to..../increased by......)

* Goodwill then appears at its full value in the balance sheet of the firm

c. If the raised value of goodwill is less than the existing value of goodwill, then excess over

raised value of goodwill is written off:
Old partners' capital A/C................Dr. To Goodwill A/C
(Being the goodwill written off by the reduction in value)

d. Whatever the case may be stated in a,b,c, the partners may not wish goodwill in the books for an indefinite period after the admission of new one, as the value of goodwill changes constantly. They may write off the whole or some portion of the value of goodwill. For writing off the goodwill:
All partners' capital A/C.............Dr. To Goodwill A/C
(Being goodwill written off)

4. When the new partner can bring only a portion of his share of goodwill

When the new partner cannot bring the entire amount of his share of goodwill and he brings only a part of this, it is shared by the old partners in sacrificing ratio. Then goodwill A/C is raised in the books for the portion not brought by the new partner which is also credited to the old partners in their sacrificing ratio. Goodwill raised for the part of goodwill not brought in by the new partner is calculated as under:
= (Full value of goodwill/share of goodwill of new partner) X goodwill not brought in

But it should be remembered that , if there exists any goodwill in the books, first it should be written off by crediting to the old partners in old ratio. Therefore, the entries are:
i) Old partners' capital A/C..................Dr. To Goodwill A/C
(Being goodwill written off)
ii Cash/Bank A/C........................Dr. To Goodwill A/C
(Being the portion of goodwill brought in by new partner) iii) Goodwill A/C......................Dr.
To old partners' capital A/C
(Being the goodwill brought in by new partner credited to old partners)
iv) When the goodwill is raised for the part of goodwill not brought in by the new partner, the amount of goodwill is calculated as said above. The entry would be the same as in iii), only the amount being different, which is shared by the old partners in their old profit sharing ratio.

5. Hidden Goodwill

When the value of goodwill is not given in the question, the value of goodwill has to be calculated on the basis of total capital/net worth of the firm and profit sharing ratio.

A. New partner's capital X Reciprocal of the share of new partner....XXX B. Less net worth(excluding goodwill) of new firm...............................XXX

C. A-B = Value of goodwill..XXX

Retirement of a Partner:

When one or more partners leaves the firm and the remaining partners continue to do the business of the firm, it is known as retirement of a partner. Amit, Sunil and Ashu are partners in a firm. Due to some family problems, Ashu wants to leave the firm. The other partners decide to allow him to withdraw from the partnership. Thus, due to some reasons like old age, poor health, strained relations etc., an existing partner may decide to retire from the partnership. Due to retirement, the existing partnership comes to an end and the remaining partners form a new agreement and the partnership firm

is reconstituted with new terms and conditions. At the time of retirement the retiring partner's claim is settled.

A partner retires either :

(i) with the consent of all partners, or (ii) as per terms of the agreement; or (iii) at his or her own will.

The terms and conditions of retirement of a partner are normally provided in the partnership deed. If not, they are agreed upon by the partners at the time of retirement. At the time of retirement the following accounting issues are dealt :

(a) New profit sharing ratio and gaining ratio. (b) Goodwill

(c) Adjustment of changes in the value of Assets and liabilities (d) Treatment of reserve and accumulated profits.

(e) Settlement of retiring partners dues, (f) New capital of the continuing partners.

New profit sharing ratio and gaining ratio

As soon as a partner retires the profit sharing ratio of the continuing partners get changed. The share of the retiring partner is distributed amongst the continuing partners. In the absence of information, the continuing partners take the retiring partner's share in their profit sharing ratio or in agreed ratio. The ratio in which retiring partner's share is distributed amongst continuing partners is known as gaining ratio. It is

Gaining Ratio = New Ratio – Existing Ratio

Various cases of new ratio and gaining ratio are illustrated as follows: (i) Retiring partner's share distributed in Existing Ratio :

In this case, retiring partner's share is distributed in existing ratio amongst the remaining partners. The remaining partners continue to share profits and losses in the existing ratio.

The following example illustrates this :

Tanu, Manu and Rena are partners sharing profits and losses in the ratio of = 4 : 3 : 2. Tanu retires and remaining partners decide to take Tanu's share in the existing ratio i.e. 3 : 2. Calculate the new ratio of Manu and Rena.

Existing Ratio between Manu and Rena = 3/9 and 2/9 Tanu's Ratio (retiring partner) = 4/9

Tanu's share taken by the Manu and Rena in the ratio of 3 : 2 Manu's gets = 4/9 × 3/5 = 12/45

Manu's New Share = 3/9 + 12/45 = 27/45 Rena's gets = 4/9 × 2/5 = 8/45

Rena's New Share = 2/9 + 8/45 = 18/45

New ratio between Manu and Rena is 27/45 : 18/45 = 27 : 18 = 3 : 2. Gaining Ratio = New Ratio – Existing Ratio

Manu Gain = 27/45 – 3/9 = 12/45

Rena Gain = 18/45 – 2/9 = 8/45 12/45 : 8/45

3 : 2

You may note that the new ratio is similar to existing ratio that existed between Manu and Rena before Tanu's retirement.

Note: In absence of any information in the question, it will be presumed that retiring partner's share has been distributed in existing ratio.

(ii) Retiring partner's share distributed in Specified proportions: Sometimes the remaining partners purchase the share of the retiring partner in specified ratio. The share purchased by them is added to their old share and the new ratio is arrived at. The following example illustrates this:

A, B and C are partners in the firm sharing profits in the ratio of

3 : 2 : 1. B retired and his share was divided equally between A and C. Calculate the new profit sharing ratio of A and C.

B's Share = 2/6

B's share is divided between A and C in the ratio of 1 : 1. A gets 1/2 of 2/6 = 2/6 × 1/2 = 1/6

A's New Share = 3/6 + 1/6 = 4/6 C's gets 1/2 of 2/6 = 2/6 × 1/2 = 1/6 C's New share = 1/6+1/6 = 2/6

Gaining Ratio = New Ratio – Existing, Ratio

(iii)Retiring Partner's share is taken by one of the partners

The retiring partner's share is taken up by one of the remaining partners.

In this case, the retiring partner's share is added to that of partner's existing share. Only his/her share changes. The other partners continue to share profit in the existing ratio. An example illustrating this point is given below: Anuj, Babu and Rani share profit in the ratio of 5 : 4 : 2. Babu retires and his share is taken by Rani, So Rani's share is 2/11 + 4/11 = 6/11, Anuj share will remain unchanged i.e, 5/11. Thus, the new profit sharing ratio of Anuj and Rani is 5 : 6.

<u>Treatment of Goodwill</u>

The retiring partner is entitled to his/her share of goodwill at the time of retirement because the goodwill is the result of the efforts of all partners including the retiring one in the past. The retiring To Liabilities A/c [Individually]

(Increase in the value of liabilities)

(iv) **For decrease in value of Liabilities:** Liabilities A/c Dr. [Individually]

To Revaluation A/c

(decrease in the value of liabilities)

Revaluation account is prepared to record the change in the value of assets or liabilities. It will reveal profit or loss on revaluation. This profit or loss

is divided amongst all partners including the retiring/deceased partner in existing profit sharing ratio.

(iv) **For Profit on Revaluation :** Revaluation A/c Dr. (Individually) To Partner's Capital A/c

(Profit on revaluation divided amongst all partners in their existing profit sharing ratio)

(v**) For loss on Revaluation:**

Partner's Capital A/c Dr. (Individually) To Revaluation A/c

(Loss on revaluation borne by all partners in their existing profit sharing ratio.

(vi**) For distribution of undistributed profit and reserve.** Reserves A/c Dr

Profit & Loss A/c (Profit) Dr.

To Partners' Capital A/c (individually)

(Reserves and Profit & Loss (Profit) transferred to all partners capitals A/c in existing profit sharing ratio) (vii)**For distribution of undistributed loss** Partners' Capital A/c Dr. (individually)

To Profit & Loss A/c (Loss) [Profit & Loss (loss) transferred to

all partners Capitals A/c in old profit sharing ratio]

<u>Death of a Partner:</u>

The death of a partner dissolves the partnership. On the date of death, the accounts are closed and the net income for the year to date is allocated to the partners' capital accounts. Most agreements call for an audit and revaluation of the assets at this time. The balance of the deceased partner's capital account is then transferred to a liability account with the deceased's estate. The surviving partners may continue the business or liquidate. If the business continues, the procedures for settling with the estate are the same as those described earlier for the withdrawal of a partner. On the death of a partner, the accounting treatment regarding goodwill, Revaluation of assets and reassessment of liabilities, accumulated reserves and undistributed profit are similar to that of the retirement of a partner, When the partner dies the amount payable to him/her is paid to his/her legal representatives. The representatives are entitled to the followings :

(a) The amount standing to the credit to the capital account of the deceased partner

(b) Interest on capital, if provided in the partnership deed upto the date of death:

(c) Share of goodwill of the firm;

(d) Share of undistributed profit or reserves;

(e) Share of profit on the revaluation of assets and liabilities; (f) Share of profit upto the date of death;

(g) Share of Joint Life Policy.

The following amounts are debited to the account of the deceased partner's legal representatives: (i) Drawings

(ii) Interest on drawings

(iii) Share of loss on the revaluation of assets and liabilities;

(iv) Share of loss that have occurred till the date of his/her death.

The above adjustments are made in the capital account of the deceased partner and then the balance in the capital account is transferred to an account opened in the name of his/her executor.

The payment of the amount of the deceased partner depends on the

agreement. In the absence of an agreement the legal representative of adeceased partner is entitled to interest @ 6% p.a. on the amount due from the date of death till the date of final payment.

Calculation of profit upto the date of death of a partner.

If the death of a partner occurs during the year, the representatives of the deceased partner are entitled to his/her share of profits earned till the date of his/her death. Such profit is ascertained by any of the following methods: (i) Time Basis (ii) Turnover or Sales Basis

(i) Time Basis

In this case, it is assumed that profit has been earned uniformly throughout the year. For example: The total profit of previous year is Rs. 2,25,000 and a partner dies three months after the close of previous year, the profit of three months
is Rs. 31,250 i.e. 1,25,000 × 3/12, if the deceased partner took 2/10 share of profit, his/her share of profit till the date of death is Rs. 6,250 i.e. Rs. 31,250 × 2/10.

(ii) Turnover or Sales Basis

In this method, we have to take into consideration the profit and the total sales of the last year. Thereafter the profit upto the date of death is estimated on the basis of the sale of the last year. Profit is assumed to be earned uniformly at the same rate.

Illustration 12

Arun, Tarun and Neha are partners sharing profits in the ratio of 3 : 2 : 1 Neha dies on 31st May 2006. Sales for the year 2005-2006 amounted to Rs.4,00,000.and the profit on sales is Rs.60,000. Accounts are closed on 31 March every year. Sales from lst April 2006 to 31st May 2006 is Rs.1,00,000.Calculate the deceased partner's share in the profit upto the date of death.

Solution :

Profit from 1st April 2006 to 31st May 2006 on the basis of sales: If sales are Rs.4,00,000, profit is Rs.60,000

If the sales are Rs.1,00,000 profit is : 60,000/4,00,000 × 1,00,000 = Rs.15,000

Neha's share= 15,000 × 1/6 = Rs.2,500 Alternatively profit is calculated as

Rate of profit =60000/400000×100 = 15% Sale up to date of death = 1,00,000

Profit = 1,00,000X15/100 = Rs 15000.

OBJECT ORIENTED PROGRAMME (209)

UNIT – I

Introducing Object-Oriented Approach

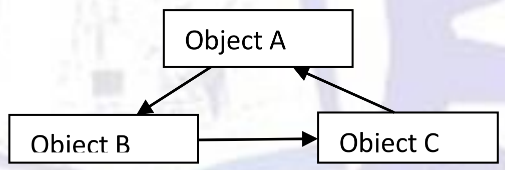
C++ is a statically typed, compiled, general-purpose, case-sensitive, free-form programming language that supports procedural, object-oriented, and generic programming. C++ is regarded as a **middle-level** language, as it comprises a combination of both high-level and low-level language features. C++ was developed by Bjarne Stroustrup starting in 1979 at Bell Labs in Murray Hill, New Jersey, as an enhancement to the C language and originally named C with Classes but later it was renamed C++ in 1983. C++ is a superset of C, and that virtually any legal C program is a legal C++ program.

Note: A programming language is said to use static typing when type checking is performed during compile-time as opposed to run-time.

Object oriented Programming is a programming methodology that associated data structure with a set of operators which act upon it. An instance of such an entity is known as object which is a combination or collection of data and code designed to emulate a physical or abstract entity. Each object has its own identity and distinguishes from each other. The fundamental idea behind an object-oriented language is to combine both data and the functions that operate on the data into a single unit. Such a unit is called an object. Examples of object oriented programming C++, Small talk, Java. There are two components of an object in C++.

I. Data Member
II. Member function

Structured programming and object-Oriented programming fundamentally differ in the following way: structured programming views the two core elements of any program- data and function as two separate entities whereas OOP views them as a single entity. Data members of an object can be accessed only by using the functions of the object. Therefore, the data is hidden and is safe from accidental alteration. Data and functions are stored into a single entity. The functions of an object are called the member functions.



These member functions are called method in other object-oriented languages like Small Talk.

Figure 1 Objected Oriented Programming

Relating to other paradigms (functional, data decomposition)

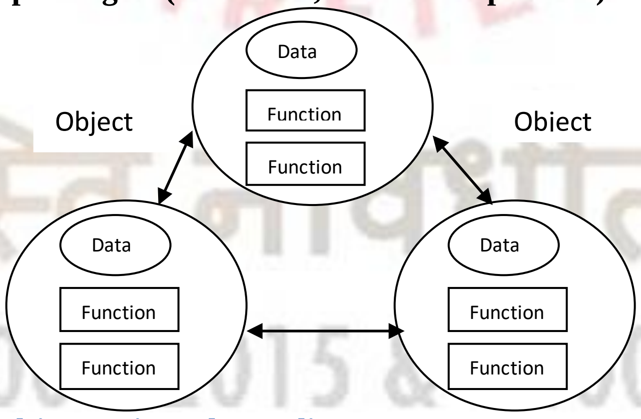


Figure 2 Object-Oriented paradigm

During program execution, the objects interact with each other by sending messages and receiving responses.

Features of Procedure oriented programming

Procedure programming is base on the algorithms. It best suited for the design of a knowledge base. The following are features of procedural programming.

Programs are organized in the form of subroutines and all data items are global.

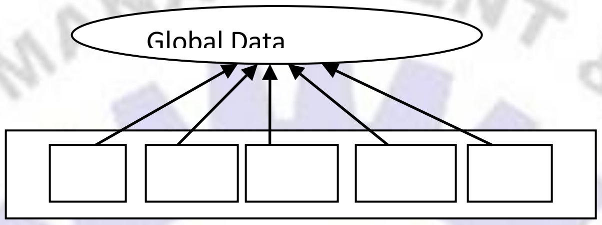
Program controls are through jumps and calls to subroutines.

Subroutines are abstracted to avoid repetitions

Suitable for medium sized software applications

Difficult to maintain and enhance the program code

Example FORTRAN and COBOL



Subprograms

Figure 3 Procedural Programmin

Basic Concepts of Object Oriented Programming

- **Objects**

Objects are the basic run time entities. This represents a person, a place, a bank A/C or any item that the program has to handle. They can represent user define data type

Programming problem is analysis I term of object and the nature of communication between them. Objects represents space in the memory and have an associate address like one record or structure in 'C'. When a program is executed the objects interacts by sending massages to one another each object contains data & code to manipulate the

- **Classes**

The entire set of data & code of an object can be made a user define data type with the help of a class. Objects are variables of the class. Once the class has been define we can create any number of objects belong to that class. A class is a collection of objects of similar type.

- **Encapsulation**

It is a mechanism that associates the code and the data it manipulates into a single unit and keeps them safe from external inference and misuse. It is supported by a construct called class. An instance of a class is known as an object which represents real world entity.

- **Data Abstraction**

It creates new data types that are well suited to an application to be programmed is known as data abstraction.

- **Inheritance**

  It provides the flexibility to reuse the existing code.
- **Multiple Inheritance**

  A class is derived from more than one base class is known as multiple inheritance.
- **Polymorphism**

  It allows a single name/operator to be associated with different operations depending on the type of data passed to it. It can be achieved by function overloading, operator overloading and dynamic binding.
- **Message Passing**

  Object oriented program consist of the set of object that communicate with each other object communicate by sending & receiving information in the form of message.

- **Dynamic binding**

  Binding refers to the linking of a processor call to the code to be executed I response to the call dynamic binding is known as late binding means that the code associated with a given procedure call is not known until the time of the call at run time. This is associated with polymorphism & inheritance.

Benefits of OOP

It emphasis on the data rather than algorithm.

Data and associated operations are unified into a single unit, thus the objects are grouped with common attributes, operation and semantics.

Program are designed around the data being operated, rather than operations themselves (data decomposition rather than algorithm decomposition)

Inheritance increases the reusability of code.

We can build programs from the standard working modules that communicate with one another, rather than having to start writing the code from scratch which happens procedure oriented approach. This leads to saving of development time and higher productivity.

Data hiding helps to secure programs that cannot be invaded by code in other parts of the program.

Multiple instances of object can use without any interference.

It is possible to map objects in the problem domain to those in the program.

The data-centered design approach enables us to capture more details of a model in implementable from.

Object oriented systems can be easily upgraded from small to large systems.

Message passing techniques for communication between objects makes the interface descriptions with external systems much simpler.

Software complexity can be easily managed.

Applications of OOP

OOP Provides Many Applications:-

1) **Real time Systems -** A real time system is a system that give output at given instant and its parameters changes at every time. A real time system is nothing but a dynamic system. Dynamic means the system that changes every moment based on input to the system. OOP approach is very useful for Real time system because code changing is very easy in OOP system and it leads toward dynamic behavior of OOP codes thus more suitable to real time system.

2) Simulation and Modeling- System modeling is another area where criteria for OOP approach is countable. Representing a system is very easy in OOP approach because OOP codes are very easy to understand and thus is proffered to represent a system in simpler form.

3) Hypertext and Hypermedia- Hypertext and hypermedia is another area where OOP approach is very useful. Its ease of using OOP codes that makes it suitable for various media approaches.

4) Decision support system- Decision support system is an example of Real time system that too very advance and complex system.

5) CAM/CAE/CAD System- Computer has wide use of OOP approach. This is due to time saving in writing OOP codes and dynamic behaviour of OOP codes.

6) Office Automation System- Automation system is just a part or type of real time system. Embedded systems make it easy to use OOP for automated system.

7) AI and expert system- It is mixed system having both hypermedia and real time system.

Difference between C and C++, cin, cout, new, delete operators

We can summarize the differences as follows:

• **Procedural Programming**

– Top down design

– create functions to do small tasks

– communicate by parameters and return values

• **Object Oriented Programming**

– design and represent objects

– determine relationships between objects

– determine attributes each object has

– determine behaviors each object will respond to

– create objects and send messages to them to use or manipulate their attributes

‘c’ Language	**C++ Language**
Emphasis is on action rather than data.	Emphasis is on data rather than procedures
subprograms or sub procedures are used	Objects are used.
Functions & data are separate	Functions that operate on the data of an object are typed together in the data structure.
Data can be access by external function	Data is hidden and can’t be access by external functions. Objects may communicate with each their through function.
New data & function for the whole application can’t be easily added whenever necessary.	New data & function can be easily added when ever necessary
Follows up-bottom approach in program designed	Follows bottom-up approach in program designed.

cin	To input the data
cout	Shows the output
new	To allocate memory to newly created variable
delete	To deallocate memory already assign to some variable

C++ Environment: Program development environment

Program development environment is a software application that provides comprehensive facilities to computer programmers for software development.

Language and the C++ language standards

Standard Libraries

Standard C++ consists of three important parts: The core language giving all the building blocks including variables, data types and literals etc.The C++ Standard Library giving a rich set of functions manipulating files, strings etc. The Standard Template Library (STL) giving a rich set of methods manipulating data structures etc.

The ANSI Standard

The ANSI standard is an attempt to ensure that C++ is portable -- that code you write for Microsoft's compiler will compile without errors, using a compiler on a Mac, Unix, a Windows box, or an Alpha. The ANSI standard has been stable for a while, and all the major C++ compiler manufacturers support the ANSI standard.

The Standard Function Library:

The standard function library is divided into the following categories:

I/O

String and character handling

Mathematical

Time, date, and localization

Dynamic allocation

Miscellaneous

Wide-character functions

The Object Oriented Class Library:

Standard C++ Object Oriented Library defines an extensive set of classes that provide support for a number of common activities, including I/O, strings, and numeric processing. This library includes following:

- The Standard C++ I/O Classes
- The String Class
- The Numeric Classes
- The STL Container Classes
- The STL Algorithms
- The STL Function Objects
- The STL iterators
- The STL Allocators
- The Localization library
- Exception Handling Classes

Miscellaneous Support Library

Input/Output with files

C++ provides the following classes to perform output and input of characters to/from files:

_ ofstream: Stream class to write on files

_ ifstream: Stream class to read from files

_ fstream: Stream class to both read and write from/to files.

These classes are derived directly or indirectly from the classes istream, and ostream.

Introduction to various C++ compilers

C++ is a huge language so much that it uses various sets of instructions from different parts to do its work. Some of these instructions come in computer files that you simply "put" in your program. These instructions or files are also called libraries.

#include iostream.h

#include "books.h"

When we include a library that came with C++, we enclose it between < and > as follows:

#include <iostream.h>

Following this same technique, we can add as many libraries as we can. Before adding a file, we will need to know what that file is and why we need it. This will mostly depend on our application. For example, we can include a library called stdio like this:

#include <iostream.h>

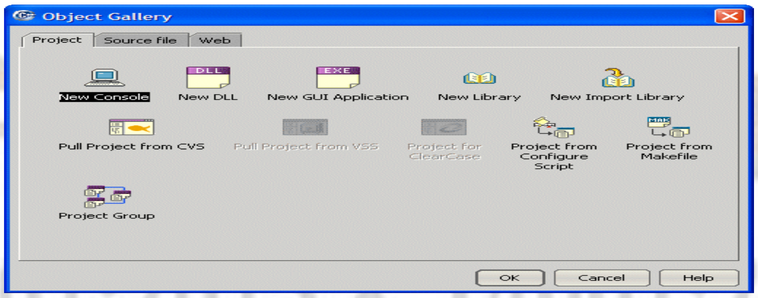
#include <stdio.h>

There are usually two kinds of libraries or files you will use in our programs: libraries that came with C++, and those that we write. To include our own library, we would enclose it between double quotes, like this

Testing the C++ program in Turbo C++/Borland C++/MicroSoft VC++/GNU C++ compile

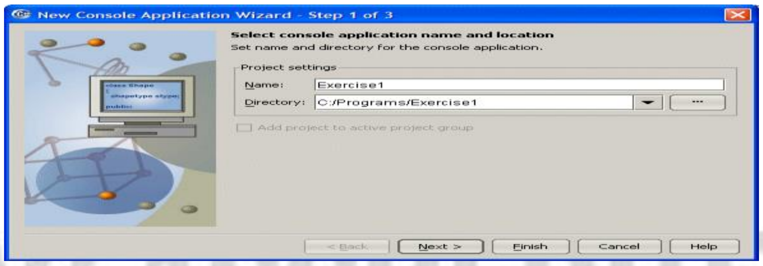
Borland C++BuilderX is a commercial programming environment developed by Borland. To help programmers, Borland published a free version, called Personal Edition, that we can download and use.

1. On the main menu of C++BuilderX, click File -> New...
2. In the Object Gallery dialog box, click New Console



3. Click OK
4. In the New Console Application - Step 1 of 3, enter the name of the new application

in the Name edit box. In this case, we can type Exercise1



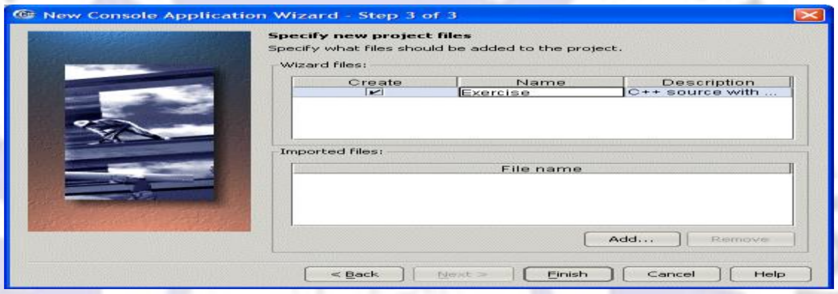
Select console application name and location
Set name and directory for the console application.
Project settings
Name: Exercise1
Directory: C:/Programs/Exercise1
Add project to active project group

5. Click Next

6. In the New Console Application Wizard - Step 2 of 3, accept all defaults and click Next

7. In the New Console Application Wizard - Step 3 of 3, click the check box under Create

8. Click Untitled1 and delete it to replace it with Exercise



Specify new project files
Specify what files should be added to the project.
Wizard files:

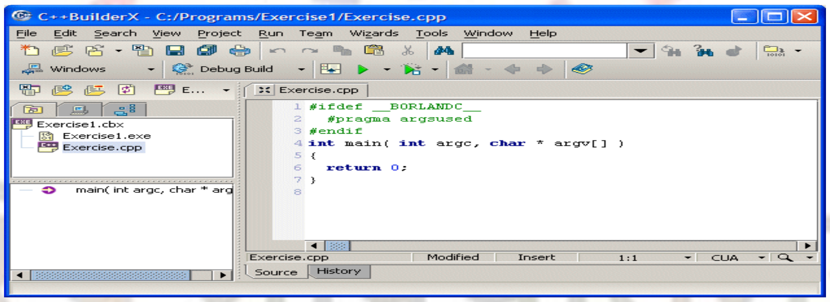
Create	Name	Description
✓	Exercise	C++ source with ...

Imported files:

File name

9. Click Finish

10. In the Project Content frame, double-click Exercise.cpp to display it in the right Frame


C++BuilderX - C:/Programs/Exercise1/Exercise.cpp
File Edit Search View Project Run Team Wizards Tools Window Help
Windows
Debug Build
E...
Exercise1.cbx
Exercise1.exe
Exercise.cpp
main(int argc, char * arg
Exercise.cpp
Modified
Insert
1:1
CUA
Source
History
```
#ifdef __BORLANDC__
  #pragma argsused
#endif
int main( int argc, char * argv[] )
{
  return 0;
}
```

11. To execute the application, on the main menu, click Run -> Run Project

Program structure

When we consider a C++ program, it can be defined as a collection of objects that communicate via invoking each other's methods. Let us now briefly look into what a class, object, methods, and instant variables mean.

□**Object -** Objects have states and behaviors. Example: A dog has states - color, name, breed as well as behaviors - wagging, barking, and eating.

An object is an instance of a class.

Class - A class can be defined as a template/blueprint that describes the behaviors/states that object of its type support.

Methods - A method is basically a behavior. A class can contain many methods. It is in methods where the logics are written, data is manipulated and all the actions are executed.

Instant Variables - Each object has its unique set of instant variables. An object's state is created by the values assigned to these instant variables.

C++ Program Structure:

Let us look at a simple code that would print the words *Hello World*.

```
#include <iostream>
using namespace std;
// main() is where program execution begins.
int main()
{
cout << "Hello World"; // prints Hello World
return 0;
}
```

Let us look at the various parts of the above program:

1. The C++ language defines several headers, which contain information that is either necessary or useful to your program. For this program, the header **<iostream>** is needed.

2. The line **using namespace std;** tells the compiler to use the std namespace. Namespaces are a relatively recent addition to C++.

3. The next line '**// main() is where program execution begins.'** is a single-line comment available in C++. Single-line comments begin with // and stop at the end of the line.

4. The line **int main()** is the main function where program execution begins.

5. The next line **cout << "This is my first C++ program.";** causes the message "This is my first C++ program" to be displayed on the screen.

6. The next line **return 0;** terminates main() function and causes it to return the value 0 to the calling process.

Compile & Execute C++ Program:

Let's look at how to save the file, compile and run the program. Please follow the steps given below:

1. Open a text editor and add the code as above.
2. Save the file as: hello.cpp
3. Open a command prompt and go to the directory where you saved the file.
4. Type 'g++ hello.cpp' and press enter to compile your code. If there are no errors in your code the command prompt will take you to the next line and would generate a.out executable file.
5. Now, type 'a.out' to run your program.
6. You will be able to see ' Hello World ' printed on the window.

Comments in C++

Program comments are explanatory statements that you can include in the C++ code. These comments help anyone reading the source code. All programming languages allow for some form of comments. C++ supports single-line and multi-line comments. All characters available inside any comment are ignored by C++ compiler.

C++ comments start with /* and end with */. For example:

```
/* This is a comment */
/* C++ comments can also
* span multiple lines
*/
```

A comment can also start with //, extending to the end of the line. For example:

```
#include <iostream>
using namespace std;
main()
{
cout << "Hello World"; // prints Hello World
return 0;
}
```

When the above code is compiled, it will ignore **// prints Hello World** and final executable will produce the following result:

```
Hello World
```

All C++ programs are composed of following two fundamental elements:

Program statements (code): This is the part of a program that performs actions and they are called functions. Program data: The data is the information of the program which affected by the program functions.

Encapsulation is an Object Oriented Programming concept that binds together the data and functions that manipulate the data, and that keeps both safe from outside interference and misuse.

Data encapsulation led to the important OOP concept of data hiding. Data encapsulation is a mechanism of bundling the data, and the functions that use them and data abstraction is a mechanism of exposing only the interfaces and hiding the implementation details from the user.

C++ supports the properties of encapsulation and data hiding through the creation of user defined types, called classes.

Information hiding

Data abstraction refers to, providing only essential information to the outside word and hiding their background details to represent the needed information in program without presenting the details. Data abstraction is a programming (and design) technique that relies on the separation of interface and implementation. C++ classes provide great level of data abstraction. They provide sufficient public methods to the outside world to play with the functionality of the object and to manipulate object data to state without actually knowing how class has been implemented internally.

For example, our program can make a call to the sort() function without knowing what algorithm the function actually uses to sort the given values. In fact, the underlying implementation of the sorting functionality could change between releases of the library, and as long as the interface stays the same, your function call will still work.

Benefits of Data Abstraction*:*

Data abstraction provides two important advantages:

• Class internals are protected from inadvertent user-level errors, which might corrupt the state of the object.

• The class implementation may evolve over time in response to changing requirements or bug reports without requiring change in user-level code.

By defining data members only in the private section of the class, the class author is free to make changes in the data. If the implementation changes, only the class code needs to be examined to see what affect the change may have. If data are public, then any function that directly accesses the data members of the old representation might be broken.

Abstract data Type

An abstract data type is simply an encapsulation that includes only the data representation of one specific data type and the subprograms that provide the operations for that type. It is a data type that satisfies two conditions:-

1) The representation, or definition, of the type and the operations are contained in a single syntactic unit.

2) The representation of objects of the type is hidden from the program units that use the type, so only direct operations possible on those objects are those provided in the type's definition

Classes

When we define a class, we define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

A class definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. A class enclosed both the data and functions that operate on the data into a single unit shown in fig 4.

Defining variables of a class type is known as *instantiation* and such variables are called *objects*.

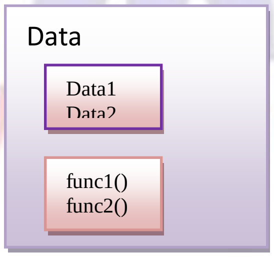


Figure 4

In fig 4 a class called data is created and it holds data members and data functions.

C++ Class:

When we define a class, we define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object.

A class definition starts with the keyword class followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations.

For example we defined the Box data type using the keyword class as follows:

```
Class data

{

Public:

Int  roll_no;

char name[20];
```

Where

- Data is class a name
- Public is a keyword. It determines the access attributes of the members of the class that follow it. A public member can be accessed from outside the class anywhere within the scope of the class object.
- roll_no is a variable
- name[20] is a variable.

C++ Objects

A class provides the blueprints for objects, so basically an object is created from a class. We declare objects of a class with exactly the same sort of declaration that we declare variables of basic types. Following statements declare two objects of class Box:

```
St1  //object 1

St2  // object2
```

Member /function In C++

A function is a group of statements that together perform a task. Every C++ program has at least one function which is main(), and all the most trivial programs can define additional functions

The general form of a C++ function definition is as follows:

return_type function_name(parameter list)

{

```
body of the function
}
```

A C++ function definition consists of a function header and a function body. Here are all the parts of a function:

• **Return Type:** A function may return a value. The return_type is the data type of the value the function returns. Some functions perform the desired operations without returning a value. In this case, the return_type is the keyword void.

• **Function Name:** This is the actual name of the function. The function name and the parameter list together constitute the function signature.

• **Parameters:** A parameter is like a placeholder. When a function is invoked, we pass a value to the parameter. This value is referred to as actual parameter or argument. The parameter list refers to the type, order, and number of the parameters of a function. Parameters are optional; that is, a function may contain no parameters.

• **Function Body**: The function body contains a collection of statements that define what the function does. The public data members of objects of a class can be accessed using the direct member access operator (.). Let us try the following example to make the things clear:

```
#include <iostream>
using namespace std;



class Box
{
public:
double length; // Length of a box
double breadth; // Breadth of a box
double height; // Height of a box
};
int main( )
{
Box Box1; // Declare Box1 of type Box
Box Box2; // Declare Box2 of type Box
double volume = 0.0; // Store the volume of a box here
// box 1 specification
Box1.height = 5.0;
Box1.length = 6.0;
Box1.breadth = 7.0;
// box 2 specification
Box2.height = 10.0;
Box2.length = 12.0;
Box2.breadth = 13.0;
// volume of box 1
volume = Box1.height * Box1.length * Box1.breadth;
cout << "Volume of Box1 : " << volume <<endl;
// volume of box 2
volume = Box2.height * Box2.length * Box2.breadth;
cout << "Volume of Box2 : " << volume <<endl;
return 0;
}
```

When the above code is compiled and executed, it produces the following result:

```
Volume of Box1 : 210
Volume of Box2 : 1560
```

Defining attributes in C++ Programming

The first part of the class is the private part, where the attributes of the class are listed. The attributes of a class are defined in a very similar way to the way in which variables are defined. The only difference is that they cannot be given an initial value. The format of the declaration is :

```
<type> <identifier>;
```

For example:

```
string name;
int age;
```

C++ class declaration, (references, this pointer)

A class is an expanded concept of a data structure: instead of holding only data, it can hold both data and functions. An object is an instantiation of a class. In terms of variables, a class would be the type, and an object would be the variable. Classes are generally **declared** using the keyword class, with the following format:

```
class class_name {
access_specifier_1:
```

member1;
access_specifier_2:
member2;
...
} object_names;

Where

class_name is a valid identifier for the class,
object_names is an optional list of names for objects of this class. The body of the declaration can contain members that can be either data or function declarations, or optionally access specifiers.

Encapsulation

The wrapping up of data and function into a single unit (called class) is known as encapsulation. Data encapsulation is the most striking feature of a class. The data is not accessible to the outside world, and only those functions which are wrapped in the class can access it. These functions provide the interface between the object's data and the program. This insulation of the data from direct access by the program is called *data hiding* or *information hiding.*

Object & classes, attributes, Methods

The main purpose of C++ programming is to add object orientation to the C programming language and classes are the central feature of C++ that supports object-oriented programming and are often called user-defined types. A class is used to specify the form of an object and it combines data representation and methods for manipulating that data into one neat package.

The data and functions within a class are called members of the class.

C++ Class Definitions

When you define a class, you define a blueprint for a data type. This doesn't actually define any data, but it does define what the class name means, that is, what an object of the class will consist of and what operations can be performed on such an object. A class definition starts with the keyword **class** followed by the class name; and the class body, enclosed by a pair of curly braces. A class definition must be followed either by a semicolon or a list of declarations. For example, we define the Box data type using the keyword **class** as follows:

This pointer

Function Overloading

In C++, two or more functions can be given the same name and provide unique signature.

Example

```
swap(int, int)
swap(float,float)
```

In this example a method is used with same name but parameters are changed.

When we call an overloaded function or operator, the compiler determines the most appropriate definition to use by comparing the argument types you used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called overload resolution. We can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. We cannot overload function declarations that differ only by return type

Constructors and destructors

A class constructor is a special member function of a class that is executed whenever we create new objects of that class. A constructor will have exact same name as the class and it does not have any return type at all, not even void. Constructors can be very useful for setting initial values for certain member variables.

Example

```
Class student
        {
int m,n;
Public:
        student(void);
};
studebt::student(void)
{
        M=0; n=0;
}
```

Characteristics of constructor

- It always declares in public section.
- They invoked automatically when the objects are created.
- No return type.

- Cannot inherit and cannot refer their addresses.
- An object with a constructor cannot be used as a member of a union.
- ‘New’ and ‘Delete’ operators are called implicitly when memory allocation is required.

Types of Constructor

- **copy constructor**
- **parameterized constructor**
- **Dynamic Constructor**

The Class Destructor:

A destructor is a special member function of a class that is executed whenever an object of it's class goes out of scope or whenever the delete expression is applied to a pointer to the object of that class. A destructor will have exact same name as the class prefixed with a tilde (~) and it can neither return a value nor can it take any parameters. Destructor can be very useful for releasing resources before coming out of the program like closing files, releasing memories etc.

Instantiation of objects

Instantiation of a class literally means creating an instance of a class. This is the process of allocating memory for an object that we can use in your program.

```
class class_name {




access_specifier_1:
member1;
access_specifier_2:
member2;
} object_names;
```

Default parameter value

A default parameter is a function parameter that has a default value provided to it. If the user does not supply a value for this parameter, the default value will be used. If the user does supply a value for the default parameter, the user-supplied value is used.

```
void PrintValues(int nValue1, int nValue2=10)
{
using namespace std;
cout << "1st value: " << nValue1 << endl;
cout << "2nd value: " << nValue2 << endl;
}
```

```
int main()
{
PrintValues(1); // nValue2 will use default parameter of 10
PrintValues(3, 4); // override default value for nValue2
}
```

This program produces the following output:

```
1st value: 1
2nd value: 10
1st value: 3
2nd value: 4
```

In the first function call, the caller did not supply an argument for nValue2, so the function used the default value of 10. In the second call, the caller did supply a value for nValue2, so the user-Supplied value was used. Default parameters are an excellent option when the function needs a value that the user may or may not want to override. For example, here are a few function prototypes for which default parameters might be commonly used:

References

A reference variable is an alias, that is, another name for an already existing variable. Once a reference is initialized with a variable, either the variable name or the reference name may be used to refer to the variable.

References vs Pointers

- References are often confused with pointers but three major differences between references and pointers are:
- You cannot have NULL references. You must always be able to assume that a reference is connected to a legitimate piece of storage.
- Once a reference is initialized to an object, it cannot be changed to refer to another object. Pointers can be pointed to another object at any time.
- A reference must be initialized when it is created. Pointers can be initialized at any time.

Dynamic Memory Allocation

Until now, in all our programs, we have only had as much memory available as we declared for our variables, having the size of all of them to be determined in the source code, before the execution of the program. But, what if we need a variable amount of memory that can only be determined during runtime. For example, in the case that we need some user input to determine the necessary amount of memory space. For that we need dynamic memory, for which C++ integrates the operators **new and delete.**

- **Operators new and new[]**

  In order to request dynamic memory we use the operator new. New is followed by a data type specifier and -if a sequence of more than one element is required- the number of these within brackets []. It returns a pointer to the beginning of the new block of memory allocated. Its form is:

  ```
  pointer = new type
  pointer = new type [number_of_elements]
  ```

- **Operators delete and delete[]**

  Since the necessity of dynamic memory is usually limited to specific moments within a program, once it is no longer needed it should be freed so that the memory becomes available again for other requests of dynamic memory.

Garbage Collection

It is a process that many Operating Systems and some applications, especially services perform.

The purpose of Garbage Collection is to recover or reorganize system resources such as available blocks of RAM to prevent failure due to resource starvation. Many applications allocate blocks of RAM for storing variables. When the blocks of RAM are no longer required they are returned to the application. The addresses of the blocks are usually stored on a free list and when the application later requests more RAM, it checks the free list first. The effect of this over a period of time is to split RAM into smaller and smaller blocks. Eventually an application will request a block that is too big and fail. Garbage Collection merges all the free blocks into one large block. In older languages like C++ programmers had to write their own Garbage Collection routines but newer languages like C# provide this.

Abstract class

An interface describes the behavior or capabilities of a C++ class without committing to a particular implementation of that class. The C++ interfaces are implemented using abstract classes and these abstract classes should not be confused with data abstraction which is a concept of keeping implementation detail separate from associated data. The purpose of an abstract class (often referred to as an ABC) is to provide an appropriate base class from which other classes can inherit. Abstract classes cannot be used to instantiate objects and serves only as an interface. Attempting to instantiate an object of an abstract class causes a compilation error.

Metaclass: A class whose instances are classes.

UNIT- III

Inheritance and Polymorphism

Inheritance

One of the most important concepts in object-oriented programming is that of inheritance. Inheritance allows us to define a class in terms of another class, which makes it easier to create and maintain an application. This also provides an opportunity to reuse the code functionality and fast implementation time. When creating a class, instead of writing completely new data members and member functions, the programmer can designate that the new class should inherit the members of an existing class. This existing class is called the **base** class, and the new class is referred to as the **derived** class. The idea of inheritance implements the **is a** relationship. For example, mammal IS-A animal, dog IS-A mammal hence dog IS-A animal as well and so on.

Base & Derived Classes

A class can be derived from more than one classes, which means it can inherit data and functions from multiple base classes. To define a derived class, we use a class derivation list to specify the base class (es).

A class derivation list names one or more base classes and has the form: class derived-class: access-specifier base-class. Where access-specifier is one of **public, protected,** or **private**, and base-class

is the name of a previously defined class. If the access-specifier is not used, then it is private by default. Consider a base class **Shape** and its derived class **Rectangle** as follows:

```
#include <iostream>
using namespace std;
// Base class
class Shape
{
public:
void setWidth(int w)
{
width = w;
}
void setHeight(int h)
{
height = h;
}
protected:
int width;
int height;
};
// Derived class
class Rectangle: public Shape
{
public:
int getArea()
{
return (width * height);
}
};
int main(void)



{
Rectangle Rect;
Rect.setWidth(5);
Rect.setHeight(7);
// Print the area of the object.
cout << "Total area: " << Rect.getArea() << endl;
return 0;
}
```

Access Control and Inheritance

A derived class can access all the non-private members of its base class. Thus base-class members that should not be accessible to the member functions of derived classes should be declared private in the base class. We can summarize the different access types according to - who can access them, in the following way:

Access	**public**	**protected**	**private**
Same class	yes	yes	yes
Derived classes	yes	yes	no

Outside classes	yes	no	no

Class hierarchy

Derivation

Class Access Modifiers

Data hiding is one of the important features of Object Oriented Programming which allows preventing the functions of a program to access directly the internal representation of a class type. The access restriction to the class members is specified by the labeled **public, private,** and **protected** sections within the class body. The keywords public, private, and protected are called access specifiers.

A class can have multiple public, protected, or private labeled sections. Each section remains in effect until either another section label or the closing right brace of the class body is seen. The default access for members and classes is private.

Public Members

A **public** member is accessible from anywhere outside the class but within a program. You can set and get the value of public variables without any member function as shown in the following example:

```
#include <iostream>
using namespace std;
class Line
{
public:
        double length;
        void setLength( double len );
        double getLength( void );
        };
```

Private

A **private** member variable or function cannot be accessed, or even viewed from outside the class. Only the class and friend functions can access private members. By default all the members of a class would be private, for example in the following class **width** is a private member, which means until you label a member, it will be assumed a private member:

```
class Box
{
```

double width;

public:

double length;

void setWidth(double wid);

double getWidth(void);

};

Practically, we define data in private section and related functions in public section so that they can be called from outside of the class as shown in the following program.

Protected

A **protected** member variable or function is very similar to a private member but it provided one additional benefit that they can be accessed in child classes which are called derived classes. You will learn derived classes and inheritance in next chapter. For now you can check following example where I have derived one child class **SmallBox** from a parent class **Box**.

Following example is similar to above example and here **width** member will be accessible by any member function of its derived class SmallBox.

#include <iostream>

using namespace std;

class Box

{

protected:

double width;

};

Types of Inheritance

- Single Inheritance
- Multilevel Inheritance
- Multiple Inheritance

Compositions:

- Typically use normal member variables
- Can use pointer values if the composition class automatically handles allocation/deallocation
- Responsible for creation/destruction of subclasses

Aggregations:

- Typically use pointer variables that point to an object that lives outside the scope of the aggregate class
- Can use reference values that point to an object that lives outside the scope of the aggregate class
- Not responsible for creating/destroying subclasses

Polymorphism

The word **polymorphism** means having many forms. Typically, polymorphism occurs when there is a hierarchy of classes and they are related by inheritance. C++ polymorphism means that a call to a member function will cause a different function to be executed depending on the type of object that invokes the function.

Consider the following example where a base class has been derived by other two classes:

```
#include <iostream>
using namespace std;
class Shape {
protected:
int width, height;
public:




Shape( int a=0, int b=0)
{
width = a;
height = b;
}
int area()
{
cout << "Parent class area :" <<endl;
return 0;
}
};
class Rectangle: public Shape{
public:
Rectangle( int a=0, int b=0):Shape(a, b) { }
int area ()
{
```

```
cout << "Rectangle class area :" <<endl;
return (width * height);
}
};
class Triangle: public Shape{
public:
Triangle( int a=0, int b=0):Shape(a, b) { }
int area ()
{
cout << "Triangle class area :" <<endl;
return (width * height / 2);
}
};
// Main function for the program
int main( )
{
Shape *shape;



Rectangle rec(10,7);
Triangle tri(10,5);
// store the address of Rectangle
shape = &rec;
// call rectangle area.
shape->area();
// store the address of Triangle
shape = &tri;
// call triangle area.
shape->area();
return 0;
}
```

Operator overloading

C++ allows you to specify more than one definition for a **function** name or an **operator** in the same scope, which is called **function overloading** and **operator overloading** respectively. An overloaded declaration is a declaration that is declared with the same name as a previously declared declaration in the same scope, except that both declarations have different arguments and obviously different definition (implementation). When you call an overloaded **function** or **operator**, the compiler determines the most appropriate definition to use, by comparing the argument types you have used to call the function or operator with the parameter types specified in the definitions. The process of selecting the most appropriate overloaded function or operator is called **overload resolution**.

Function Overloading in C++

You can have multiple definitions for the same function name in the same scope. The definition of the function must differ from each other by the types and/or the number of arguments in the argument list. You cannot overload function declarations that differ only by return type.

Following is the example where same function **print()** is being used to print
different data types:

```
#include <iostream>
using namespace std;

class printData
{
public:
void print(int i) {
cout << "Printing int: " << i << endl;
}
void print(double f) {
cout << "Printing float: " << f << endl;
}
void print(char* c) {
cout << "Printing character: " << c << endl;
}
};
```

```
int main(void)
{
```

```
printData pd;
// Call print to print integer
pd.print(5);
// Call print to print float
pd.print(500.263);
// Call print to print character
pd.print("Hello C++");
return 0;
}
```

Parametric polymorphism:

If all code is written without mention of any specific type and thus can be used transparently with any number of new types it is called parametric polymorphism.

Polymorphism by parameter

Parametric polymorphism is obtained when a function works uniformly on a range of types; these types normally exhibit some common structure. Ad-hoc polymorphism is obtained when a function works, or appears to work, on several different types (which may not exhibit a common structure) and may behave in unrelated ways for each type."

Virtual Function

A **virtual** function is a function in a base class that is declared using the keyword **virtual**. Defining in a base class a virtual function, with another version in a derived class, signals to the compiler that we don't want static linkage for this function. What we do want is the selection of the function to be called at any given point in the program to be based on the kind of object for which it is called. This sort of operation is referred to as **dynamic linkage**, or **late binding**.

Pure Virtual Functions

It is possible that you want to include a virtual function in a base class so that it may be redefined in a derived class to suit the objects of that class, but that there is no meaningful definition you could give for the function in the base class. We can change the virtual function area() in the base class to the following:

```
class Shape {
protected:
int width, height;
public:
```

```
Shape( int a=0, int b=0)
{
width = a;
height = b;
}
// pure virtual function
virtual int area() = 0;
};
```

Early v/s Late Binding

The reason for the incorrect output is that the call of the function area() is being set once by the compiler as the version defined in the base class. This is called **static resolution** of the function call, or **static linkage** - the function call is fixed before the program is executed. This is also sometimes called **early binding** because the area() function is set during the compilation of the program. A **virtual** function is a function in a base class that is declared using the keyword **virtual**. Defining in a base class a virtual function, with another version in a derived class, signals to the compiler that we don't want static linkage for this function. What we do want is the selection of the function to be called at any given point in the program to be based on the kind of object for which it is called. This sort of operation is referred to as **dynamic linkage**, or **late binding**.

4.1 Generic Programming

4.1.1Introduction- In the simplest definition, **generic programming** is a style of computer programming in whichalgorithms are written in terms of to-be-specified-later types that are then *instantiated* when needed for specific types provided as parameters C++ provides unique abilities to express the ideas of Generic Programming through *templates*.

Templates provide a form of parametric polymorphism that allows the expression of generic algorithms and data structures. The instantiation mechanism of C++ templates insures that when a generic algorithm or data structure is used, a fully-optimized and specialized version will be created and tailored for that particular use, allowing generic algorithms to be as efficient as their non-generic counterparts.

4.1.2 Templates

Templates are the foundation of generic programming, which involves writing code in a way that is independent of any particular type. A template is a blueprint or formula for creating a generic class or a function. The library containers like iterators and algorithms are examples of generic programming and have been developed using template concept. There is a single definition of each container, such as **vector**, but we can define many different kinds of vectors for example, **vector <int>** or **vector <string>**. You can use templates to define functions as well as classes, let us see how they work:

- **Function Template**

  The general form of a template function definition is shown here:

  ```
  template <class type> ret-type func-name(parameter list)
  {
  // body of function
  }
  ```

- **Template functions**

  Function templates are special functions that can operate with *generic types*. This allows us to create a function template whose functionality can be adapted to more than one type or class without repeating the entire code for each type.

  In C++ this can be achieved using *template parameters*. A template parameter is a special kind of parameter that can be used to pass a type as argument: just like regular function parameters can be used to pass values to a function, template parameters allow to pass also types to a function. These function templates can use these parameters as if they were any other regular type.

  The format for declaring function templates with type parameters is:

  ```
  template<class identifier>function_declaration;
  template <typename identifier> function_declaration;
  ```

- **Overloading of template functions**

  You may overload a function template either by a non-template function or by another function template. If you call the name of an overloaded function template, the compiler will try to deduce its template arguments and check its explicitly declared template arguments. If successful, it will instantiate a function template specialization, then add this specialization to the set of *candidate functions* used in overload resolution. The compiler proceeds with overload resolution, choosing the most appropriate function from the set of candidate functions. Non-template functions take precedence over template functions. The following example describes this:

```
#include <iostream>
                              using namespace std;
template<class T> void f(T x, T y) { cout << "Template" << endl; }
void f(int w, int z) { cout << "Non-template" << endl; }
```

4.1.3 Overriding inheritance methods

When a method in a derived class has the same name, arguments in numbers and types and the same return type the method says to override that base class method. The base class has an overloaded method with the same name as the overridden method, that method says to be hided and cannot be accessed by the derived class without the special class (baseclass::) notation.

4.2 Files and Exception Handling

4.2.1 Persistant objects

A persistent object can live after the program which created it has stopped. Persistent objects can even outlive different versions of the creating program, can outlive the disk system, the operating system, or even the hardware on which the OS was running when they were created. The challenge with persistent objects is to effectively store their member function code out on secondary storage along with their data bits. This is non-trivial when you have to do it yourself.

In C++, you have to do it yourself. C++ databases can help hide the mechanism for all this.

4.2.2Streams and files

The C++ standard libraries provide an extensive set of input/output capabilities. C++ I/O occurs in streams, which are sequences of bytes. If bytes flow from a device likes a keyboard, a disk drive, or a network connection etc. to main memory, this is called input operation and if bytes flow from main memory to a device likes a display screen, a printer, a disk drive, or a network connection, etc, this is called output operation. Of course, it's also possible to open files or other I/O sources) explicitly. We can open files using the function fopen; certain systems may also provide specialized ways of opening streams connected to I/O devices or set up in more exotic ways. A successful call to fopen returns a pointer of type FILE *, that is, ``pointer to FILE,'' where FILE is a special type defined by
<stdio.h>. A FILE * (also called ``file pointer'') is the handle by which we refer to an I/O stream in

C++ I/O Library Header Files:

There are following header files important to C++ programs:

Header File Function and Description

<iostream> This file defines the cin, cout, cerr and clog objects, which correspond to the standard input stream, the standard output stream, the un-buffered standard error stream and the buffered standard error stream, respectively.

<iomanip> This file declares services useful for performing formatted I/O with so-called parameterized stream manipulators, such as setw and set precision.

<fstream> This file declares services for user-controlled file processing. We will discuss about it in detail in File and Stream related chapter.

4.2.3The standard output stream (cout):

The predefined object cout is an instance of ostream class. The cout object is said to be "connected to" the standard output device, which usually is the display screen. The cout is used in conjunction with the stream insertion operator, which is written as << .

The standard input stream (cin):

The predefined object cin is an instance of istream class. The cin object is said to be attached to the standard input device, which usually is the keyboard. The cin is used in conjunction with the stream extraction operator, which is written as >>

The standard error stream (cerr):

The predefined object cerr is an instance of ostream class. The cerr object is said to be attached to the standard error device, which is also a display screen but the object cerr is un-buffered and each stream insertion to cerr causes its output to appear immediately

4.2.4Namespaces

Consider a situation, when we have two persons with the same name, Zara, in the same class. Whenever we need to differentiate them definitely we would have to use some additional information along with their name, like either the area, if they live in different area or their mother’s or father’s name, etc. Same situation can arise in your C++ applications. For example, you might be writing some code that has a function called xyz() and there is another library available which is also having same function xyz(). Now the compiler has no way of knowing which version of xyz() function you are referring to within your code. A **namespace** is designed to overcome this difficulty and is used as additional information to differentiate similar functions, classes, variables etc. with the same name available in different libraries. Using namespace, you can define the context in which names are defined. In essence, a namespace defines a scope. Defining a Namespace efinition begins with the keyword **namespace** followed by the namespace name as follows:

namespace namespace_name {

```
// code declarations
}
```

To call the namespace-enabled version of either function or variable, prepend (::) the namespace name as follows:

```
namespace namespace_name {
// code declarations
}
```

Let us see how namespace scope the entities including variable and functions:

```
#include <iostream>
using namespace std;
// first name space
namespace first_space{
void func(){
cout << "Inside first_space" << endl;
}
}
```

4.2.5 The basic stream classes:

C++ predefined streams,

C++ provides the following predefined streams:

cin The standard input stream

cout The standard output stream

cerr The standard error stream, unit-buffered such that characters sent to this stream are flushed on each output operation

clog The buffered error stream

All predefined streams are tied to cout. When you use cin, cerr, or clog, cout gets flushed sending the contents of cout to the ultimate consumer.

C++ standard streams create all I/O to I/O streams:

- Input to cin comes from stdin (unless cin is redirected)

- cout output goes to stdout (unless cout is redirected)
- cerr output goes to stderr (unit-buffered) (unless cerr is redirected)

• clog output goes to stderr (unless clog is redirected)

4.2.6Types of Exception

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero. There are many different derived exception types .Here, we look at various exception types

• ArgumentException

• ArgumentNullException

• ArgumentOutOfRangeException

• ArrayTypeMismatchException

• DirectoryNotFoundException

• DivideByZeroException

• FileNotFoundException

• FormatException

• IndexOutOfRangeException

• InvalidCastException

• InvalidOperationException

• IOException

• KeyNotFoundException

• NullReferenceException

• OutOfMemoryException

• OverflowException

• StackOverflowException

• TypeInitializationException

- **Error handling during files operations**

we perform the file operation without any knowledge of failure or success of the function open( ) that opens the file. There are many reasons and they may result in error during read/write operation of the program.

(1) An attempt to read a file which does not exist.

(2) The file name specified for opening a new file may already exist.

(3) An attempt to read contents of file when file pointer is at the end of file.

(4) Insufficient disk space.

(5) Invalid file name specified by the programmer.

(6) An effort to write data to the file that is opened in read only mode.

(7) A file opened may already be opened by another program.

(8) An attempt to open read only file for writing operation.

(9) Device error.

- **ERROR HANDLING FUNCTIONS**

  eof() Return true if end-of-file is encountered while reading

  fail() Return true when an input or output operation has failed .

  bad() Return true if an invalid operation is attempted or any unrecoverable error has occured

  good() Return true if no error has occured

  These functions may be used in appropriate places in a program to locate the status of a file stream and therefore to take the necessary corrective measures**.**

4.2.7 Command Line Arguments.

In C++ it is possible to accept command line arguments. Command-line arguments are given after the name of a program in command-line operating systems like DOS or Linux, and are passed in to the program from the operating system. To use command line arguments in your program, you must first understand the full declaration of the main function, which previously has accepted no arguments.

In fact, main can actually accept two arguments: one argument is number of command line arguments, and the other argument is a full list of all of the command line arguments.

4.2.8 Error handling during file operations

An exception is a problem that arises during the execution of a program. A C++ exception is a response to an exceptional circumstance that arises while a program is running, such as an attempt to divide by zero.

Exceptions provide a way to transfer control from one part of a program to another. C++ exception handling is built upon three keywords: **try, catch,** and **throw**.

- **throw:** A program throws an exception when a problem shows up. This is done using a **throw** keyword.
- **catch:** A program catches an exception with an exception handler at the place in a program where you want to handle the problem.

- The **catch** keyword indicates the catching of an exception.
- **try:** A **try** block identifies a block of code for which particular exceptions will be activated. It is followed by one or more catch blocks.

Assuming a block will raise an exception, a method catches an exception using a combination of the **try** and **catch** keywords. A try/catch block is placed around the code that might generate an exception. Code within a try/catch block is referred to as protected code, and the syntax for using try/catch is as follows:

Types of Exception, Catching and Handling Exceptions.

```
try
{
// protected code
}catch( ExceptionName e1 )
{
// catch block
}catch( ExceptionName e2 )
{
// catch block
}catch( ExceptionName eN )
{
// catch block
}
```

You can list down multiple **catch** statements to catch different type of exceptions in case your **try** block raises more than one exception in different situations.

Catching Exceptions

The **catch** block following the **try** block catches any exception. You can specify what type of exception you want to catch and this is determined by the exception declaration that appears in parentheses following the keyword catch.

```
try
{
// protected code
}catch( ExceptionName e )
{
// code to handle ExceptionName exception
}
```

Above code will catch an exception of **Exception Name** type. If you want to specify that a catch block should handle any type of exception that is thrown in a try block, you must put an ellipsis, ..., between the parentheses enclosing the
exception declaration as follows:

```
try
{
// protected code
}catch(...)
{
// code to handle any exception
}
```