KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
Coimbatore-21

SYLLABUS

DEPARTMENT OF CS, CA & IT

SUBJECT NAME: .NET PROGRAMMING **SUBJECT CODE: 17CAP502**

SEMESTER: V **CLASS: III MCA**

Scope: This course provides functional understanding of the components of the .NET Framework and the common language runtime (CLR) and know how COM components. Also provides better knowledge of Architecture and Basic Concepts required for building database application with ADO.NET Programming. It also provides an platform to apply all the window controls.

Objective:

- Become a good .NET programmer.
- Know how COM components and the .NET Framework interoperate with each other.
- Identify and use the classes and namespaces in the .NET Framework class library.
- Build WEB Applications using Microsoft ASP.NET programming.

UNIT I

Introduction: Getting Started With VB.NET: The Integrated Development Environment-IDE Components-Environment Options.Visual Basic: The Language Variables-Constants-Arrays – Variables as Objects-Flow Control Statements.Working with forms: The appearance of Forms-Loading and Showing Forms-Designing Menus.

UNIT II

Basic Windows Controls: Textbox Control- ListBox, CheckedListBox-Scrollbar and TrackBar Controls- More Windows Control-The Common Dialog Controls-The Rich TextBox Control - Handling Strings, characters and Dates. The TreeView and ListView Controls: Examining the Advanced Controls-The TreeView Control-The ListView Control

UNIT III

The Multiple Document Interface-Databases: Architecture and Basic Concepts-Building Database Application with ADO.NET-Programming with ADO.NET

UNIT IV

Goal of ASP.NET –ASP.NET Web Server Control-Validation Server Controls-Themes and Skins -Content Page Holder

UNIT V

Data Binding in ASP.Net: Data source Controls – Configuring data source control caching – storing connection information-Using Bound list controls with Data Source Controls – Other Data bound Controls-Data Management with ADO.Net

SUGGESTED READINGS

1. Evangelos Petroutsos, Mastering Visual Basic.Net, BPB Publications, New Delhi.
2. Bill Evjen, Scott Hanselman , Devin Rader, Farhan Muhammad and S.Srinivasa Sivakumar (2006), Professional ASP.net 2.0, Special Edition.
3. Dave Mercer, ASP.Net Beginner's Gudie (2003), 2nd Edition McGraw Hill, New Delhi.
4. Duncan Mackenzie Kent Sharkey (2006), Sams Teach yourself Visual Basic.JNet, 1st Edtion, McGraw Hill, NewDelhi.
5. Shirish Chavan. (2007), Visual Basic.Net, 1st Edition, Pearson Education, New Delhi.

WEB SITES

1. www.microsoft.com/NET/
2. www.en.wikipedia.org/wiki/.net
3. www.w3schools.com/ngws/default.asp
4. www.vbtutot.com

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
Coimbatore-21

LECTURE PLAN

DEPARTMENT OF CS, CA & IT

STAFF NAME: K.RAMESH

SUBJECT NAME: .NET PROGRAMMING **SUBJECT CODE: 17CAP502**

SEMESTER: V **CLASS: III MCA**

S.No.	Lecture Duration (Period)	Topics to be Covered	Support Materials
		Unit – I	
1.	1	Introduction: Getting Stated with VB.Net.	T1: 3-12
2.	1	The integrated development Environment start page, Project type	T1: 12-24
3.	1	IDE components: Menu, Toolbox, Solution Explorer, properties window	T1: 25-32
4.	1	Environment options	T1: 79-121 R1:72-80
5.	1	Visual Basic: The Language Variables, Declaration, Scope, Constants.	T1: 122-148
6.	1	Arrays, Variables as object	T1: 2185-207, J1
7.	1	Flow Control Statement	T1:207-219
8.	1	Working with Forms: The Appearance, Properties of Form Control Loading and Showing Forms, Startup form Controlling Designing Menus, Menu Editor, properties	T1: 219-231 W1
9.	1	Recapitulation and Discussion of important questions	
		Total No. of Hours Planned for Unit-I	**9**

Unit – II			
1.	1	Basic Windows Controls: Text box Control, List Box,	T1: 241-279,W2
2.	1	checked List Box, Scroll Bar and Track Bar Controls	T1: 279-287
3.	1	More windows Control,	T1: 289-304 R2:450-460
4.	1	The Common Dialog Controls,Rich Text Box Control	T1: 305-326
5.	1	Handing Strings & Characters, String Class, String Builder class,	T1: 529-552, W2
6.	1	Tree view and list view control	T1: 552-568
7.	1	Examining the Advanced Controls, The Tree view control	T1:741-766
8.	1	The tree view and List view Control	T1:769-783
9.	1		
		Total No. of Hours Planned for Unit-II	**6**
Unit – III			
1.	1	The Multiple Document Interface MDI Application	T1: 831-864
2.	1	Databases: Architecture and Basic Concepts, Server Explorer	T1: 869-883
3.	1	Building Database Applications with Ado.Net: The Architecture of Ado.Net, Creation a dataset, Data Binding	T1: 926-946
4.	1	Programming The Data Adapter object, The Command and Data Reader Objects	
5.	1	Programming The ADO.Net Object:, The Structure of a data set	T1: 951-956, W3
6.	1	Recapitulation and Discussion of important questions	
		Total No. of Hours Planned for Unit-III	**10**
Unit – IV			

1.	1	Goal of ASP.Net	T2: 1-9
2.	1	ASP .Net Web Server Controls: Overview, Label, Literal, Textbox and button Sever Controls	T2: 155-165 R4:902-914
3.	1	Validation Server Controls	T2: 190-204
4.	1	Themes	T2: 244-272
5.	1	Themes and Cascading style sheet	T2: 244-272
6.	1	Themes graphics and other Resources	T2: 244-272
7.	1	Skins	T2: 245-285
8.	1	Contents Page Holder	T2: 307-325
9.	1	Adding content page holder	T2: 325-355
10.	1	Recapitulation and Discussion of important questions	
		Total No. of Hours Planned for Unit-IV	**14**
		Unit – V	
1.	1	Data Binding in ASP.Net	T2: 365-389
2.	1	Data source Control	T2: 380-389. W5
3.	1	Data source view	T2: 389-427
4.	1	Configuring data source control caching	T2: 427-435
5.	1	Storing connection information	T2: 440-453 R3:940-963
6.	1	Using bound list control with data source control	T2: 453-46, W5

7.	1	Other databound control.	T2: 461-481
8.	1	DBC: composit and hierarchical	T2: 481-506
9.	1	Data management with ADO.net	R3:978-986
10.	1	Data set in ASP.net	W5, J3
11.	1	Recapitulation and Discussion of important questions	
12.	1	Discussion of previous ESE question papers	
13.	1	Discussion of previous ESE question papers	
14.	1	Discussion of previous ESE question papers	
		Total No. of Hours Planned for Unit-V	**14**
		Total No. of Hours Planned for this Syllabus	**48**

SUPPORT MATERIALS

1. Evangelos Petroutsos, Mastering Visual Basic.Net, BPB Publications, New Delhi.
2. Bill Evjen, Scott Hanselman , Devin Rader, Farhan Muhammad and S.Srinivasa Sivakumar (2006), Professional ASP.net 2.0, Special Edition.
3. Dave Mercer, ASP.Net Beginner's Gudie (2003), 2nd Edition McGraw Hill, New Delhi.
4. Duncan Mackenzie Kent Sharkey (2006), Sams Teach yourself Visual Basic.JNet, 1st Edtion, McGraw Hill, NewDelhi.
5. Shirish Chavan. (2007), Visual Basic.Net, 1st Edition, Pearson Education, New Delhi.

WEB SITES

1. www.microsoft.com/NET/
2. www.en.wikipedia.org/wiki/.net
3. www.w3schools.com/ngws/default.asp
4. www.vbtutot.com

UNIT-I

SYLLABUS

Introduction: Getting Started with VB.NET: The Integrated Development Environment-IDE Components-Environment Options.Visual Basic: The Language Variables-Constants-Arrays – Variables as Objects-Flow Control Statements.Working with forms: The appearance of Forms-Loading and Showing Forms-Designing Menus.

Introduction to .NET

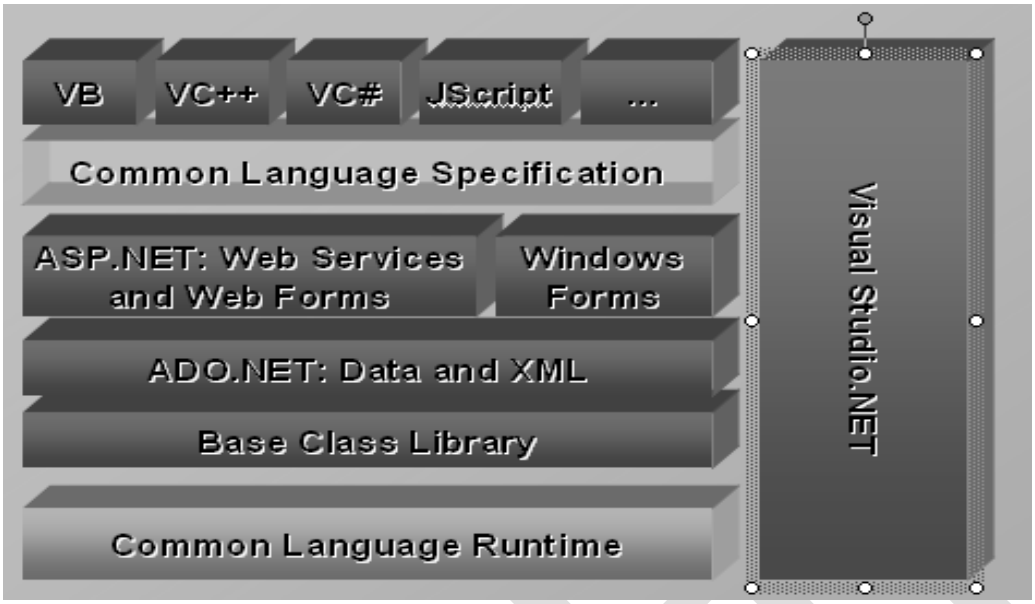
- .NET technology was introduced by Microsoft.
- It is a platform neutral framework.
- It is a layer between the operating system and the programming language. It supports many programming languages.
- .NET provides a common set of class libraries, which can be accessed from any .NET based programming language.

.NET supports the following languages:

- C#
- VB.NET
- C++
- J#

.NET Framework:

The components that make up the .NET platform are collectively called the .NET Framework. It is designed for cross-language compatibility. Cross language compatibility means, an application written in Visual Basic .NET may reference a DLL file written in C#.

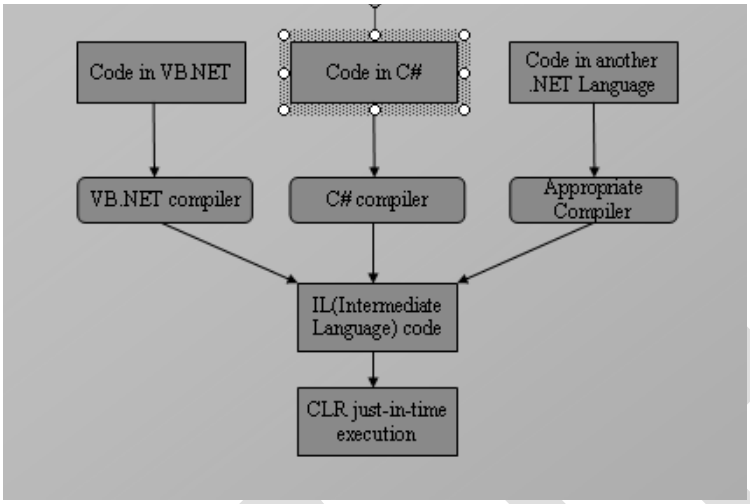


The .NET Framework consists of two main components:

- Common Language Runtime (CLR)
- Class Libraries

Common Language Runtime (CLR)

1. The CLR is described as the "execution engine" of .NET. It provides the environment within which the programs run.
2. CLR that manages the execution of programs and provides core services, such as code compilation, memory allocation, thread management, and garbage collection.
3. When the .NET program is compiled, the output of the compiler is not an executable file but a file that contains a special type of code called the Microsoft Intermediate Language (MSIL), which is a low-level set of instructions understood by the common language run time.
4. The JIT compiler converts MSIL into native code(executable code0.



Class Libraries

- Class library is designed to integrate with the common language runtime.
- The class library consists of lots of prewritten code that all the applications created in VB .NET and Visual Studio .NET will use. The code for all the elements like forms, controls and the rest in VB .NET applications actually comes from the class library.

Features of .NET Framework

1. Consistent Programming Model

2. Direct Support for Security

3. Simplified Development Efforts

4. Easy Application Deployment and Maintenance

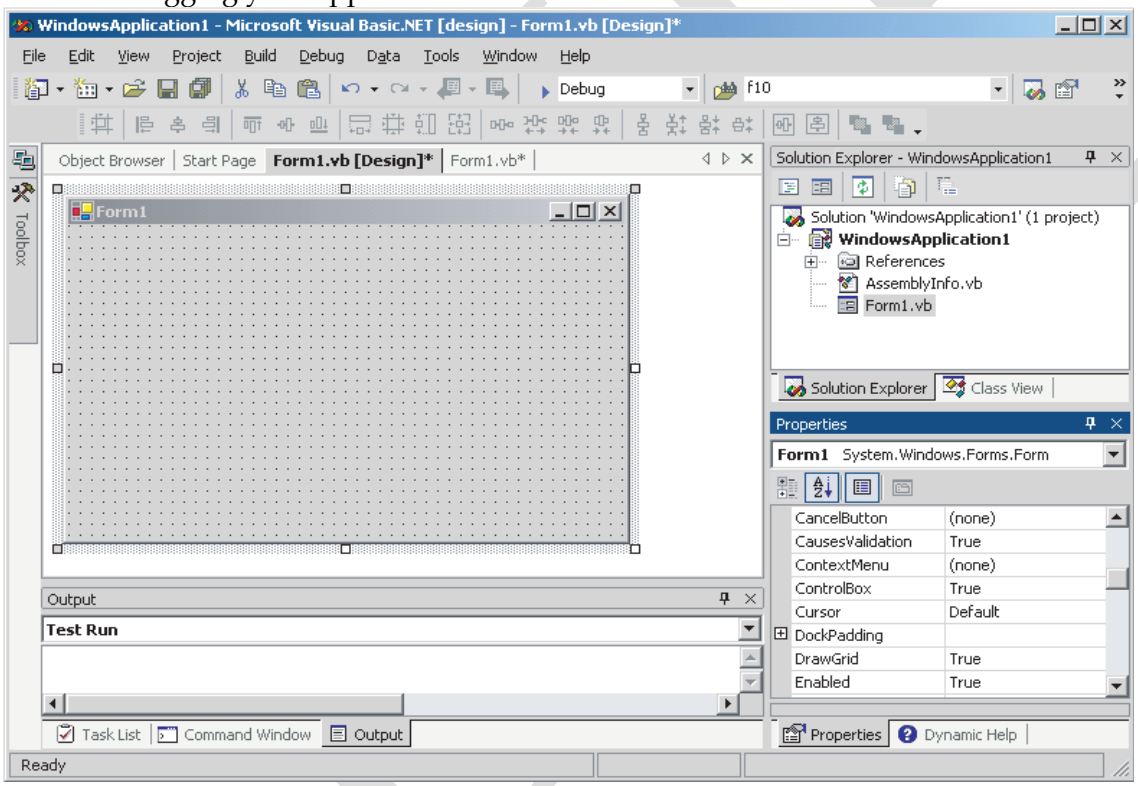
Getting Started with VB.NET

- Visual Studio .NET is an integrated environment for building, testing, and debugging a variety of Applications: Windows applications, Web applications, classes and custom controls, console applications.

- It provides numerous tools for automating the development process, visual tools to perform many common design and programming tasks.

The Integrated Development Environment

- To simplify the process of application development, Visual Studio .NET provides an environment that's common to all languages, which is known as *integrated development environment (IDE).*
- The purpose of the IDE is to enable the developer to do as much as possible with visual tools, before writing code. The IDE provides tools for designing, executing, and debugging your applications.



1. **The IDE Menu**

The IDE main menu provides the following commands, which lead to submenus.

- **File Menu**

The File menu contains commands for opening and saving projects, or project items.

Edit Menu

The Edit menu contains the usual editing commands. It also contains two more special commands.

I) The Advanced command
II) The IntelliSense command.

Advanced Submenu

Advanced submenu is visible only when the code editor opens.

View White Space Space characters are replaced by periods.

Word Wrap When a code line's length exceeds the length of the code window, it's automatically wrapped.

Comment Selection/Uncomment Selection Comments are lines you insert between your code's statements to document your application.

IntelliSense submenu:

IntelliSense is a feature of the editor that displays as much information as possible, whenever possible

The IntelliSense submenu includes the following options.

i) **List Members** When this option is on, the editor lists all the members (properties, methods, events, and argument list) in a drop-down list. This list will appear when you enter the name of an object or control followed by a period.

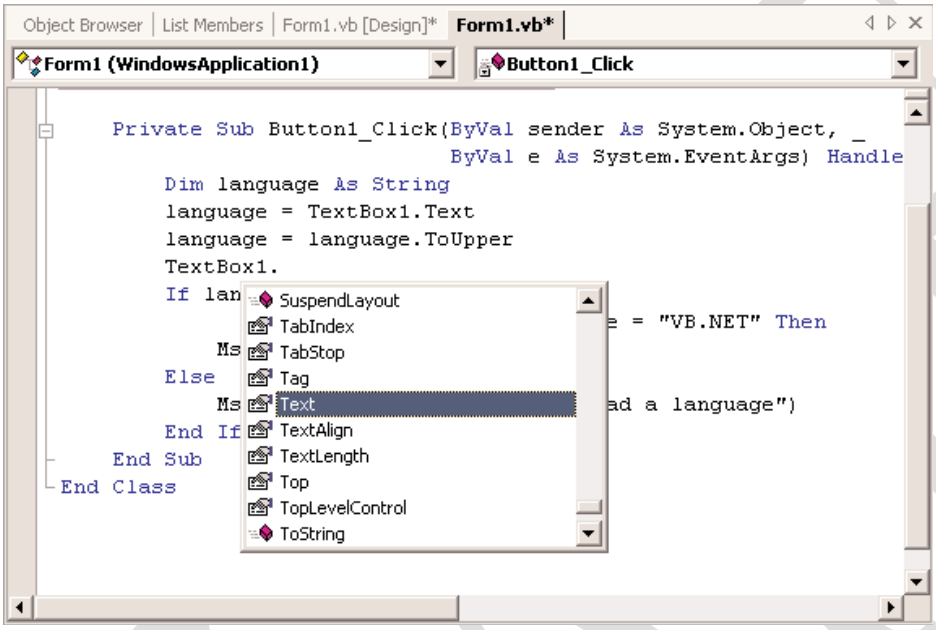
Example:

TextBox1.

a list with the members of the TextBox control will appear. Select the

Text property and then type the equal sign, followed by a string in quotes like the following:

TextBox1.Text = "Your User Name"



ii) Parameter Info While editing code, you can move the pointer over a variable, method, or property and see its declaration in a yellow tooltip.

iii)Quick Info This is another IntelliSense feature that displays information about commands and functions.

iv) Complete Word: The Complete Word feature enables you to complete the current word by pressing Ctrl+spacebar.

View Menu

This menu contains commands to display any toolbar or window of the IDE.

Project Menu

This menu contains commands for adding items to the current project.

Build Menu

The Build menu contains commands for building (compiling) your project. The two basic commands in this menu are the Build and Rebuild All commands.

The Build command compiles (builds the executable) of the entire solution, but it doesn't compile any components of the project that haven't changed since the last build.

The Rebuild all command does the same, but it clears any existing files and builds the solution from scratch.

Debug Menu

This menu contains commands to start or end an application, as well as the basic debugging tools.

Data Menu

This menu contains commands to access data for the project.

Format Menu

The Format menu contains commands for aligning the controls on the form.

Tools Menu

This menu contains a list of tools, and most of them apply to C++. The Macros command of the Tools menu leads to a submenu with commands for creating macros.

Window Menu

It contains command to open and hide windows.

Help Menu

This menu contains the various help options. The Dynamic Help command opens the Dynamic Help window, which is populated with topics that apply to the current operation. The Index command opens the Index window, where you can enter a topic and get help on the specific topic.

The Toolbox Window

The Toolbox window is usually retracted, and you must move the pointer over it to view the Toolbox. This window contains these tabs:

- Crystal Reports
- Data
- XML Schema
- Dialog Editor
- Web Forms
- Components
- Windows Forms
- HTML
- Clipboard Ring
- General

The Windows Forms tab contains the icons of the controls which can be placed on a Windows form.

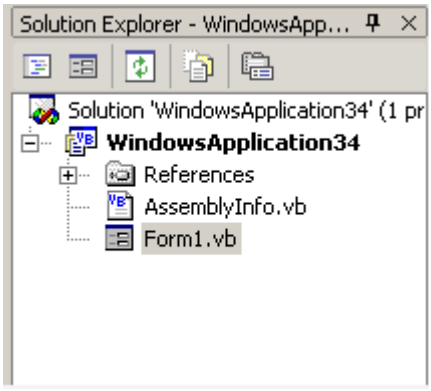
The Web Forms and HTML tabs contain the icons of the controls you can place on a Web form.

The Data tab contains the icons which are used to build data-driven applications.

The XML Schema tab contains the tools to work with schema XML files.

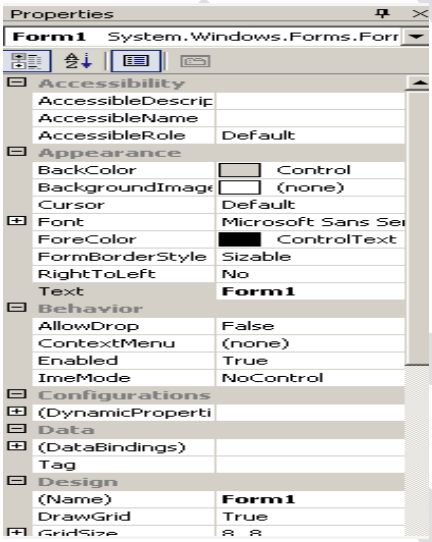
The Solution Explorer

This window contains a list of the items in the current solution. A solution may contain multiple projects, and each project may contain multiple items. The Solution Explorer displays a hierarchical list of all the components, organized by project.



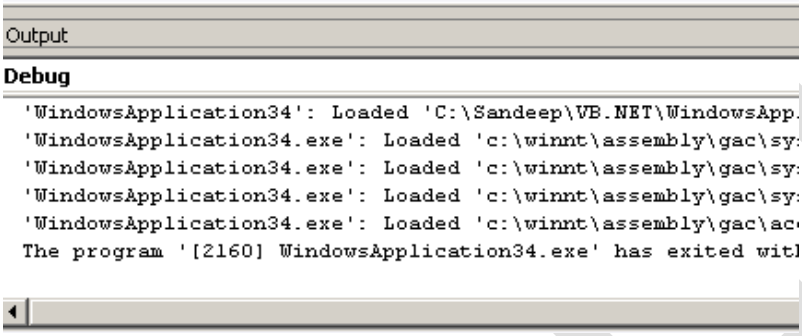
5. Properties Window

The properties window allows to set properties for various objects at design time. For example, if you want to change the font, font size, backcolor, name, text that appears on a button, textbox etc, you can do that in this window.



The Output Window

The Output window is where the compiler send its output. Every time you start an application, a series of messages is displayed on the Output window. If the Output window is not visible, select the View ➢ Other Windows ➢ Output command from the menu.



The Command Window

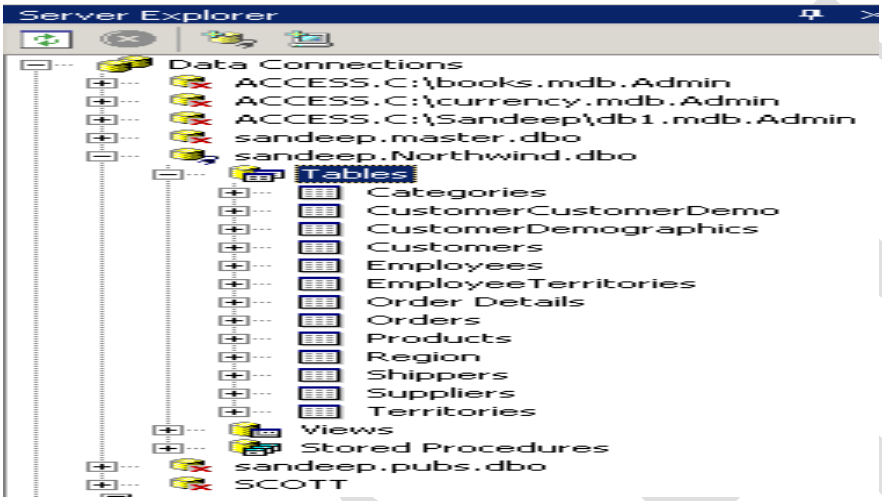
The execution of a program can be interrupted by inserting a breakpoint. When the breakpoint is reached, the program's execution is suspended and a statement in the Command window can be executed.

The Task List Window

This window is usually populated by the compiler with error messages, if the code can't be successfully compiled. You can double-click an error message in this window, and the IDE will take you to the line with the statement in error.

Server Explorer Window

The Server Explorer window is helped to work with databases in an easy graphical environment. For example, if we drag and drop a database table onto a form, VB .NET automatically creates connection and command objects that are needed to access that table. The image below displays Server Explorer window.



==========

Environment Options

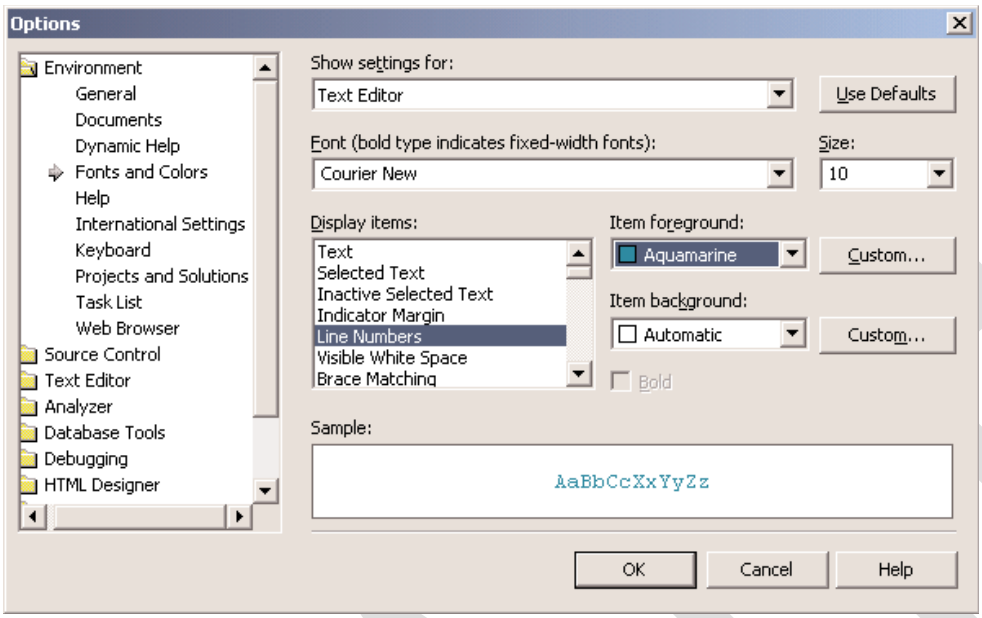
The visual studio IDE is highly customizable, Environment options deals with the customizable options exists in the visual studio IDE environment.

Font and Color Environment

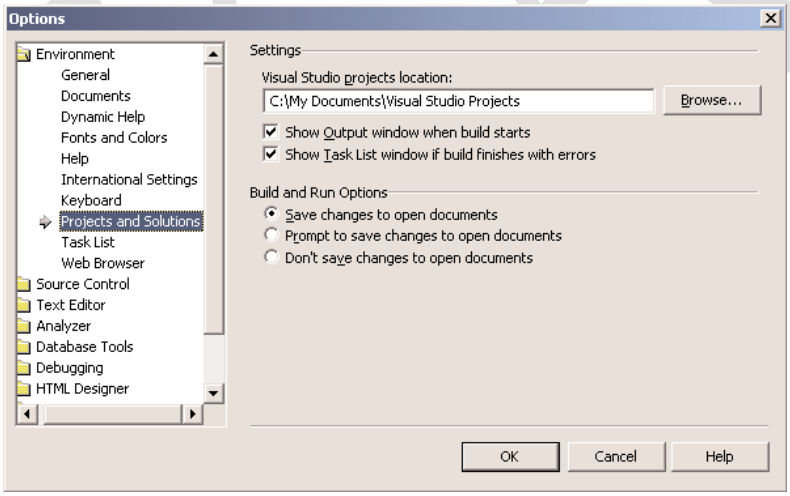
Open the Tools menu and select Options (the last item in the menu). The Options dialog box will appear where you can set all the options regarding the environment.

Projects and Solutions Options

The top box is the default location for new projects. The three radio buttons in the lower half of the dialog box determine when the changes to the project are saved. By default, changes are saved when you run a project. If you activate the last option, then you must save your project from time to time with the File ➢ Save All command



Font and Color Window



Project and Solution Window

Variables:

- Variables are used to store data.
- Each variable has a name and a value.
- Variables are declared with the Dim keyword. Dim stands for Dimension.

Declaring Variables

- A variable should be declared before it is used in program.
- It should declared with the datatype. It specifies the type of data it can hold.

Declare integer variables:

Dim width As Integer

Multiple variables declaration of the same type can done in a single line

Dim width, depth, height as Integer

The following statement will create **two integer and two Double variables**:

Dim width, height As Integer, area, volume As Double

Variable-Naming Conventions

- Start with a letter.
- The only special character that can appear in the variable's name is the underscore character.
- Limits of the characters are 255.
- Must be unique within the scope.
- Variable names in VB.NET are case-insensitive.

Variable initialization

The variables can be initialized in the same line that declares them.

Here declares an integer variable and initializes in to 50:

Dim instance as Integer = 50

Declare and initialize multiple variables, of the same or different type, on the same line:

Dim quantity As Integer = 1, discount As Single = 0.25

Types of Variables

Visual Basic recognizes the following five categories of variables:

_ Numeric

_ String

_ Boolean

_ Date

_ Object

Numeric Datatype:

All programming languages provide a variety of numeric data types, including the following:

_ Integers

_ Decimals

_ Single, or floating-point numbers with limited precision

_ Double, or floating-point numbers with extreme precision

Visual Basic Numeric Data Types

Data Type	Memory Representation	Stores
Short (Int16)	2 bytes	Integer values in the range –32,768 to 32,767.
Integer (Int32)	4 bytes	Integer values in the range –2,147,483,648 to 2,147,483,647.
Long (Int64)	8 bytes	Very large integer values
Single	4 bytes	Single-precision floating-point numbers. It can represent negative numbers in the range 3.402823E38 to –1.401298E–45 and positive numbers in the range 1.401298E–45 to 3.402823E38.
Double	8 bytes	Double-precision floating-point numbers. It can represent negative numbers in the range –1.79769313486232E308 to –4.94065645841247E–324 and positive numbers in the range 4.94065645841247E–324 to 1.79769313486232E308.
Decimal	16 bytes	Integer and floating-point numbers scaled by a factor in the range from 0 to 28

Example;

Dim a As Single, b As Double

a = 1 / 3

Console.WriteLine(a)

Console.WriteLine(a)

Output:

.3333333

0.333333333333333

Not a Number (NaN)

The value NaN indicates that the result of an operation can't be defined: it's not a regular number, not zero, and not Infinity.

To divide zero by zero, set up two variables as follows:

Dim var1, var2 As Double

Dim result As Double

var1 = 0

var2 = 0

result = var1 / var2

MsgBox(result)

The result of the above statements will be a NaN.

The Decimal Data Type

Decimal, are stored internally as integers in 16 bytes.

Dim a As Decimal = 328.558

Dim b As Decimal = 12.4051

Dim c As Decimal

c = a * b

Console.WriteLine(c)

The result of the above statements will be truncated to 3 decimal digits: 4,075.795.

The Byte Data Type

The Byte type holds an integer in the range 0 to 255.If the variables *A*and *B* are initialized as follows:

Dim A As Byte, B As Byte

A = 233

B = 50

The following statement will produce an overflow exception:

Console.WriteLine(A + B)

Boolean Variables

The Boolean data type stores True/False values.

Boolean variables are declared as:

Dim failure As Boolean

and they are initialized to False.

String Variables

The String data type stores only text. It is declared with the String type:

Dim someText As String

Example:

Dim aString As String

aString = "Now is the time for all good men to come to the aid of their country"

Character Variables

Character variables store a single Unicode character in two bytes. Characters are unsigned short integers (UInt16).

CChar() function --→ to convert integers to characters

CInt() function --→ to convert characters to their equivalent integer values.

To declare a character variable, use the Char keyword:

Dim char1, char2 As Char

Dim char1 As Char = "a", char2 As Char = "ABC"

Console.WriteLine(char1)

Console.WriteLine(char2)

Char data type exposes interesting methods like IsLetter, IsDigit, IsPunctuation, and so on.

Processing Individual Characters

Dim password As String, ch As Char

Dim i As Integer

Dim valid As Boolean = False

While Not valid

password = InputBox(“Please enter your password”)

For i = 0 To password.Length - 1

ch = password.Chars(i)

If Not System.Char.IsLetterOrDigit(ch) Then

valid = True

Exit For

End If

Next

If valid Then

MsgBox(“Your new password will be activated immediately!”)

Else

MsgBox(“Your password must contain at least one special symbol!”)

End If

End While

Date Variables

A variable declared as Date can store both date and time values with a statement like the following:

Dim expiration As Date

The following are all valid assignments:

expiration = #01/01/2004#

expiration = #8/27/2001 6:29:11 PM#

expiration = "July 2, 2002"

expiration = Now()

Data Type Identifiers

They are special symbols, which can be appended to the variable name to denote the variable's type.

To create a string variable, you can use the statement:

Dim myText$

Data Type Definition Characters

Symbol	Data Type	Example
$	**String**	*A$, messageText$*
%	**Integer**	*counter%, var%*
&	**Long**	*population&, colorValue&*
!	**Single**	*distance!*
#	**Double**	*ExactDistance#*
@	**Decimal**	*Balance@*

Object Variables

- *Variants*—variables without a fixed data type
- Variants were the most flexible data type because they could accommodate all other types.

Object Variables

Variants—variables without a fixed data type

Variants are the opposite of strictly typed variables: they can store all types of values, from a single character to an object.

Variants were the most flexible data type because they could accommodate all other types. A variable declared as Object (or a variable that hasn't been declared at all) is handled by Visual Basic according to the variable's current contents. If you assign an integer value to an Object variable,

Visual Basic treats it as an integer. If you assign a string to an Object variable, Visual Basic treats it as a string. Variants can also hold different data types in the course of the same program. Visual Basic performs the necessary conversions for you.

To declare a variant, you can turn off the Strict option and use the Dim statement without specifying a type, as follows:

Dim myVar

If you don't want to turn off the Strict option (which isn't recommended anyway), you can declare the variable with the Object data type:

Dim myVar As Object

Converting Variable Types

CBool ------- Boolean

CByte ------- Byte

CChar ----------Unicode character

CDate ---------Date

CDbl ----------Double

CDec ---------Decimal

CInt ---------Integer (4-byte integer, Int32)

CLng -------Long (8-byte integer, Int64)

CObj -----------Object

CShort -------Short (2-byte integer, Int16)

CSng ----------Single

CStr ------------String

User-Defined Data Types

A structure for storing multiple values (of the same or different type) is called a *record*. For example, each check in a checkbook-balancing application is stored in a separate record.

Record Structure

Check Number	Check Date	Check Amount	Check Paid To

Array Of Records

275	04/12/01	104.25	Gas Co.
276	04/12/01	48.76	Books
277	04/14/01	200.00	VISA
278	04/21/01	430.00	Rent

To define a record in VB.NET, use the Structure statement, which has the following syntax:

Structure structureName

Dim variable1 As varType

Dim variable2 As varType

...

Dim variablen As varType

End Structure

varType can be any of the data types supported by the framework. The Dim statement can be replaced by the Private or Public access modifiers. For structures, Dim is equivalent to Public.

The declaration for the record structure

Structure CheckRecord

Dim CheckNumber As Integer

Dim CheckDate As Date

Dim CheckAmount As Single

Dim CheckPaidTo As String

End Structure

A variable's Scope

The *scope* (or *visibility*) of a variable is the section of the application that can see and manipulate the variable. If a variable is declared within a procedure, only the code in the specific procedure has access to that variable. This variable doesn't exist for the

rest of the application. When the variable's scope is limited to a procedure, it's called *local*.

A variable is said to have *procedure-level* scope. It's visible within the procedure and invisible outside the procedure.

Variables declared outside any procedure in a module are visible from within all procedures in the same module, but they're invisible outside the module.

The Lifetime of a Variable

- The lifetime of a variable is the period for which they retain their value.
- Variables declared as Public exist for the lifetime of the application.
- Local variables declared within procedures with the Dim or Private statement.

Constants

Variables don't change value during the execution of a program.

Constants don't change value

This is a safety feature. Once a constant has been declared, you can't change its value in subsequent statements, so you can be sure that the value specified in the constant's declaration will take effect in the entire program.

Constants don't change value

When the program is running, the values of constants don't have to be looked up. The compiler substitutes constant names with their values, and the program executes faster.

Example:

```
Public Const pi As Double = 3.14159265358979

Public Const pi2 As Double = 2 * pi
```

Arrays

Arrays can hold sets of data of the same type. An array has a name, as does a variable, and the values stored in it can be accessed by an index.

Declaring Arrays

Arrays must be declared with the Dim (or Public, or Private) statement followed by the name of the array and the index of the last element in the array in parentheses.

Example,

Dim Salaries(15) As Integer

Array Limits

The first element of an array has index 0. The number that appears in parentheses in the Dim statement is one less than the array's total capacity and is the array's upper limit (or upper bound).

The index of the last element of an array (its upper bound) is given by the function UBound(), which accepts as argument the array's name.

For the array

Dim myArray(19) As Integer

its upper bound is 19, and its capacity is 20 elements.

Initializing Arrays

Initialization and declaration of variables can be done by the following constructor:

Dim arrayname() As type = {entry0, entry1, … entryN}

Here's an example that initializes an array of strings:

Dim names() As String = {"Joe Doe", "Peter Smack"}

This statement is equivalent to the following statements, which declare an array with two elements and then set their values:

Dim names(1) As String

names(0) = "Joe Doe"

names(1) = "Peter Smack"

Multidimensional Arrays

A two-dimensional array has two indices. The first identifies the row, and the second identifies the column. To access the name and temperature of the third city in the two-dimensional array, use the following indices:

Temperatures(2, 0) ' the third city's name

Temperatures(2, 1) ' the third city's average temperature

The array could be declared as follows:

```
Dim Board(9, 9) As Integer
```

Dynamic Arrays

The size of a dynamic array can vary during the course of the program. To create a dynamic array, declare it as usual with the Dim statement but don't specify its dimensions:

Dim DynArray() As Integer

ReDim is executable—it forces the application to carry out an action at runtime. Dim statements aren't executable, and they can appear outside procedures.

A dynamic array also can be redimensioned to multiple dimensions. Declare it with the Dim statement outside any procedure as follows:

Dim Matrix() As Double

and then use the ReDim statement in a procedure to declare a three-dimensional array:

ReDim Matrix(9, 9, 9)

The Preserve Keyword

Preserve keyword, which forces it to resize the array without discarding the existing data.

ReDim Preserve DynamicArray(UBound(DynArray) + 1)

Example:

```
Public Class Form1
    Private Sub Button1_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles Button1.Click
        Dim i As Integer
        Dim scores() As Integer
        ReDim scores(1)
        scores(0) = 100
        scores(1) = 200
        For i = 0 To scores.Length – 1
            MsgBox(scores(i))
        Next
        ReDim Preserve scores(2)
        scores(2) = 300
        For i = 0 To scores.Length – 1
            MsgBox(scores(i))
        Next
    End Sub
End Class
```

Arrays of Arrays

If an array is declared as Object, you can assign other types to its elements, including arrays. Suppose you have declared and populated two arrays, one with integers and another with strings.

You can then declare an Object array with two elements and populate it with the two arrays

```
Dim IntArray(9) As Integer

Dim StrArray(99) As String

Dim BigArray(1) As Object

Dim i As Integer

‘ populate array IntArray

For i = 0 To 9

IntArray(i) = i

Next

‘ populate array StrArray

For i = 0 To 99

StrArray(i) = “ITEM “ & i.ToString(“0000”)

Next

BigArray(0) = IntArray

BigArray(1) = StrArray

Console.WriteLine(BigArray(0)(7))

Console.WriteLine(BigArray(1)(16))
```

The last two statements will print the following values on the Output window:

7

ITEM 0016

Variables as Objects

An *object* is a collection of data and code. An integer variable, *intVar*, is an object because it has a value and some properties and methods.

Properties and methods are implemented as functions. The method intVar.ToString for instance, returns the numeric value held in the variable as a string, so that you can use it in string operations. In other words, an Integer variable is an object that knows about itself.

It knows that it holds a whole number; it knows how to convert itself to a string; it knows the minimum and maximum values it can store (properties MinValue and MaxValue); and so on.

Formatting Numbers

The ToString method, exposed by all data types except the String data type, converts a value to the equivalent string and formats it at the same time

ToString(formatString)

The *formatString* argument is a format specifier (a string that specifies the exact format to be applied to the variable) this argument can be a specific character that corresponds to a predetermined format (*standard numeric format string*, as it's called) or a string of characters that have special meaning in formatting numeric values (a *picture numeric format string*). Use standard format strings for the most common operations and picture strings to specify unusual formatting requirements. To format the value 9959.95 as a dollar amount, you can use the following standard currency format string:

Dim int As Single = 9959.95

Dim strInt As String

strInt = int.ToString("C")

or the following picture numeric format string:

strInt = int.ToString("$###,###.00")

Both statements will format the value as "$9,959.95

Formatting Dates

If the variable *birthDate* contains the value #1/1/2000#, the following expressions return the values shown below them, in bold:

Console.WriteLine(birthDate.ToString("d"))

1/1/2000

Console.WriteLine(birthDate.ToString("D"))

Saturday, January 01, 2000

Console.WriteLine(birthDate.ToString("f"))

Saturday, January 01, 2000 12:00 AM

Console.WriteLine(birthDate.ToString("s"))

2000-01-01T00:00:00

Console.WriteLine(birthDate.ToString("U"))

Saturday, January 01, 2000 12:00:00 AM

Flow Control Statements

The one that differentiates the computers from calculators is, as a programmer one can control the flow of the program according to the external programs

Test Structures

An application needs a built-in capability to test conditions and take a different course of action depending on the outcome of the test. Visual Basic provides three such decision structures:

- If…Then
- If…Then…Else
- Select Case

If…Then

The If…Then statement tests the condition specified; if it's True, the program executes the statement(s) that follow. The If structure can have a single-line or a multiple-line syntax. To execute one statement conditionally, use the single-line syntax as follows:

If condition Then statement

Visual Basic evaluates the *condition*, and if it's true, executes the statement that follows. If the condition is False, the application continues with the statement following the If statement.

You can also execute multiple statements by separating them with colons:

If condition Then statement: statement: statement

Here's an example of a single-line If statement:

If Month(expDate) > 12 Then expYear = expYear + 1: expMonth = 1

You can break this statement into multiple lines by using End If, as shown here:

If expDate.Month > 12 Then

expYear = expYear + 1

expMonth = 1

End If

If condition Then

statementblock1

Else

statementblock2

End If

If…Then…Else

A variation of the If…Then statement is the If…Then…Else statement, which executes one block of statements if the condition is True and another block of statements if the condition is False.

The syntax of the If…Then…Else statement is as follows:

If condition1 Then

statementblock1

ElseIf condition2 Then

statementblock2

ElseIf condition3 Then

statementblock3

Else

statementblock4

End If

For example

score = InputBox("Enter score")

If score < 50 Then

```
Result = "Failed"
ElseIf score < 75 Then
Result = "Pass"
ElseIf score < 90 Then
Result = "Very Good"
Else
Result = "Excellent"
End If
MsgBox Result
```

Select Case

An alternative to the efficient, but difficult-to-read, code of the multiple-ElseIf structure is the select Case structure, which compares one expression to different values. The advantage of the Select Case statement over multiple If…Then…Else/ElseIf statements is that it makes the code easier to read and maintain.

The Select Case structure tests a single expression, which is evaluated once at the top of the structure. The result of the test is then compared with several values, and if it matches one of them, the corresponding block of statements is executed.

Here's the syntax of the Select Case

statement:

Select Case expression

Case value1

statementblock1

Case value2

statementblock2

.

.

.

Case Else

statementblockN

End Select

For Example

Dim message As String

Select Case Now.DayOfWeek

Case DayOfWeek.Monday

message = "Have a nice week"

Case DayOfWeek.Friday

message = "Have a nice weekend"

Case Else

message = "Welcome back!"

End Select

MsgBox(message)

Loop Structures

Loop structures allow you to execute one or more lines of code repetitively. Many tasks consist of trivial operations that must be repeated over and over again, and looping structures are an important part of any programming language.

Visual Basic supports the following loop structures:

- **For…Next**
- **Do…Loop**
- **While…End While**

For…Next

The For…Next loop is one of the oldest loop structures in programming languages. Unlike the other two loops, the For…Next loop requires that you know how many times the statements in the loop will be executed. The For…Next loop uses a variable (it's called the loop's *counter*) that increases or decreases in value during each repetition of the loop.

The For…Next loop has the following syntax:

For counter = start To end [Step increment]

statements

Next [counter]

The keywords in the square brackets are optional. The arguments *counter*, *start*, *end*, and *increment* are all numeric. The loop is executed as many times as required for the *counter* to reach (or exceed) the *end* value.

In executing a For…Next loop, Visual Basic completes the following steps:

1. Sets *counter* equal to *start*

2. Tests to see if *counter* is greater than *end*. If so, it exits the loop. If *increment* is negative, Visual

Basic tests to see if *counter* is less than *end*. If it is, it exits the loop.

3. Executes the statements in the block

4. Increments *counter* by the amount specified with the *increment* argument. If the *increment* argument isn't specified, *counter* is incremented by 1

5. Repeat the Statements

For Example

```
Dim i As Integer, total As Double
For i = 0 To data.GetUpperBound(0)
total = total + data(i)
Next i
Console.WriteLine (total / data.Length)
```

The single most important thing to keep in mind when working with For…Next loops is that the loop's *counter* is set at the beginning of the loop. Changing the value of the *end* variable in the loop's body won't have any effect.

For example, the following loop will be executed 10 times, not 100 times:

```
endValue = 10

For i = 0 To endValue

endValue = 100

{ more statements }

 Next i
```

You can, however, adjust the value of the *counter* from within the loop. The following is an example of an endless (or infinite) loop:

```
For i = 0 To 10

Console.WriteLine(i)

i = i - 1

Next i
```

This loop never ends because the loop's *counter,* in effect, is never increased.

Do…Loop

The Do…Loop executes a block of statements for as long as a condition is True. Visual Basic evaluates an expression, and if it's True, the statements are executed. When the end of block is reached, the expression is evaluated again and, if it's True, the statements are repeated. If the expression is False, the program continues and the statement following the loop is executed.

There are two variations of the Do…Loop statement; both use the same basic model.

A loop can be executed either while the condition is True or until the condition becomes True. These two variations use the keywords While and Until to specify how long the statements are executed.

To execute a block of statements while a condition is True, use the following syntax:

Do While condition

statement-block

Loop

Here's a typical example of using a Do…Loop. Suppose the string *MyText* holds a piece of text, and you want to count the words in the text.

To locate an instance of a character in a string, use the InStr() function, which accepts three arguments:

- The starting location of the search
- The text to be searched
- The character being searched

The following loop repeats for as long as there are spaces in the text. Each time the InStr() function finds another space in the text, it returns the location (a positive number) of the space. When there are no more spaces in the text, the InStr() function returns zero, which signals the end of the loop, as shown:

Dim MyText As String = "The quick brown fox jumped over the lazy dog"

Dim position, words As Integer

position = 1

Do While position > 0

position = InStr(position + 1, MyText, " ")

words = words + 1

Loop

Console.WriteLine "There are " & words & " words in the text"

The Do…Loop is executed while the InStr() function returns a positive number, which happens for as long as there are more words in the text. The variable *position* holds the location of each successive space character in the text.

The search for the next space starts at the location of the current space plus 1 (so that the program won't keep finding the same space). For each space found the program increments the value of the *words* variable, which holds the total number of words when the loop ends.

Nested Control Structures

Control structures in Visual Basic can be nested in as many levels as you want, It's common practice to indent the bodies of nested decision and loop structures to make the program easier to read.

The following pseudocode demonstrates how to nest several flow-control statements:

```
For a = 1 To 100
{ statements }
If a = 99 Then
{ statements }
End If
While b < a
{ statements }
If total <= 0 Then
{ statements }
End If
End While
```

```
For c = 1 to a

{ statements }

Next

Next
```

For Example

```
Dim Array2D(6, 4) As Integer

Dim iRow, iCol As Integer

For iRow = 0 To Array2D.GetUpperBound(0)

For iCol = 0 To Array2D.GetUpperBound(1)

Array2D(iRow, iCol) = iRow * 100 + iCol

Console.Write(iRow & ", " & iCol & " = " & Array2D(iRow, iCol) & " ")

Next iCol

Console.WriteLine()

Next iRow
```

The Exit Statement

The Exit statement allows you to exit prematurely from a block of statements in a control structure, from a loop, or even from a procedure. Suppose you have a For…Next loop that calculates the square root of a series of numbers. Because the square root of negative numbers can't be calculated (the Sqrt() function will generate a runtime error), you might want to halt the operation if the array contains an invalid value.

To exit the loop prematurely, use the Exit For statement as follows:

```
For i = 0 To UBound(nArray)
If nArray(i) < 0 Then Exit For
nArray(i) = Math.Sqrt(nArray(i))
Next
```

If a negative element is found in this loop, the program exits the loop and continues with the statement following the Next statement.

There are similar Exit statements for the Do loop (Exit Do) and the While loop (Exit While), as well as for functions and subroutines (Exit Function and Exit Sub). If the previous loop was part of a function, you might want to display an error and exit not only the loop, but the function itself:

```
For i = 0 To nArray.GetUpperBound()
If nArray(i) < 0 Then
MsgBox "Negative value found, terminating calculations"
Exit Function
End If
nArray(i) = Sqr(nArray(i))
Next
```

If this code is part of a subroutine procedure, you use the Exit Sub statement. The Exit statements for loops are Exit For, Exit While, and Exit Do.

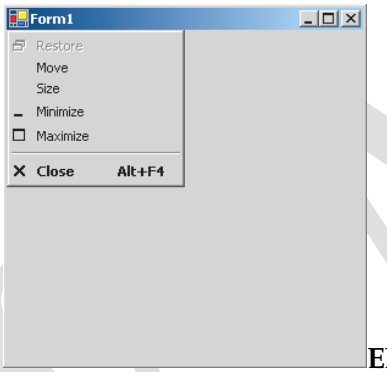
Working with Forms

In Visual Basic, the form is the container for all the controls that make up the user interface. When a Visual Basic application is executing, each window it displays on the desktop is a form.

The terms form and window describe the same entity. A window is what the user sees on the desktop when the application is running. A form is the same entity at design time.

Appearance of Forms

Applications are made up of one or more forms (usually more than one), and the forms are what users see. You should craft your forms carefully, make them functional, and keep them simple and intuitive. You already know how to place controls on the form, but there's more to designing forms than populating them with controls



Elements of the Form

Clicking the icon on the left end of the title bar opens the Control menu, which contains the commands. On the right end of the title bar are three buttons: Minimize Maximize and Close.

Clicking these buttons performs the associated function. When a form is maximized, the Maximize button is replaced by the Restore button. When clicked, this button resets the form to the size and position before it was maximized. The Restore button is then replaced by the Maximize button.

Command	Effect

Restore	Restores a maximized form to the size it was before it was maximized; available only if the form has been maximized
Move	Lets the user move the form around with the mouse
Size	Lets the user resize the form with the mouse
Minimize	Minimizes the form
Maximize	Maximizes the form
Close	Closes the current form

Properties of the Form Control

The floating toolbars used by many graphics applications, for example, are actually forms with a narrow title bar. The dialog boxes that display critical information or prompt you to select the file to be opened are also forms. You can duplicate the look of any window or dialog box through the following properties of the Form object.

AcceptButton, CancelButton

These two properties let you specify the default Accept and Cancel buttons. The Accept button is the one that's automatically activated when you press Enter, no matter which control has the focus at the time, and is usually the button with the OK caption.

Likewise, the Cancel button is the one that's automatically activated when you hit the Esc key and is usually the button with the Cancel caption.

To specify the Accept and Cancel buttons on a form, locate the AcceptButton and Cancel Button properties of the form and select the corresponding controls from a drop-down list, which contains the names of all the buttons on the form. You can also set them to the name of the corresponding button from within your code.

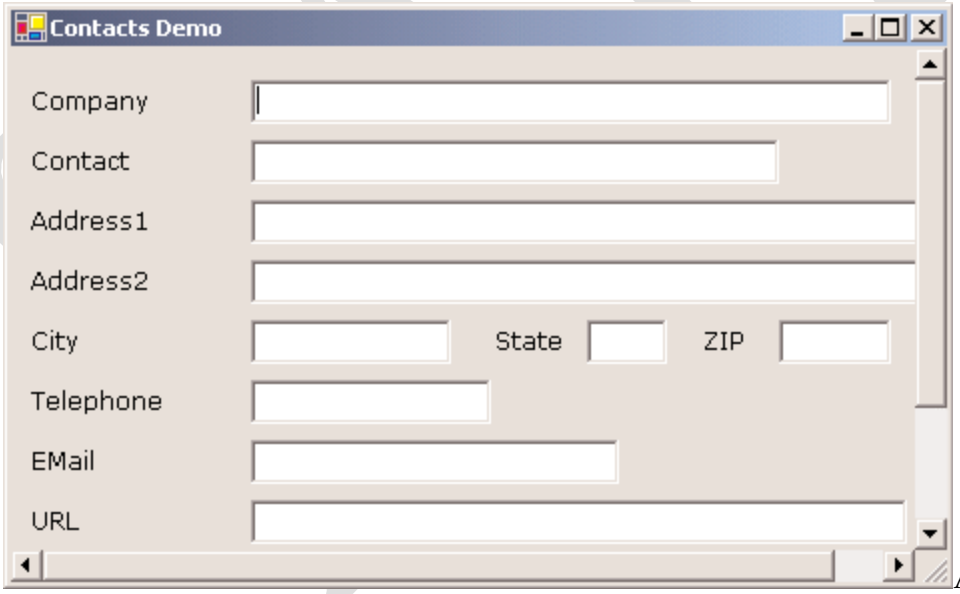
AutoScale

This property is a True/False value that determines whether the controls you place on the form are automatically scaled to the height of the current font. When you place a TextBox control on the form, for example, and the AutoScale property is True, the control will be tall enough to display a single line of text in the current font. The default value is True, which is why you can't make the controls smaller than a given size.

This is a property of the form, but it affects the controls on the form. If you change the Font property of the form after you have placed a few controls on it, the existing controls won't be affected. The controls are adjusted to the current font of the form the moment they're placed on it.

AutoScroll

This is one of the most needed of the Form object's new properties. The AutoScroll property is a True/False value that indicates whether scroll bars will be automatically attached to the form if it's resized to a point that not all its controls are visible.



AutoScroll

AutoScrollMargin

This is a margin, expressed in pixels, that's added around all the controls on the form. If the form is smaller than the rectangle that encloses all the controls adjusted by the margin, the appropriate scroll bar(s) will be displayed automatically.

If you expand the AutoScrollMargin property in the Properties window, you will see that it's an object (a Size object, to be specific). It exposes two members, the Width and Height properties, and you must set both values. The default value is (0,0).

To set this property from within your code, use statements like these:

Me.AutoScrollMargin.Width = 40

Me.AutoScrollMargin.Height = 40

ControlBox

This property is also True by default. Set it to False to hide the icon and disable the Control menu. Although the Control menu is rarely used, Windows applications don't disable it. When the ControlBox property is False, the three buttons on the title bar are also disabled. If you set the Text property to an empty string, the title bar disappears altogether.

KeyPreview

This property enables the form to capture all keystrokes before they're passed to the control that has the focus. Normally, when you press a key, the KeyPress event of the control with the focus is triggered (as well as the other keystroke-related events), and you can handle the keystroke from within the control's appropriate handler.

In most cases, we let the control handle the keystroke and don't write any form code for that. If you want to use "universal" keystrokes in your application, you must set the KeyPreview property to True.

Doing so enables the form to intercept all keystrokes, so that you can process them from within the form's keystroke events. The same keystrokes are then passed to the control with the focus, unless you "kill" the keystroke by setting its Handled property to True when you process it on the form's level. For more information on processing keystrokes at the Form level and using special keystrokes throughout your application

MinimizeBox, MaximizeBox

These two properties are True by default. Set them to False to hide the corresponding buttons on the title bar.

MinimumSize, MaximumSize

These two properties read or set the minimum and maximum size of a form. When users resize the form at runtime, the form won't become any smaller than the dimensions specified with the MinimumSize property and no larger than the dimensions specified by MaximumSize.

The MinimumSize property is a Size object, and you can set it with a statement like the following:

Me.MinimumSize = New Size(400, 300)

Top, Left

These two properties set or return the coordinates of the form's top-left corner in pixels. These properties are rarely used in the code, since the location of the window on the desktop is determined by the user at runtime.

TopMost

This property is a True/False value that lets you specify whether the form will remain on top of all other forms in your application. Its default property is False, and you should change it only in rare occasions.

Some dialog boxes, like the Find and Replace dialog box of any text processing application, are always visible, even when they don't have the focus. To make a form remain visible while it's open, set its TopMost property to True.

Width, Height

These two properties set or return the form's width and height in pixels. They are usually set from within the form's Resize event handler, to keep the size of the form at a minimum size. The forms width and height are usually controlled by the user at runtime.

Placing Controls on Forms

Designing a form means placing Windows controls on it, setting their properties, and then writing code to handle the events of interest. Visual Studio.NET is a rapid application development (RAD) environment.

It has come to mean that you can rapidly prototype an application and show something to the customer. And this is made possible through the visual tools that come with VS.NET, especially the new Form Designer.

To place controls on your form, you select them in the Toolbox and then draw, on the form, the rectangle in which the control will be enclosed. Or, you can double-click the control's icon to place an instance of the control on the form. All controls have a default size, and you can resize the control on the form with the mouse.

Next, you set the control's properties in the Properties window. Each control's dimensions can also be set in the Properties window, through the Width and Height properties.

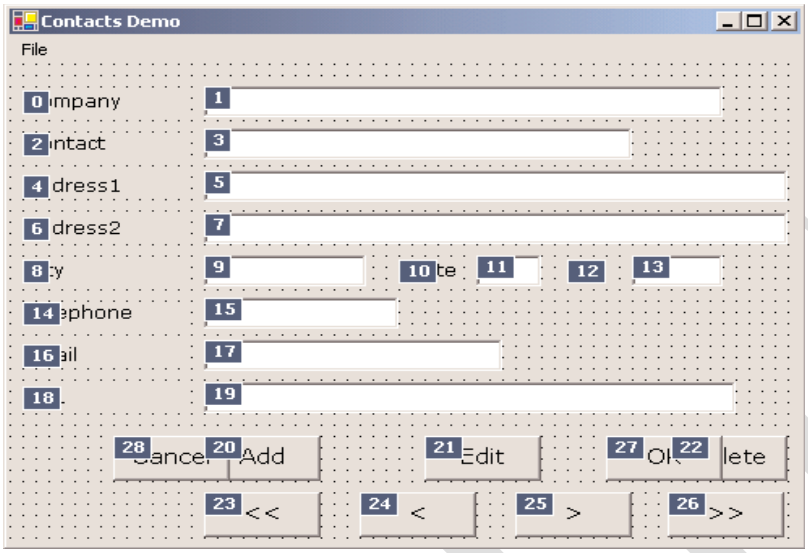
These two properties are expressed in pixels. You can also call the Width and Height properties from within your code to read the dimensions of a control. Likewise, the Top and Left properties return (or set) the coordinates of the top-left corner of the control.

Setting the TabOrder

Another important issue in form design is the tab order of the controls on the form. As you know, pressing the Tab key at runtime takes you to the next control on the form. The order of the controls isn't determined by the form; you specify the order when you design the application, with the help of the TabOrder property.

Each control has its own TabOrder setting, which is an integer value. When the Tab key is pressed, the focus is moved to the control whose tab order immediately follows the tab order of the current control. The TabOrder of the various controls on the form need not be consecutive.

To specify the tab order of the various controls, you can set their TabOrder property in the properties window, or you can select the Tab Order command from the View menu. The tab order of each control will be displayed on the corresponding control.



Setting the TabOrder of the controls

To set the tab order of the controls, click each control in the order in which you want them to receive the focus. Notice that you can't change the tab order of a few controls only. You must click all of them in the desired order, starting with the first control in the tab order.

The tab order need not be the same as the physical order of the controls on the form, but controls that are next to each other in the tab order should be placed next to each other on the form as well.

The Form's Events

The Form object triggers several events, the most important of them being Activate, Deactivate, Closing, Resize, and Paint.

The Activate and Deactivate Events

When more than one form is displayed, the user can switch from one to the other with the mouse or by pressing Alt+Tab. Each time a form is activated, the Activate event takes place. Likewise, when a form is activated, the previously active form receives the Deactivate event.

The Closing Event

This event is fired when the user closes the form by clicking its Close button. If the application must terminate because Windows is shutting down, the same event will be fired as well. Users don't always quit applications in an orderly manner, and a professional application should behave gracefully under all circumstances.

The same code you execute in the application's Exit command must also be executed from within the Closing event as well. For example, you may display a warning if the user has unsaved data, or you may have to update a database, and so on.

Place the code that performs these tasks in a subroutine and call it from within your menu's Exit command, as well as from within the Closing event's handler. You can cancel the closing of a form by setting the Cancel property to True.

For Example

```
Public Sub Form1_Closing(ByVal sender As Object, _
ByVal e As System.ComponentModel.CancelEventArgs) _
Handles Form1.Closing
Dim reply As MsgBoxResult
reply = MsgBox("Current document has been edited. Click OK to terminate " & _
"the application, or Cancel to return to your document.", _
MsgBoxStyle.OKCancel)
If reply = MsgBoxResult.Cancel Then
e.Cancel = True
```

End If

End Sub

The Resize Event

The Resize event is fired every time the user resizes the form with the mouse. With previous versions of VB, programmers had to insert quite a bit of code in the Resize event's handler to resize the controls and possibly rearrange them on the form. With the Anchor and Dock properties, much of this overhead can be passed to the form itself.

Many VB applications used the Resize event to impose a minimum size for the form. To make sure that the user can't make the form smaller than, say 300 by 200 pixels, you would insert these lines into the Resize event's handler:

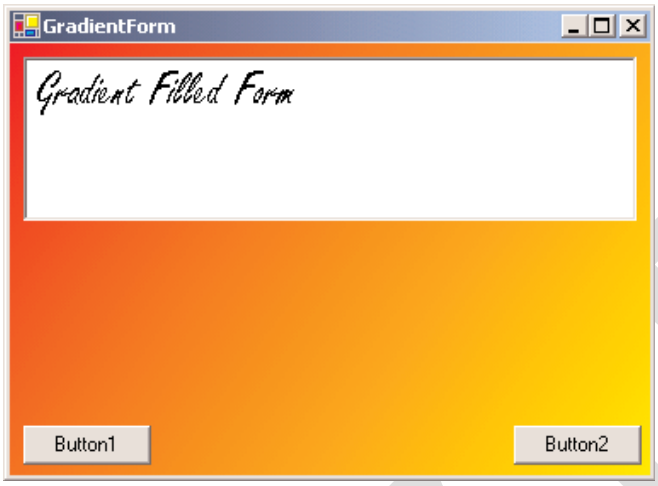
```
Private Form1_Resize(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles Form1.Resize
If Me.Width < minWidth Then Me.Width = minWidth
If Me.Height < minHeight Then Me.Height = minHeight
End Sub
```

The Paint Event

This event takes place every time the form must be refreshed. When you switch to another form that partially or totally overlaps the current form and then switch back to the first form, the Paint event will be fired to notify your application that it must redraw the form.

In this event's handler, we insert the code that draws on the form. The form will refresh its controls automatically, but any custom drawing on the form won't be refreshed automatically.

For Example



Loading and Showing Forms

To access a form from within another form, you must first create a variable that references the second form. Let's say your application has two forms, named Form1 and Form2, and that Form1 is the project's startup form.

To show Form2 when an action takes place on Form1, first declare a variable that references Form2:

Dim frm As New Form2

This declaration must appear in Form1 and must be placed outside any procedure. (If you place it in a procedure's code, then every time the procedure is executed, a new reference to Form2 will be created.

This means that the user can display the same form multiple times. All procedures in Form1 must see the same instance of the Form2, so that no matter how many procedures show Form2, or how many times they do it, they'll always bring up the same single instance of Form2.) Then, to invoke Form2 from within Form1, execute the following statement:

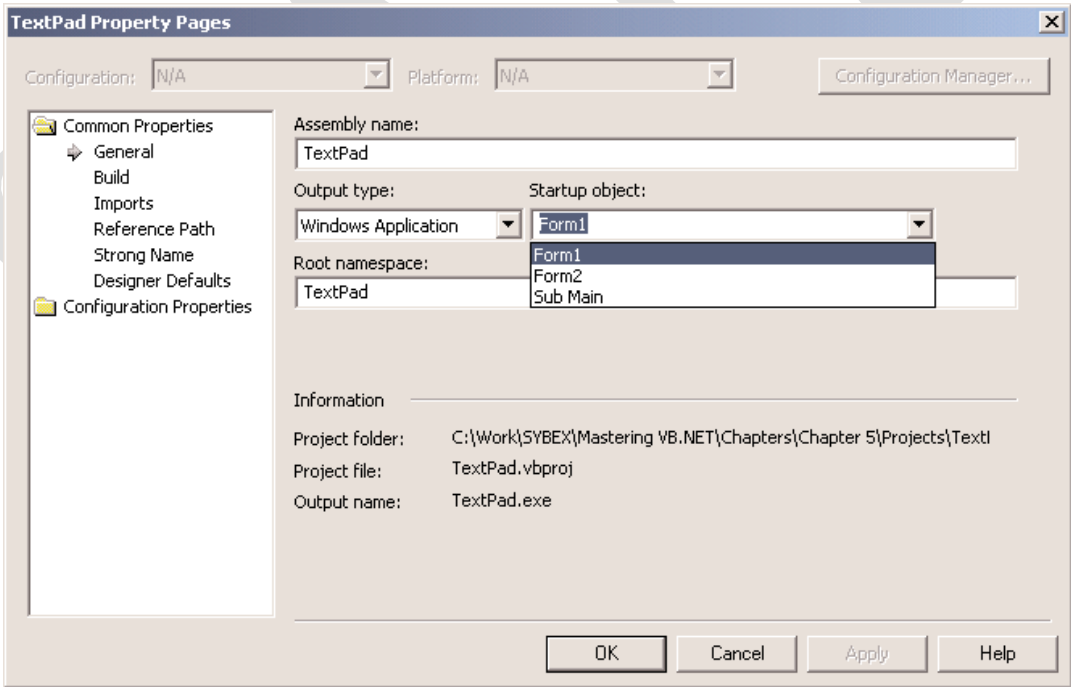
frm.Show

Startup Forms

A typical application has more than a single form. When an application starts, the main form is loaded. You can control which form is initially loaded by setting the startup object in the Project Properties window

To open this, right-click the project's name in the Solution Explorer and select Properties. In the project's Property Pages, select the Startup Object from the drop-down list. You can also see other parameters in the same window, which are discussed elsewhere in this book.

By default, Visual Basic suggests the name of the first form it created, which is Form1. If you change the name of the form, Visual Basic will continue using the same form as startup form, with its new name.



You can also start an application with a subroutine without loading a form. This subroutine must be called Main() and must be placed in a Module. Right-click the project's name in the Solution Explorer window and select the Add Item command. When the dialog box appears, select a Module. Name it StartUp (or anything you like; you can keep the default name Module1) and then insert the Main() subroutine in the module.

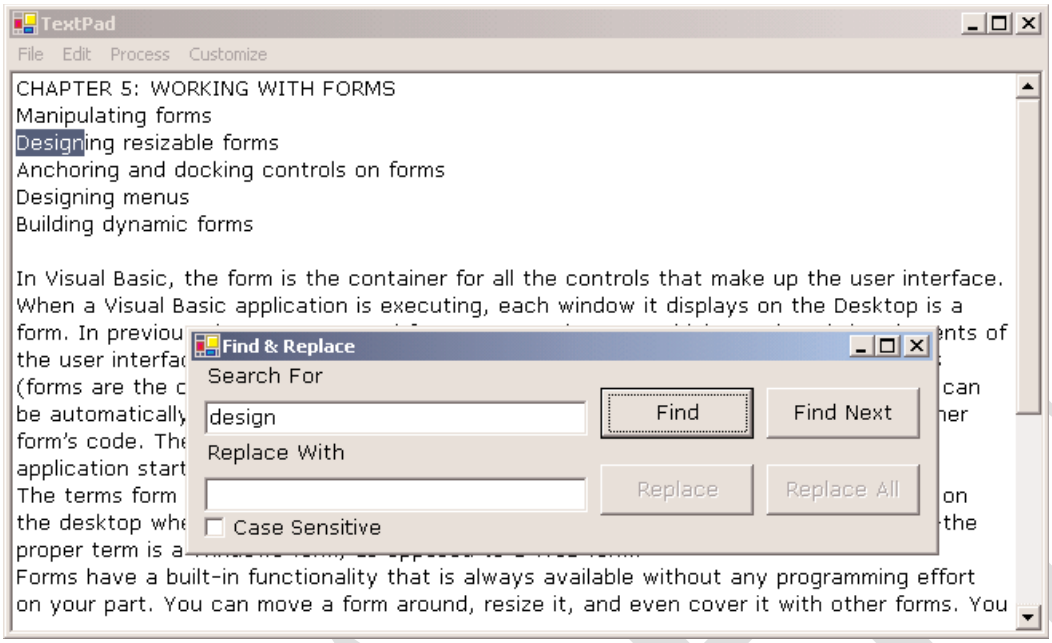
The Main() subroutine usually contains initialization code and ends with a statement that displays one of the project's forms; to display the AuxiliaryForm object from within the Main() subroutine, use the following statements (I'm showing the entire module's code):

```
Module StartUpModule

Sub Main()

System.Windows.Forms.Application.Run(New AuxiliaryForm())

End Sub

End Module
```

Controlling One Form from within Another

Loading and displaying a form from within another form's code is fairly trivial. In some situations, this is all the interaction you need between forms. Each form is designed to operate independently of the others, but they can communicate via public variables

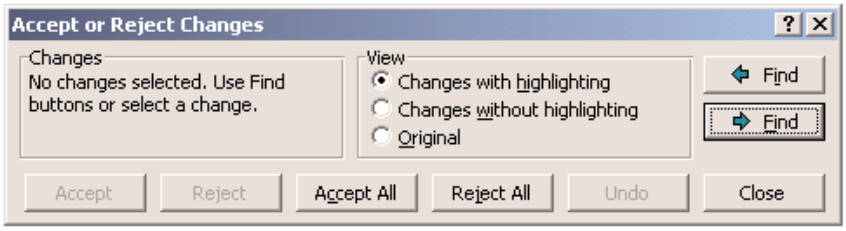
When the user clicks one of the Find & Replace form's buttons, the corresponding code must access the text on the main form of the application and search for a word or replace a string with another. The Find & Replace dialog box not only interacts with the TextBox control on the main form, it also remains visible at all times while it's open, even if it doesn't have the focus, because its TopMost property was set to True

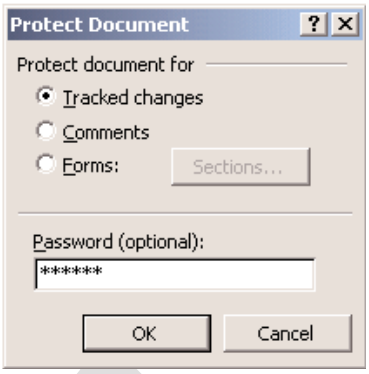


Forms vs. Dialog Boxes

Dialog boxes are special types of forms with rather limited functionality, which we use to prompt the user for data. The Open and Save dialog boxes are two of the most familiar dialog boxes in Windows.

They're so common, they're actually known as *common dialog boxes*. Technically, a dialog box is a good old Form with its BorderStyle property set to FixedDialog. Like forms, dialog boxes may contain a few simple controls, such as Labels, TextBoxes, and Buttons. You can't overload a dialog box with controls and functionality, because you'll end up with a regular form.





Typical dialog boxes of Word

Another difference between forms and dialog boxes is that forms usually interact with each other. If you need to keep two windows open and allow the user to switch from one to the other, you need to implement them as regular forms. If one of them is modal, then you should implement it as a dialog box. A characteristic of dialog boxes is that they provide an OK and a Cancel button. The OK button tells the application that you're done using the dialog box and the application can process the information on it.

The Cancel button tells the application that it should ignore the information on the dialog box and cancel the current operation. As you will see, dialog boxes allow you to quickly

find out which button was clicked to close them, so that your application can take a different action in each case.

In short, the difference between forms and dialog boxes is artificial. If it were really important to distinguish between the two, they'd be implemented as two different objects—but they're the same object.

To create a dialog box, start with a Windows Form, set its BorderStyle property to FixedDialog and set the ControlBox, MinimizeBox, and MaximizeBox properties to False. Then add the necessary controls on the form and code the appropriate events, as you would do with a regular Windows form.

Designing Menus

Menus are one of the most common and characteristic elements of the Windows user interface. Even in the old days of character-based displays, menus were used to display methodically organized choices and guide the user through an application. Despite the visually rich interfaces of Windows applications and the many alternatives, menus are still the most popular means of organizing a large number of options.

Many applications duplicate some or all of their menus in the form of toolbar icons, but the menu is a standard fixture of a form. You can turn the toolbars on and off, but not the menus.

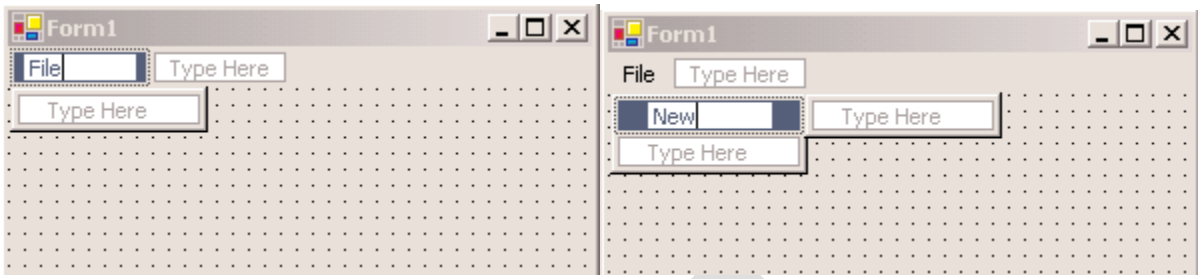
The Menu Editor

Menus can be attached only to forms, and they're implemented through the MainMenu control. The items that make up the menu are MenuItem objects. The MainMenu control and MenuItem objects give you absolute control over the structure and appearance of the menus of your application.

The IDE provides a visual tool for designing menus, and then you can program their Click event handlers. In principle, that's all there is to a menu: you design it, then you program each command's actions.

Depending on the needs of your application, you may wish to enable and disable certain commands, add context menus to some of the controls on your form, and so on. Because each item (command) in a menu is represented by a MenuItem object, you can control the application's menus

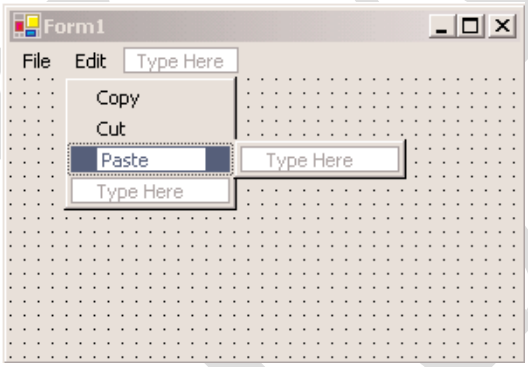
Double-click the MainMenu icon on the Toolbox. The MainMenu control will be added to the form, and a single menu command will appear on your form. Its caption will be Type Here. If you don't see the first menu item on the Form right away, select the MainMenu control in the Components tray below the form. Do as the caption says; click it and enter the first command's caption,



Enter the remaining items of the File menu—**Open**, **Save**, and **Exit**—and then click somewhere on the form. All the temporary items (the ones with the Type Here caption) will disappear, and the menu will be finalized on the form.

At any point, you can add more items by right-clicking one of the existing menu items and selecting Insert New. To add the Edit menu, select the MainMenu icon to activate the visual menu editor and then click the File item.

In the new item that appears next to that, enter the string **Edit**. Press Enter and you'll switch to the first item of the Edit menu.



Each menu item has a name, which allows you to access the properties of the menu item from within your code. The same name is also used in naming the Click event handler of the item. The default names of the menu items you add visually to the application's menu are MenuItem1, MenuItem2, and so on.

To change the default names to something more meaningful, you can change the Name property in the Properties window. To view the properties of a menu item, select it with the left mouse button, then rightclick it and select Properties from the context menu.

The MenuItem Object's Properties

The MenuItem object represents a menu command, at any level. If a command leads to a submenu, it's still represented by a MenuItem object, which has its own collection of MenuItem objects. Each individual command is represented by a MenuItem object. The MenuItem object provides the following properties, which you can set in the Properties window at design time or manipulate from within your code:

Checked

Some menu commands act as toggles, and they are usually checked to indicate that

they are on or unchecked to indicate that they are off. To initially display a check mark next to a menu command, right-click the menu item, select Properties, and check the Checked box in its Properties window.

You can also access this property from within your code to change the checked status of a menu command at runtime. For example, to toggle the status of a menu command called FntBold, use the statement:

FntBold.Checked = Not FntBold.Checked

DefaultItem

This property is a True/False value that indicates whether the MenuItem is the default item in a submenu. The default item is displayed in bold and is automatically activated when the user double-clicks a menu that contains it.

Enabled

Some menu commands aren't always available. The Paste command, for example, has no meaning if the Clipboard is empty (or if it contains data that can't be pasted in the current application).

To indicate that a command can't be used at the time, you set its Enabled property to False. The command then appears grayed in the menu, and although it can be highlighted, it can't be activated.

The following statements enable and disable the Undo command depending on whether the TextBox1 control can undo the most recent operation.

```
If TextBox1.CanUndo Then
cmdUndo.Enabled = True
Else
cmdUndo.Enabled = False
End If
```

cmdUndo is the name of the Undo command in the application's Edit menu. The CanUndo property of the TextBox control returns a True/False value indicating whether the last action can be undone or not.

IsParent

If the menu command, represented by a MenuItem object, leads to a submenu, then that MenuItems object's IsParent property is True. Otherwise, it's False. The IsParent property is read-only.

Mnemonic

This read-only property returns the character that was assigned as an access key to the specific menu item. If no access key is associated with a MenuItem, the character 0 will be returned.

Visible

To remove a command temporarily from the menu, set the command's Visible property

to False. The Visible property isn't used frequently in menu design

MDIList This property is used with Multiple Document Interface (MDI) applications to

maintain a list of all open windows.

Programming Menu Commands

Menu commands are similar to controls. They have certain properties that you can manipulate from within your code, and they trigger a Click event when they're clicked with the mouse or selected with the Enter key. If you double-click a menu command at design time, Visual Basic opens the code for the Click event in the code window.

The name of the event handler for the Click event is composed of the command's name followed by an underscore character and the event's name, as with all othercontrols.

To program a menu item, insert the appropriate code in the MenuItem's Click event handler.

A related event is the Select event, which is fired when the cursor is placed over a menu item, even if it's not clicked. The Exit command's code would be something like:

```
Sub menuExit(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles menuExit.Click
End
End Sub
```

If you need to execute any clean-up code before the application ends, place it in the CleanUp() subroutine and call this subroutine from within the Exit item's Click event handler:

```
Sub menuExit(ByVal sender As Object, ByVal e As System.EventArgs) _
Handles menuExit.Click
CleanUp()
```

End

End Sub

The same subroutine must also be called from within the Closing event handler of the application's main form, as some users might terminate the application by clicking the form's Close button.

An application's Open menu command contains the code that prompts the user to select a file and then open it. All you really need to know is that each menu item is a MenuItem object, and it fires the Click event every time it's selected with the mouse or the keyboard. In most cases, you can treat the Click event handler of a MenuItem object just like the Click event handler of a Button.

You can also program multiple menu items with a single event handler. Let's say you have a Zoom menu that allows the user to select one of several zoom factors. Instead of inserting the same statements in each menu item's Click event handler, you can program all the items of the Zoom menu with a single event handler. Select all the items that share the same event handler (click them with the mouse while holding down the Shift button).

Then click the Event button on the Properties window and select the event that you want to be common for all selected items. The handler of the Click event of a menu item has the following declaration:

Private Sub Zoom200_Click(ByVal sender As System.Object, _

ByVal e As System.EventArgs) Handles Zoom200.Click

End Sub

This subroutine handles the menu item 200%, which magnifies an image by 200%. Let's say the same menu contains the options 100%, 75%, 50%, and 25%, and that the names of these commands are Zoom100, Zoom75, and so on. The common handler for their Click event will have the following declaration:

Private Sub Zoom200_Click(ByVal sender As System.Object, _

ByVal e As System.EventArgs) Handles Zoom200.Click, _

Zoom100.Click, Zoom75.Click, Zoom50.Click, Zoom25.Click

End Sub

The common event handler wouldn't do you any good, unless you could figure out which item was clicked from within the handler's code. This information is in the event's *sender* argument. Convert this argument to the MenuItem type, then look up all the properties of the MenuItem object that received the event. The following statement will print the name of the menu item that was clicked

Console.WriteLine(CType(sender, MenuItem).Text)

When you program multiple menu items with a single event handler, set up a Select Case statement based on the caption of the selected menu item, like the following:

Select Case sender.Text

Case "Zoom In"

{ statements to process Zoom In command }

Case "Zoom Out"

{ statements to process Zoom Out command }

Case "Fit"

{ statements to process Fit command }

End Select

It's also common to manipulate the MenuItem's properties from within its Click event handler.These properties are the same properties you set at design time, through the Menu Editor window. Menu commands don't have methods you can call. Most menu object

properties are toggles. To change the Checked property of the FontBold command, for instance, use the following statement:

FontBold.Checked = Not FontBold.Checked

If the command is checked, the check mark will be removed. If the command is unchecked, the check mark will be inserted in front of its name. You can also change the command's caption at runtime, although this practice isn't common. The Text property is manipulated only when you create dynamic menus, as you will see in the section "Adding and Removing Commands at Runtime."

You can change the caption of simple commands such as Show Tools and Hide Tools. These two captions are mutually exclusive, and it makes sense to implement them with a single command. The code behind this MenuItem examines the caption of the command, performs the necessary operations, and then changes the caption to reflect the new state of the application:

If ShowMenu.Text = "Show Tools" Then

{ code to show the toolbar }

ShowMenu.Text = "Hide Tools"

Else

{ code to hide the toolbar }

ShowMenu.Text = "Show Tools"

End If

Manipulating Menus at Runtime

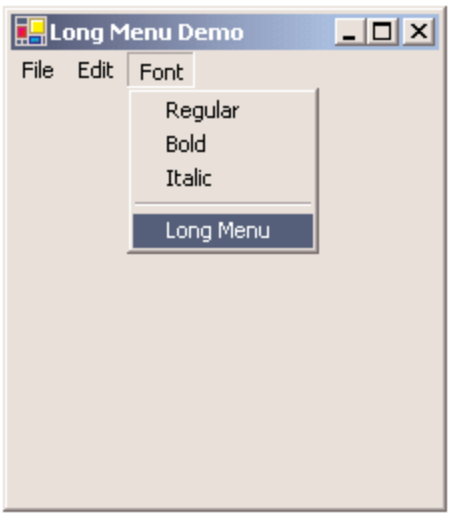
Dynamic menus change at runtime to display more or fewer commands, depending on the current status of the program. Generally there are two techniques to implement the menus as follows

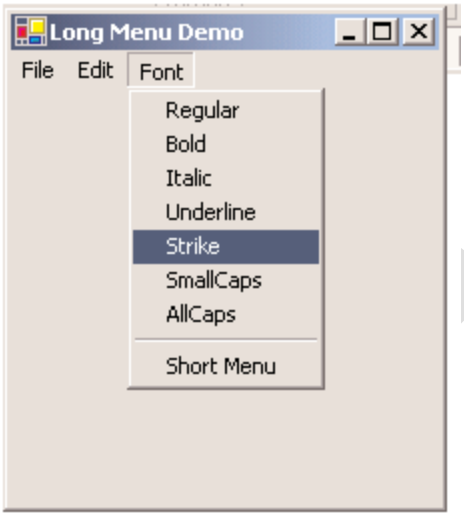
- Creating short and long versions of the same menu
- Adding and removing menu commands at runtime

Creating Short and Long Menus

A common technique in menu design is to create long and short versions of a menu. If a menu contains many commands, and most of the time only a few of them are needed, you can create one menu with all the commands and another with the most common ones. The first menu is the long one, and the second is the short one.

The last command in the long menu should be Short Menu, and when selected, it should display the short version. The last command in the short menu should be Long Menu, and it should display the long version.





Protected Sub menuSize_Click(ByVal sender As Object, _

ByVal e As System.EventArgs)

If MenuSize.text = “Short Menu” Then

MenuSize.text = “Long Menu”

Else

MenuSize.text = “Short Menu”

End If

```
mFontUnderline.Visible = Not mFontUnderline.Visible

mFontStrike.Visible = Not mFontStrike.Visible

mFontSmallCaps.Visible = Not mFontSmallCaps.Visible

mFontAllCaps.Visible = Not mFontAllCaps.Visible

End Sub
```

Adding and Removing commands at runtime

Many applications maintain a list of the most recently opened files intheir File menu. When you first start the application, this list is empty, and as you open and close files, it starts to grow.

The RunTimeMenu project demonstrates how to add items to and remove items from a menu at runtime. The main menu of the application's form contains the Run Time Menu submenu, which is initially empty.

The two buttons on the form add commands to and remove commands from the Run Time Menu. Each new command is appended at the end of the menu, and the commands are removed from the bottom of the menu first (the most recently added commands).

```
Protected Sub bttnRemoveOption_Click(ByVal sender As Object, _

ByVal e As System.EventArgs)

If RunTimeMenu.MenuItems.Count > 0 Then

RunTimeMenu.MenuItems.Remove(RunTimeMenu.MenuItems.count - 1)

End If

End Sub

Protected Sub bttnAddOption_Click(ByVal sender As Object, _
```

```
ByVal e As System.EventArgs)

RunTimeMenu.MenuItems.Add("Run Time Option " & _

RunTimeMenu.MenuItems.Count.toString, _

New EventHandler(AddressOf Me.OptionClick))

End Sub
```

Iterating a Menu's Items

The last menu-related topic in this chapter demonstrates how to iterate through all the items of a menu structure, including their submenus at any depth. The main menu of an application can be accessed by the expression Me.Menu.

This is a reference to the top-level commands of the menu, which appear in the form's menu bar. Each command, in turn, is represented by a MenuItem object. All the MenuItems under a menu command form a MenuItems collection, which you can scan and retrieve the individual commands.

The first command in a menu is accessed with the expression Me.Menu.MenuItems(0); this is the File command in a typical application. The expression Me.Menu.MenuItems(1) is the second command on the same level as the File command (typically, the Edit menu). To access the items *under* the first menu, use the MenuItems collection of the top command. The first command in the File menu can be accessed by the expression

```
Me.Menu.MenuItems(0).MenuItems(0)
```

Question Bank

Part B

1. What is CLR- What is the function of CLR-

2. Explain the term – Garbage Collection
3. What are various access modifiers of VB.Net-
4. Explain the following Controls – TextBox
5. What is event – Enlist various types of Keyboard Eevents.
6. What is Method- What are the types of methods-
7. What is Property Procedure/Properties- How to create it-
8. What is MDI Form- How to create and MDI Application explain with example-

Part C

1. Write VB.Net program to take one list box
2. Write VB .Net program to create Patient table
3. Write a program in VB.Net to do
4. Add the following to your formi)One List Box Control
5. Write VB.Net program to add three text boxes at runtime.
6. Write VB.Net program to search customer record

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
Coimbatore-21

DEPARTMENT OF CS, CA & IT

UNIT-I (Objective Type/ Multiple Choice Questions Each Question Carries One Mark)

.NET PROGRAMMING (17CAP502)

S.No	Question	Option1	Option2	Option3	Option4	Answer
1	.Net is a technology developed by _____ company	Microsoft	Sun Microsystems	IBM	Apple compters	Microsoft
2	_____is also known as the "execution engine" of .NET.	CLR	CTS	MSIL	WPF	CLR
3	Code that targets the Common Langage Runtime is known as _____	Distributed Code	Managed Code	Legacy code	Native Code	Managed Code
4	The _____ defines the minimum standards that .NET language compilers must conform source code compiled by a .NET compiler	CLS	CTS	CLR	MSIL	CLS

5	VB.Net is a _____programming paradigm.	Procedural	Structured	Object Oriented	Monolithic	Object Oriented
6	Data members of a class are by default _____	public	private	static	volatile	private
7	Member functions of a class are by default _____	public	private	static	volatile	public
8	IDE stands for _____	Internet Design Environment	Integrated Development Environment	Internet Distributed Environment	Interface Design Environment	Integrated Development Environment
9	The final compiled version of a Project is ____	Form	Software	Components	Files	Components
10	_____ is a collection of files that can be compiled to create a distributed component	Form	Software	Components	Project	Project
11	_____ is a collection of projects and files that composed an application or component	Solution	Software	Forms	Project	Solution
12	_____menu contains commands for opening and saving projects	File	Edit	View	Project	File

13	Every object has a distinct set of attributes knowns as	members	datas	properties	methods	properties
14	The property that must be set first for any new object is the _____	Name	Colour	Size	Binding	Name
15	Objects that can be placed on a form are called _____	Pictures	Tools	Buttons	Controls	Controls
16	Controls that do not have physical appearance are called _____	invisible-at-runtime-controls	visible-at-runtime-controls	virtual controls	physical controls	invisible-at-runtime-controls
17	The Design window appears_____ by default.	Auto-Hidden	Docked	Floating	Closed	Docked
18	_____ windows appears attached to the side, top or bottom of the work area or to some other window	Auto-Hidden	Docked	Floating	Closed	Docked
19	_____ can be distributed to other people/computer and do not require Visual Basic to run	Files	Forms	Projects	Components	Components
20	_____ is a programming structure that encapsulates data and functionality as a single unit.	Class	Object	Collection	methods	Object

21	_____window gives an overview of the solution we are working with and lists all the files in the project.	Solution Explorer	Properties window	Explorer	Output	Solution Explorer
22	_____window allows us to set properties for various objects at design time	Explorer	Solution Explorer	Properties window	Code Window	Properties window
23	_____window as you can see in the image below displays the results of building and running applications	Command window	output window	Task window	ToolBar Window	output window
24	When we type the period(dot) after the object name a small dropdown list containing all the properties and methods related to that object appears. This feature is called	IntelliSense	OnlineHelp	QuickMenu	DropHelp	IntelliSense
25	A property that returns an object is called_____	Collection	subroutine	Object Oriented	Object Property	Object Property
26	____window displays all the tasks that VB .NET	Task window	output window	command window	Property window	Task window
27	_____is nothing but a name given to a storage area that our programs can manipulate.	Variable name	variable declaration	variable initialization	constants	Variable
28	To declare a variable, use the _____statement followed by the variable's name, the As keyword, and its type,	Dim	integer	String	Dim as	Dim

29	The data type of the variable is defined by using the ------- clause	in	where	as	is	as
30	A composite data type is of ---------- types	3	4	5	2	2
31	Size of integer data type is -------- bits	8	16	24	32	32
32	Which of the given data types used to represent integer numbers	long	char	String	Boolean	long
33	------- is the operator used for string concatenation	Cat	Str	^	&	&
34	The order in which the operators in an expression are evaluated is known as ----------	operator precedence	operator overloading	associativity of operators	operator evaluation	operator precedence
35	Logical operators are also called _____ operators	Boolean	Relational	comparision	String	Boolean
36	In Select Case _____ Case is used to define codes that executes, if the expression does not evaluate to any of the Case statement	Default	Otherwise	Else	False	Else

37	_____data type can be used for currency values	Currency	Dollar	Object	Decimal	Decimal
38	_____ function is used to retrieve only the month part of the date	DateDiff()	DatePart()	DateInterval()	Date.Month()	DatePart()
39	Which function returns the system's current date and time	DateTime.Now	DateTime.Today	DateTime.System	DateTime.Current	DateTime.Now
40	What statement is used to close a loop started with For statement?	Close	End For	Loop	Next	End For
41	What statement is used to terminate a Do..Loop without evaluating the test expression?	End Do	Loop	Exit	Exit Do	Exit Do
42	--------- method is create a new String object with the same content	CopyTo()	Copy()	Format()	Compare()	Copy()
43	The ----- function returns an array of String containing the substrings delimited by the given System.Char array.	Length()	Length()	Split()	Format()	Split()
44	The ---------- function remeove an item from a specified position	Add	Insert()	RemoveAt	Remove	Remove

45	The String data type comes from the ------- class	System.String	System	System.Forms	System.Array	System.String
46	The String is ----------	locatable	mutable	immutable	notable	immutable
47	The --------- function in String Class will insert a String in a specified index in the String instance.	Length()	Insert()	Length()	Format()	Insert()
48	Which of the following when turned on do not allow to use any variable without proper declaration?	Option Restrict	Option Explicit	Option Implicit	Option All	Option Explicit
49	Which of the following methods cane be used to add items to an ArrayList class?	Insert method	collection method	top method	Add method	Add method
50	Parameters to methods in VB.NET are declared by default as ---------	ByVal	ByRef	Val	Ref	ByVal
51	Which of the following does not denote a arithmetic operator allowed in VB.Net?	Mod	/	*	~	~
52	Which of the following denote the method used for compatible type conversions?	TypeCov()	Type()	CTyp()	CType()	CType()

53	Which of the following does not denote a data type in VB.Net?	Boolean	Float	Decimal	Byte	Float
54	The fornat used for Date is ---------	{0:D}	{0:T}	{0:DD}	{0:Dy}	{0:D}
55	The fornat used for Time is ---------	{0:D}	{0:T}	{0:TT}	{0:TTY}	{0:T}
56	_____ is an alternative to If…Then….Else.	select…case	case…select	select	Case	select…case
57	Do Loop While Statement executes a set of statements and checks the condition, this is repeated until the condition is true. .It is also known as ____	Exit control	Entry control	control	loopback	Exit control
58	_____ is the value range of integer	-32767 to 32768	-32768 to 32767	32767 to -32768	32768 to -32767	-32768 to 32767
59	_____ are used for storing values temporarily.	character	constant	variable	module	variable
60	_____ is the value range of byte	0 to 255	1 to 255	0 to 266	1 to 266	0 to 255

UNIT-II

SYLLABUS

Basic Windows Controls: Textbox Control- ListBox, CheckedListBox-Scrollbar and TrackBar Controls-More Windows Control-The common Dialog Controls-The Rich TextBox Control - Handling Strings, characters and Dates. The TreeView and ListView Controls: Examining the Advanced Controls-The TreeView Control-The ListView Control

Basic Windows Controls

The TextBox Control

- The TextBox control is the primary mechanism for displaying and entering text and is one of the most common elements of the Windows user interface.
- The TextBox control is a small text editor that provides all the basic text-editing facilities: inserting and selecting text, scrolling if the text doesn't fit in the control's area, and even exchanging text with other applications through the Clipboard.

Basic Properties

MultiLine

This property determines whether the TextBox control will hold a single line or multiple lines of text. By default, the control holds a single line of text. To change this behavior, set the MultiLine property to True.

ScrollBars

This property controls the attachment of scroll bars to the TextBox control if the text exceeds the control's dimensions.

Single-line text boxes can't have a scroll bar attached, even if the text exceeds the width of the control. Multiline text boxes can have a horizontal or a vertical scroll bar, or both.

If you attach a horizontal scroll bar to the TextBox control, the text won't wrap automatically as the user types. To start a new line, the user must press Enter.

WordWrap

This property determines whether the text is wrapped automatically when it reaches the right edge of the control. The default value of this property is True. If the control has a horizontal scroll bar, however, you can enter very long lines of text. The contents of the control will scroll to the left, so that the insertion point is always visible as you type. You can turn off the horizontal scroll bar and still enter long lines of text; just use the left/right arrows to bring any part of the text into view.

AcceptsReturn, AcceptsTab

These two properties specify how the TextBox control reacts to the Return (Enter) and Tab keys. The Enter key activates the default button on the form, if there is one. The default button is usually an OK button that can be activated with the Enter key, even if it doesn't have the focus. In a multiline TextBox control, however, we want to be able to use the Enter key to change lines.

The AcceptsTab property determines how the control reacts to the Tab key. Normally, the Tab key takes you to the next control in the tab order. In a TextBox control, however, you may wish for the Tab key to insert a Tab character in the text of the control instead; to do this, set this property to True. The default value of the AcceptsTab property is False, so that users can move to the next control with the Tab key.

MaxLength

This property determines the number of characters the TextBox control will accept. Its default value is 32,767, which was the maximum number of characters the VB6 version of the control could hold. Set this property to zero, so that the text can have any length, up to the control's capacity limit 2 GB, or 2,147,483,647 characters to be exact.

Text-Manipulation Properties

Most of the properties for manipulating text in a TextBox control are available at runtime only.

Text

The most important property of the TextBox control is the Text property, which holds the control's text. This property is also available at design time so that you can assign some initial text to the control.

There are two methods of setting the Text property at design time.

For single-line TextBox controls, set the Text property to a short string, as usual. For multiline TextBox controls, open the Lines property and enter the text on the String Collection Editor window, which will appear.

At runtime, use the Text property to extract the text entered by the user or to replace the existing text by assigning a new value to the property. The Text property is a string and can be used as argument with the usual string-manipulation functions of Visual Basic. It also supports all the members of the String class.

The following expression returns the number of characters in the *TextBox1* control:

```
Dim strLen As Integer
strLen = TextBox1.Text.Length
```

VB6 programmers are accustomed to calling the Len() function, which does the same:

```
strLen = Len(TextBox1.Text)
```

To clear the control, you can set its Text property to a blank string:

```
TextBox1.Text = ""
```

or call the control's Clear method:

```
TextBox1.Clear
```

The IndexOf method of the String class will locate a string within the control's text. The following statement returns the location of the first occurrence of the string "Visual" in the text:

```
Dim location As Integer
location = TextBox1.Text.IndexOf("Visual")
                    Or
location = Instr(TextBox1.Text, "Visual")
```

The InStr() function allows to specify whether the search will be case-sensitive or not, while the IndexOf method doesn't.

To store the control's contents in a file, use a statement such as

```
StrWriter.Write(TextBox1.Text)
```

To retrieve the contents of the file, use a statement such as
TextBox1.Text = StrReader.ReadToEnd
where *StrReader* and *StrWriter* are two properly declared StreamReader and StreamWriter object variables.

Locating a String in a TextBox

```
Dim startIndex = -1
startIndex = TextBox1.Text.IndexOf("basic", startIndex + 1)
While startIndex > 0
Console.WriteLine("String found at " & startIndex)
startIndex = TextBox1.Text.IndexOf("basic", startIndex + 1)
End While
```

The following statement appends a string to the existing text on the control:
TextBox1.Text = TextBox1.Text & newString

To append a string to a TextBox control, use the following statement:
TextBox1.AppendText(newString)

ReadOnly, Locked

If you want to display text on a TextBox control but prevent users from editing it, you can set the ReadOnly property to True.

To prevent the editing of the TextBox control with VB6, you had to set the Locked property to True.

Lines

Lines is a string array where each element holds a line of text. The first line of the text is stored in the element Lines(0), the second line of text is stored in the element Lines(1), and so on. You can iterate through the text lines with a loop like the following:

```
Dim iLine As Integer
For iLine = 0 To TextBox1.Lines.GetUpperBound(0)– 1
{ process string TextBox1.Lines(iLine) }
Next
```

PasswordChar

Available at design time, this property turns the characters typed into any character you specify.

Text-Selection Properties

The TextBox control provides three properties for manipulating the text selected by the user: SelectedText, SelectionStart, and SelectionLength.

SelectedText

This property returns the selected text, enabling you to manipulate the current selection from within your code. For example, you can replace the selection by assigning a new value to the SelectedText property. To convert the selected text to uppercase, use the ToUpper method of the String class.

```
TextBox1.SelectedText = TextBox1.SelectedText.ToUpper
```

or use the UCase() function of VB6:

```
TextBox1.SelectedText = UCase(TextBox1.SelectedText)
```

To delete the current selection, assign an empty string to the SelectedText property:

```
TextBox1.SelectedText = “”
```

SelectionStart, SelectionLength

The SelectionStart property returns or sets the position of the first character of the selected text in the control’s text.

The SelectionLength property returns or sets the length of the selected text.

```
Dim seekString As String
Dim textStart As Integer
seekString = “Visual”
textStart = TextBox1.Text.IndexOf(seekString)
If textStart > 0 Then
```

```
TextBox1.SelectionStart = selStart – 1
TextBox1.SelectionLength = seekString.Length
End If
TextBox1.ScrollToCaret()
```

HideSelection

The selected text on the TextBox will not remain highlighted when the user moves to another control or form. To change this default behavior, use the HideSelection property. It keeps text highlighted in a TextBox control while another form or a dialog box has the focus.

Text-Selection Methods

In addition to properties, the TextBox control exposes two methods for selecting text. You can select some text with the Select method, whose syntax is shown next:

```
TextBox1.Select(start, length)
```

The Select method is new to VB.NET and is equivalent to setting the SelectionStart and SelectionLength properties. To select the characters 100 through 105 on the control, call the Select method, passing the values 99 and 6 as arguments:

```
TextBox1.Select(99, 6)
```

If the TextBox control contains the string "ABCDEFGHI," then the following statement will select the range "DEFG":

```
TextBox1.Select(3, 4)
```

The following two statements select the text on a TextBox control with the SelectionStart and SelectionLength properties:

```
TextBox1.SelectionStart = selStart – 1
TextBox1.SelectionLength = word.Length
```

These two lines can be replaced with a single call to the Select method:

TextBox1.Select(selStart – 1, word.Length)

where *word* is a string variable holding the selection.

A variation of the Select method is the SelectAll method, which selects all the text on the control.

Undoing Edits

An interesting feature of the TextBox control is that it can automatically undo the most recent edit operation. To undo an operation from within your code, you must first examine the value of the CanUndo property. If it's True, it means that the control can undo the operation; then you can call the Undo method to undo the most recent edit.

The ListBox, CheckedListBox, and ComboBox Controls

The ListBox, CheckedListBox, and ComboBox controls present lists of choices, from which the user can select one or more.

The ListBox control occupies a user-specified amount of space on the form and is populated with a list of items. If the list of items is longer than can fit on the control, a vertical scroll bar appears automatically. The items must be inserted in the ListBox control through the code or via the Properties window. To add items at design time, locate the Items property in the control's Properties window and click the button with the ellipsis. A new window will pop up, the String Collection Editor window, where you can add the items you want to display on the list.

The ComboBox control also contains multiple items but typically occupies less space on the screen. The real advantage to the ComboBox control, however, is that the user can enter new information in the ComboBox, rather than being forced to select from the items listed.

Basic Properties

The ListBox and ComboBox controls provide a few common properties that determine the basic functionality of the control and are set at design time.

IntegralHeight

This property is a Boolean value (True/False) that indicates whether the control's height will be adjusted to avoid the partial display of the last item. When set to True,

the control's actual height may be slightly different than the size you've specified, so that only an integer number of rows are displayed.

Items

The Items property is a collection that holds the items on the control. At design time, you can populate this list through the String Collection Editor window. At runtime you can access and manipulate the items through the methods and properties of the Items collection.

MultiColumn

A ListBox control can display its items in multiple columns, if you set its MultiColumn property to True.

SelectionMode

This property determines how the user can select the list's items and must be set at design time. The SelectionMode property's values determine whether the user can select multiple items.

Table The SelectionMode Enumeration

Value	Description
None	No selection at all is allowed.
One (Default)	Only a single item can be selected.
MultiSimple	Simple multiple selection: A mouse click (or pressing the spacebar) selects or deselects an item in the list. You must click all the items you want to select.
MultiExtended	Extended multiple selection: Press Shift and click the mouse (or press one of the arrow keys) to expand the selection. This will highlight all the items between the previously selected item and the current selection. Press Ctrl and click the mouse to select or deselect single items in the list.

Sorted

If you want the items to be always sorted, set the control's Sorted property to True. This property can be set at design time as well as runtime.

The following items would be sorted as shown:
"AA"
"Aa"
"aA"

Uppercase characters appear before the equivalent lowercase characters, but both upper- and lowercase characters appear together.

Text

The Text property returns the selected text on the control. To access the actual object, use the SelectedItem property.

The Items Collection

To manipulate a ListBox control from within your application, you should be able to:
_ Add items to the list
_ Remove items from the list
_ Access individual items in the list
The items in the list are represented by the Items collection. Each member of the Items collection is an object.

If you add a Color and a Rectangle object to the Items collection with the following statements:

```
ListBox1.Items.Add(Color.Yellow)
ListBox1.Items.Add(New Rectangle(0, 0, 100, 100))

Console.WriteLine(ListBox1.Items.Item(0).G)
  255
Console.WriteLine(ListBox1.Items.Item(1).Width)
  100
```

Add

To add items to the list, use the Items.Add or Items.Insert method. The syntax of the Add method is

```
   ListBox1.Items.Add(item)
```

The *item* parameter is the object to be added to the list. The Add method appends new items to the end of the list, unless the Sorted property has been set to True.

The following loop adds the elements of the array *words* to a ListBox control, one at a time:

```
Dim words(100) As String
{ statements to populate array }

Dim i As Integer
For i = 0 To 99
ListBox1.Items.Add(words(i))
Next
```

Clear

The Clear method removes all the items from the control. Its syntax is quite simple:

```
List1.Items.Clear
```

Count

This is the number of items in the list. Its syntax is

```
ListBox1.Items.Count
```

CopyTo

The CopyTo method of the Items collection retrieves all the items from a ListBox control and stores them to the array passed to the method as argument.

The syntax of the CopyTo method is

```
ListBox1.CopyTo(destination, index)
```

where *destination* is the name of the array that will accept the items and *index* is the index of an element in the array here the first item will be stored.

Insert

To insert an item at a specific location, use the Insert method, whose syntax is:

```
ListBox1.Items.Insert(index, item)
```

where *item* is the object to be added and *index* is the location of the new item.

Remove

To remove an item from the list, you must first find its position (index) in the list, and call the Remove method passing the position as argument:

```
ListBox1.Items.Remove(index)
```

The *index* parameter is the order of the item to be removed,

You can also specify the item to be removed by reference. To remove a specific item from the list, use the following syntax:

```
ListBox1.Items.Remove(item)
```

Contains

Use the Contains method to avoid the insertion of identical objects to the ListBox control. The following statements add a string to the Items collection, only if the string isn't already part of the collection:

```
Dim itm As String = "Remote Computing"
If Not ListBox1.Items.Contains(itm) Then
ListBox1.Items.Add(itm)
End If
```

Selecting Items

The ListBox control allows the user to select either one or multiple items, depending on the setting of the SelectionMode property. In a single-selection ListBox control, you can retrieve the selected item with the SelectedItem property and its index with the SelectedIndex property. SelectedItem returns the selected item, which could be an object. The text that was clicked by the user to select the item is reported by the Text property. If the control allows the selection of multiple items, they're reported with the SelectedItems property. This property is a collection of Item objects and exposes the same members as the Items collection. The SelectedItems.Count property reports the number of selected items.

To iterate through all the selected items in a multiselection ListBox control, use a loop like the following:

```
Dim itm As Object
For Each itm In ListBox1.SelectedItems
Console.WriteLine(itm)
Next
```

Searching

The single most important enhancement to the ListBox control is that it can now locate any item in the list with the FindString and FindStringExact methods. Both methods accept a string as argument (the item to be located) and a second, optional argument, the index at which the search will begin. The FindString method locates a string that partially matches the one you're searching for; Find- StringExact finds an exact match. If you're searching for "Man" and the control contains a name like "Mansfield," FindString will match the item, but FindStringExact will not.

The syntax of both methods is the same:

itemIndex = ListBox1.FindString(searchStr As String)

where *searchStr* is the string you're searching for. An alternative form of both methods allows you to specify the order of the item at which the search will begin:

itemIndex = ListBox1.FindString(searchStr As String, startIndex As Integer)

The *startIndex* argument allows you specify the beginning of the search, but you can't specify where the search will end.

The ComboBox Control

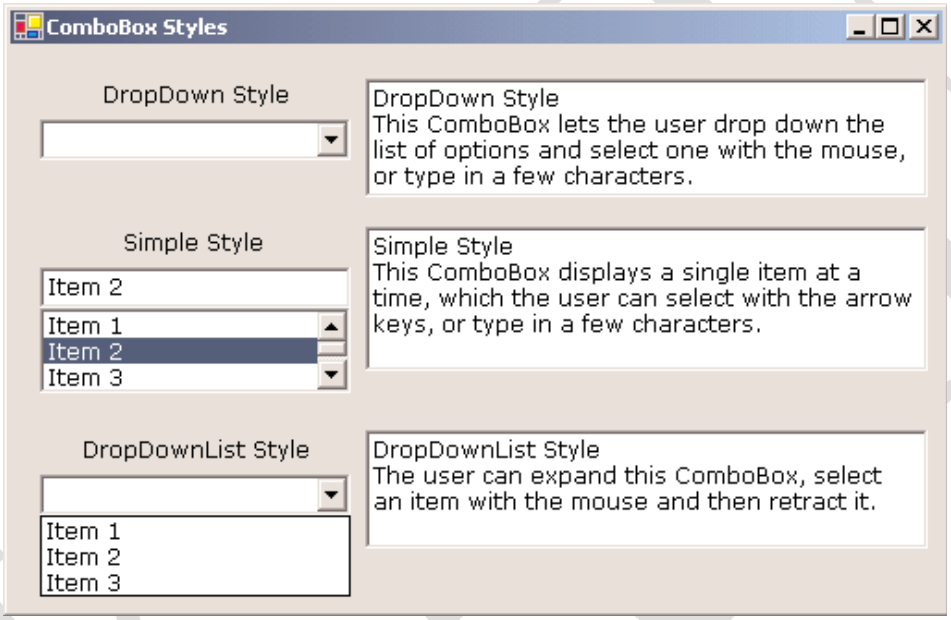
The ComboBox control is similar to the ListBox control in the sense that it contains multiple items of which the user may select one, but it typically occupies less space on-screen. The Text property of the ComboBox is read-only at runtime, and you can locate an item by assigning a value to the control's Text property.

Three types of ComboBox controls are available in Visual Basic.NET. The value of the control's Style property, whose values are as follows

Styles of the ComboBox Control

Value	**Effect**

DropDown (Default)	The control is made up of a drop-down list and a text box. The user can select an item from the list or type a new one in the text box.
DropDownList	This style is a drop-down list, from which the user can select one of its items but can't enter a new one.
Simple	The control includes a text box and a list that doesn't drop down. The user can select from the list or type in the text box.



Adding a New Item to the ComboBox Control at Runtime

```
Private Sub Button1_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) Handles Button1.Click
        Dim itm As String
        itm = InputBox("Enter new item", "New Item")
       If itm <> "" Then AddElement(itm)
  End Sub
```

The AddElement() subroutine, which accepts a string as argument and adds it to the control, is as follows.

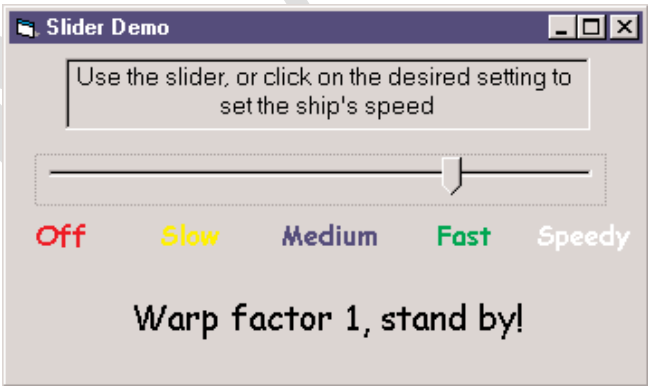
The AddElement() Subroutine

```
Sub AddElement(ByVal newItem As String)
        Dim idx As Integer
       If Not ComboBox1.Items.Contains(newItem) Then
     idx = ComboBox1.Items.Add(newItem)
    Else
 idx = ComboBox1.Items.IndexOf(newItem)
 End If
  ComboBox1.SelectedIndex = idx
End Sub
```

The ScrollBar and TrackBar Controls

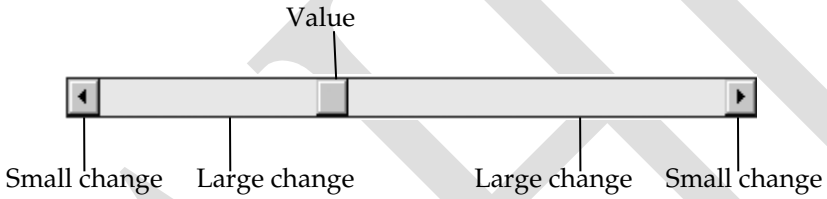
The ScrollBar and TrackBar controls let the user specify a magnitude by scrolling a selector between its minimum and maximum values. In some situations, the user doesn't know in advance the exact value of the quantity to specify (in which case, a text box would suffice), so your application must provide a more flexible mechanism for specifying a value, along with some type of visual feedback.

The TrackBar control is the old Slider control. The TrackBar control is similar to the ScrollBar control, but it doesn't cover a continuous rangeof values. The TrackBar control has a fixed number of tick marks, which the developer can label (e.g., Off, Slow, and Speedy, as shown in the following figure.



The ScrollBar Control

The ScrollBar control is a long stripe with an indicator that lets the user select a value between the two ends of the control, and it can be positioned either vertically or horizontally. Use the Orientation property to make the control vertical or horizontal. The left (or bottom) end of the control corresponds to its minimum value; the other end is the control's maximum value. The current value of the control is determined by the position of the indicator, which can be scrolled between the minimum and maximum values. The basic properties of the ScrollBar control, therefore, are properly named Minimum, Maximum, and Value (see Figure 6.13).



Minimum The control's minimum value. The default value is 0, but because this is an Integer value you can set it to negatives values as well.

Maximum The control's maximum value. The default value is 100, but you can set it to any value you can represent with the Integer data type.

Value The control's current value, specified by the indicator's position.

The ScrollBar Control's Events

The user can change the ScrollBar control's value in three ways:

By clicking the two arrows at its ends. The value of the control changes by the amount specified with the SmallChange property.

By clicking the area between the indicator and the arrows. The value of the control changes by the amount specified with the LargeChange property.

By dragging the indicator with the mouse.

You can monitor the changes on the ScrollBar's value from within your code with two events:

ValueChanged and Scroll. Both events are fired every time the indicator's position is changed. If you change the control's value from within your code, then only the ValueChanged event will be fired.

The Scroll event can be fired in response to many different actions, such as the scrolling of the indicator with the mouse or a click on one of the two buttons at the ends of the scrollbars.

The Actions That Can Cause the Scroll Event

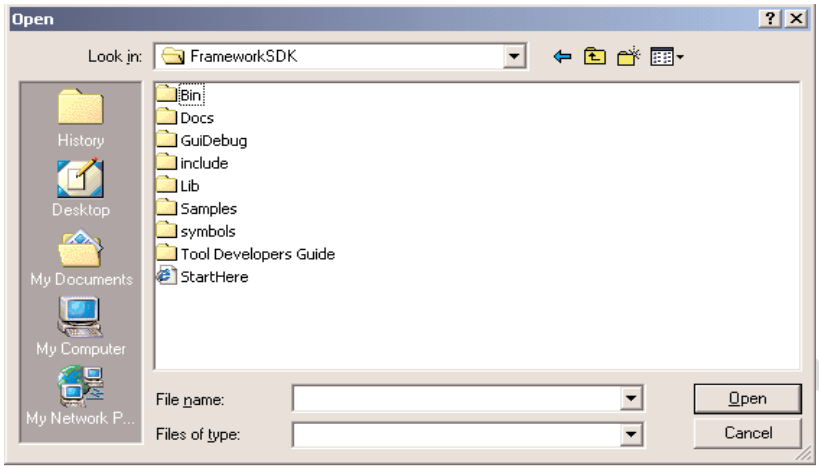
Member	Description
EndScroll	The user has stopped scrolling the control.
First	The control was scrolled to the Minimum position.
LargeDecrement	The control was scrolled by a large decrement (user clicked the bar between the button and the left arrow).
LargeIncrement	The control was scrolled by a large increment (user clicked the bar between the button and the right arrow).
Last	The control was scrolled to the Maximum position.
SmallDecrement	The control was scrolled by a small decrement (user clicked the left arrow).
SmallIncrement	The control was scrolled by a small increment (user clicked the right arrow).
ThumbPosition	The button was moved.
ThumbTrack	The button is being moved.

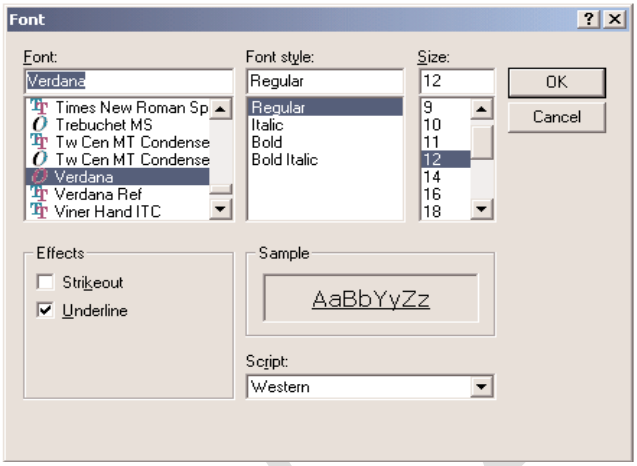
The TrackBar Control

The TrackBar control is similar to the ScrollBar control, but it lacks the granularity of ScrollBar.

Similar to the ScrollBar control, SmallChange and LargeChange properties are available. Small- Change is the smallest increment by which the Slider value can change. The user can only change the slider by the SmallChange value by sliding the indicator. To change the Slider's value by LargeChange, the user can click on either side of the indicator.

The Common Dialog Controls





If you ever want to display an Open or Font dialog box, don't design it—it already exists. To use it, just place the appropriate common dialog control on your form and activate it from within your code by calling the ShowDialog method.

The following common dialog controls are available on the Toolbox.

OpenFileDialog Lets users select a file to open. It also allows the selection of multiple files, for applications that must process many files at once.

SaveFileDialog Lets users select or specify a filename in which the current document will be saved.

ColorDialog Lets users select a color from a list of predefined colors, or specify custom colors.

FontDialog Lets users select a typeface and style to be applied to the current text selection.

PrintDialog Lets users select and set up a printer (the page's orientation, the document pages to be printed, and so on). There are two more common dialog controls, the PrintPreviewDialog and the PageSetupDialog controls.

Using the Common Dialog Controls

To display a common dialog box from within your code, you simply call the control's ShowDialog method, which is common for all controls.

Here is the sequence of statements used to invoke the Open common dialog and retrieve the selected filename:

```
If OpenFileDialog1.ShowDialog = DialogResult.OK Then
  fileName = OpenFileDialog1.FileName
End If
```

The variable *fileName* is the full pathname of the file selected by the user. You can also set the FileName property to a filename, which will be displayed when the Open dialog box is first opened. This allows the user to click the Open button to open the preselected file or choose another file.

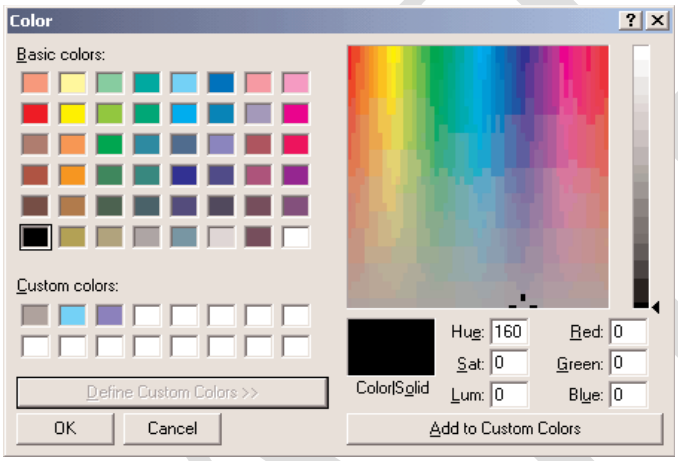
```
OpenFileDialog1.FileName = "C:\Documents\Doc1.doc"
  If OpenFileDialog1.ShowDialog = DialogResult.OK Then
  fileName = OpenFileDialog1.FileName
End If
```

Similarly, you can invoke the Color dialog box and read the value of the selected color with the following statements:

```
If ColorDialog1.ShowDialog = DialogResult.OK Then
    selColor = ColorDialog1.Color
End If
```

The Color Dialog Box

The Color dialog box, shown in Figure 7.2, is one of the simplest dialog boxes. It has a single property, Color, which returns the color selected by the user or sets the initially selected color when the user opens the dialog box. Before opening the Color common dialog with the ShowDialog method, you can set various properties, which are described next.



The following statements set the selected color of the Color common dialog control, display the control, and then use the color selected on the control to fill the form. First, place a ColorDialog control on the form, and then insert the following statements in a button's Click event handler:

```
Private Sub Button1_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) Handles Button1.Click
    ColorDialog1.Color
    ColorDialog1.AllowFullOpen = True
        If ColorDialog1.ShowDialog = DialogResult.OK Then
            Me.BackColor = ColorDialog1.Color
        End If
End Sub
```

AllowFullOpen

Set this property to True if you want users to be able to open up the dialog box and define their own custom colors.

AnyColor

This property is a Boolean value that determines whether the dialog box displays all available colors in the set of basic colors.

Color

This property is a Color value, and you can set it to any valid color. If

```
ColorDialog1.Color = Color.Azure
        If ColorDialog1.ShowDialog = DialogResult.OK Then
            Me.BackColor = ColorDialog1.Color
    End If
```

CustomColors

This property indicates the set of custom colors that will be shown in the common dialog. To display three custom colors in the lower section of the Color dialog box, use a statement like the following:

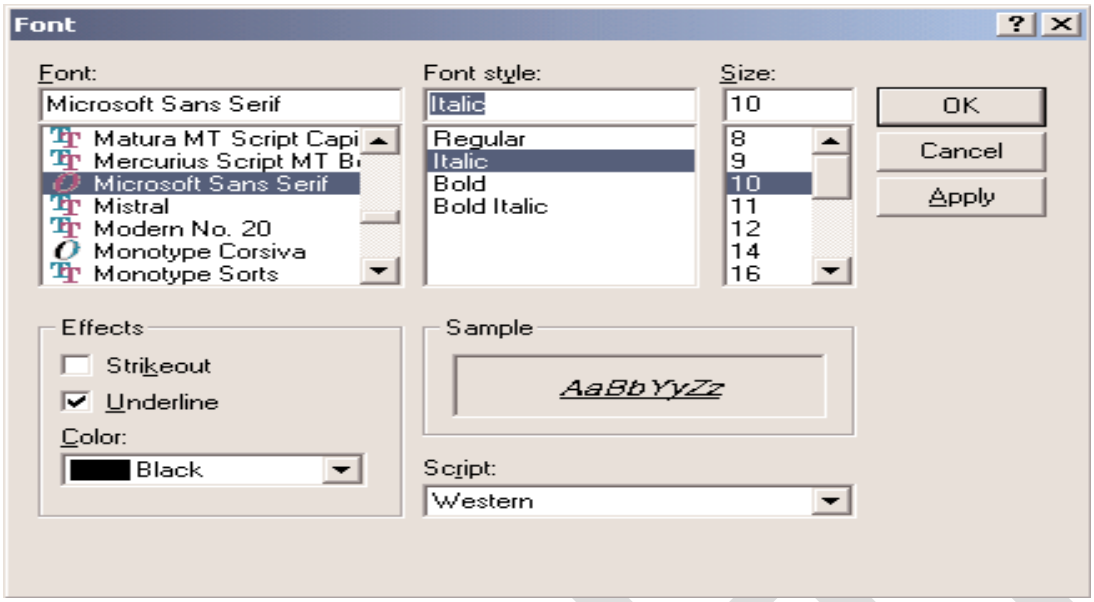
```
Dim colors() As Integer = {222663, 35453, 7888}
ColorDialog1.CustomColors = colors

Dim colors() As Color = {Color.Azure, Color.Navy, Color.Teal}
```

SolidColorOnly

Indicates whether the dialog box will restrict users to selecting solid colors only. This setting should be used with systems that can display only 256 colors.

The Font Dialog Box

The user can also select the font's color and even apply the current dialog-box settings to the selected text on a control of the form without closing the dialog box, by clicking the Apply button on the Font dialog box.




```
Private Sub Button2_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) Handles Button2.Click
        FontDialog1.Font = TextBox1.Font
        If FontDialog1.ShowDialog = DialogResult.OK Then
    TextBox1.Font = FontDialog1.Font
  End If
End Sub
```

AllowScriptChange

This property is a Boolean value that indicates whether the Script combo box will be displayed on the Font common dialog.

AllowSimulations

This property is a Boolean value that indicates whether the dialog box allows the display and selection of simulated fonts.

AllowVectorFonts

This property is a Boolean value that indicates whether the dialog box allows the display and selection of vector fonts.

AllowVerticalFonts

This property is a Boolean value that indicates whether the dialog box allows the display and selection of both vertical and horizontal fonts. Its default value is False, which displays only horizontal fonts.

Color

This property sets or returns the selected font color. The user will see the option to select a color for the selected font only if you set the ShowColor property to True.

FixedPitchOnly

This property is a Boolean value that indicates whether the dialog box allows only the selection of fixed-pitch fonts.

Font

This property is a Font object. The following statements show how to preselect the font of the *TextBox1* control on the Font dialog box and how to change the same control's font to the one selected by the user on the dialog box:

```
FontDialog1.Font = TextBox1.Font
If FontDialog1.ShowDialog = DialogResult.OK Then
    TextBox1.Font = FontDialog1.Font
End If
```

You can create a new Font object and assign it to the control's Font property. The following statements do that:

```
Dim newFont As Font
newFont = New Font("Verdana", 12, FontStyle.Underline)
FontDialog1.Font = newFont
FontDialog1.ShowDialog()
```

FontMustExist

This property is a Boolean value that indicates whether the dialog box forces the selection of an existing font.

MaxSize, MinSize

These two properties are integers that determine the minimum and maximum point size the user can select.

ScriptsOnly

This property indicates whether the dialog box allows selection of fonts for Symbol character sets, in addition to the American National Standards Institute (ANSI) character set. Its default value is True.

ShowApply

This property is a Boolean value that indicates whether the dialog box provides an Apply button.

The following statements display the Font dialog box with the Apply button:

```
Private Sub Button2_Click(ByVal sender As System.Object, _
                          ByVal e As System.EventArgs) Handles Button2.Click
        FontDialog1.Font = TextBox1.Font
        FontDialog1.ShowApply = True
        If FontDialog1.ShowDialog = DialogResult.OK Then
    TextBox1.Font = FontDialog1.Font
End If
End Sub

    Private Sub FontDialog1_Apply(ByVal sender As Object, _
                        ByVal e As System.EventArgs) Handles FontDialog1.Apply
        TextBox1.Font = FontDialog1.Font
    End Sub
```

ShowColor

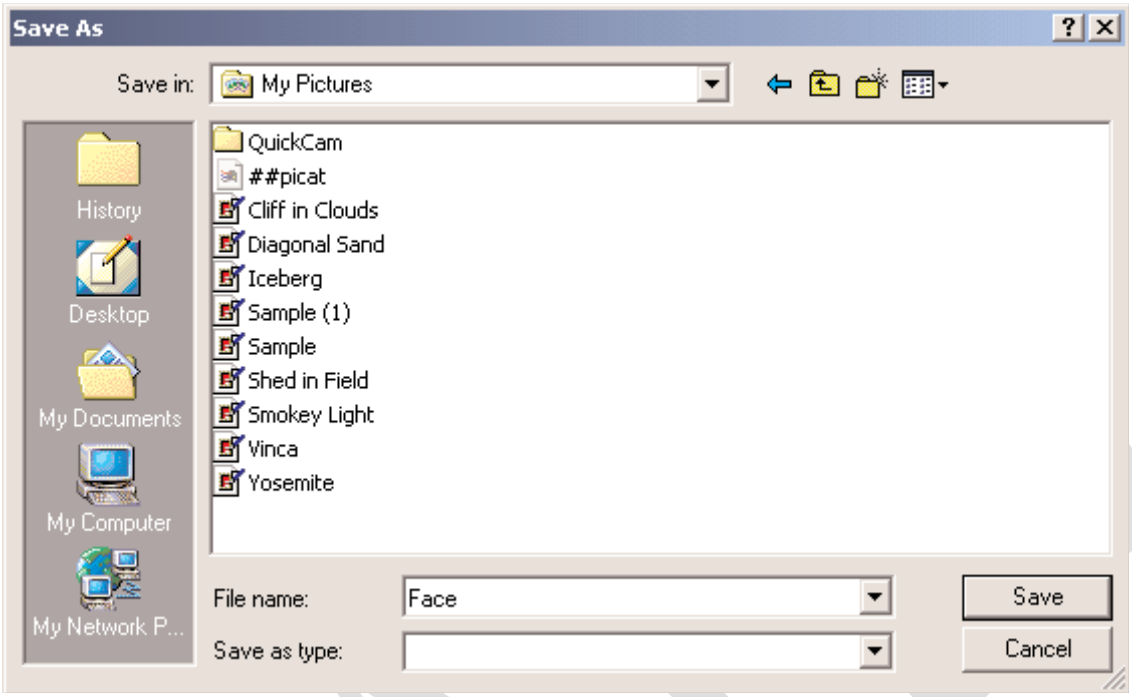
This property is a Boolean value that indicates whether the dialog box allows the user to select a color for the font.

ShowEffects

This property is a Boolean value that indicates whether the dialog box contains controls to allow the user to specify special text effects, such as strikethrough and underline.

The Open and Save As Dialog Boxes

Open and Save As are the two most widely used common dialog boxes, and they're implemented by the OpenFileDialog and SaveFileDialog controls.



The extension of the default file type for the application is described by the DefaultExtension property, and the list of the file types displayed in the Save As Type box is described by the Filter property. Both the DefaultExtension and the Filter properties are available in the control's Properties window at design time. At runtime, you must set them manually from within your code. To prompt the user for the file to be opened, use the following statements. This dialog box displays the files with the extension .BIN only.

```
Private Sub bttnSave_Click(ByVal sender As System.Object, _
                           ByVal e As System.EventArgs) Handles bttnSave.Click
        OpenFileDialog1.DefaultExt = ".BIN"
        OpenFileDialog1.AddExtension = True
        OpenFileDialog1.Filter = "Binary Files|*.bin"
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then
    Console.WriteLine(OpenFileDialog1.FileName)
  End If
End Sub
```

The following sections describe the properties of the OpenFileDialog and SaveFileDialog controls.

AddExtension

This property is a Boolean value that determines whether the dialog box automatically adds an extension to a filename, if the user omits it. The extension added automatically is the one specified by the DefaultExtension property, which must be set before you call the ShowDialog method.

CheckFileExists

This property is a Boolean value that indicates whether the dialog box displays a warning if the user enters the name of a file that does not exist.

CheckPathExists

This property is a Boolean value that indicates whether the dialog box displays a warning if the user specifies a path that does not exist, as part of the user-supplied filename.

DefaultExtension

This property sets the default extension of the dialog box. Use this property to specify a default filename extension, such as TXT or DOC.

DereferenceLinks

This property indicates whether the dialog box returns the location of the file referenced by the shortcut or the location of the shortcut itself.

FileName

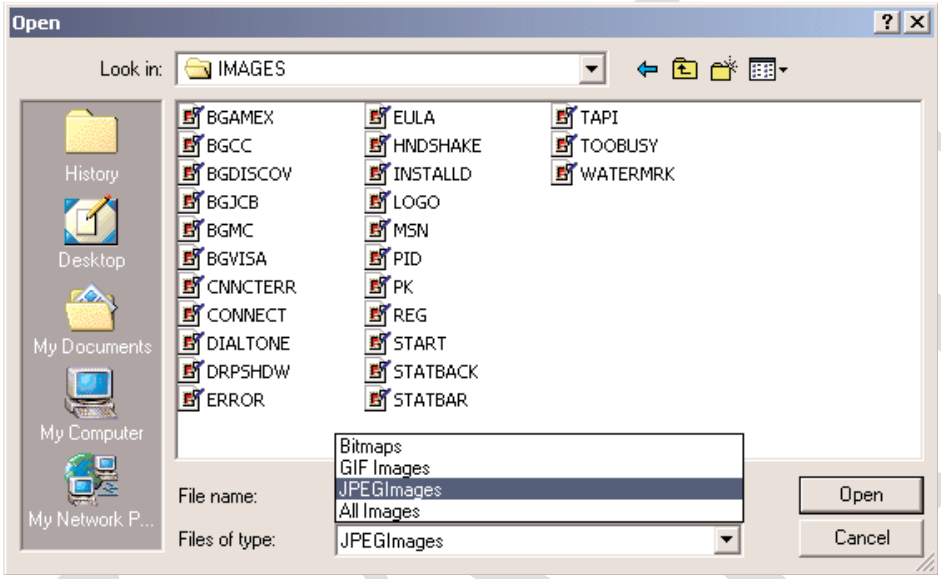
This property is the path of the file selected by the user on the control. If you set this property to a filename before opening the dialog box, this value will be the proposed filename.

Filter

This property is used to specify the type(s) of files displayed on the dialog box. To display text files only, set the Filter property to "Text files|*.txt".

```
OpenFileDialog1.Filter = "Bitmaps|*.BMP|GIF Images|*.GIF|JPEG" & _
                         "Images|*.JPG|All Images|*.BMP;*.GIF;*.JPG"
```

The Open dialog box has four options, which determine what appears in the Save As Type box



FilterIndex

It is used to determine which filter will be displayed as the default when the common dialog is opened.

InitialDirectory

This property sets the initial directory (folder) in which files are displayed the first time the Open and Save dialog boxes are opened.

```
OpenFileDialog1.InitialDirectory = Application.ExecutablePath
```

The expression Application.ExecutablePath returns the path in which the application's executable file resides.

RestoreDirectory

The RestoreDirectory property is a Boolean value that indicates whether the dialog box restores the current directory before closing.

ValidateNames

This property is a Boolean value that indicates whether the dialog box accepts only valid Win32 filenames. Its default value is True, and you shouldn't change it.

FileNames

If the Open dialog box allows the selection of multiple files, the FileNames property contains the pathnames of all selected files.

MultiSelect

This property is a Boolean value that indicates whether the user can select multiple files on the dialog box.

ReadOnlyChecked

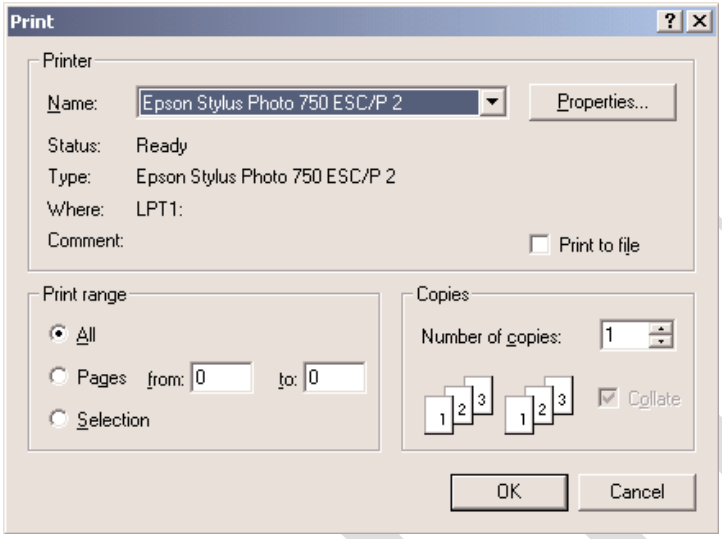
This property is a Boolean value that indicates whether the Read-Only check box is initially selected when the dialog box first pops up.

ShowReadOnly

This property is a Boolean value that indicates whether the Read-Only check box is available.

The Print Dialog Box

The Print dialog box enables users to select a printer, set certain properties of the printout, and set up a specific printer.



AllowPrintToFile This property is a Boolean value that controls whether the user will be given the option to print to a file.

AllowSelection This property is a Boolean value that determines whether the user is allowed to print the current selection of the document.

AllowSomePages This property is a Boolean value that determines whether the Pages option on the dialog will be enabled.

The following statements create a new PrinterSettings object, pass it to the Print dialog box, and then display the dialog box.

```
PrintDialog1.AllowSomePages = Tru
PrintDialog1.AllowSelection = True
PrintDialog1.PrinterSettings = _
                New System.Drawing.Printing.PrinterSettings()
   PrintDialog1.ShowDialog()
Console.WriteLine("FROM PAGE: " &
PrintDialog1.PrinterSettings.FromPage)
Console.WriteLine("TO PAGE: " & PrintDialog1.PrinterSettings.ToPage)
Console.WriteLine("# OF COPIES: " & PrintDialog1.PrinterSettings.Copies)
```

```
Console.WriteLine("PRINTER NAME:" &
PrintDialog1.PrinterSettings.PrinterName)
Console.WriteLine("PRINT RANGE: " & PrintDialog1.PrinterSettings.PrintRange)
Console.WriteLine("LANDSCAPE: " &
PrintDialog1.PrinterSettings.LandscapeAngle)
```

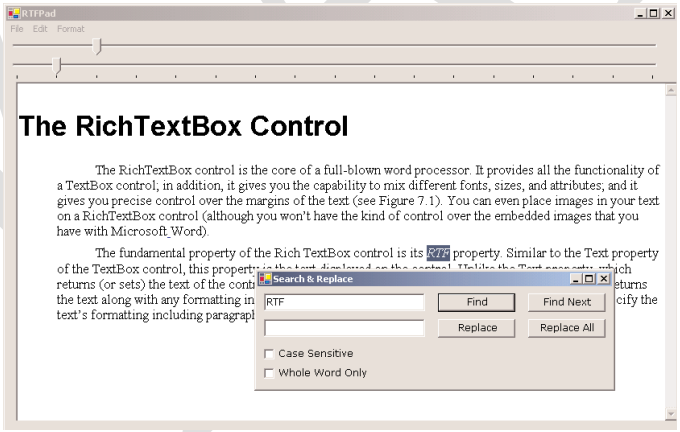
The output produced by the previous statements on my system looked like this:

```
FROM PAGE: 3
TO PAGE: 4
# OF COPIES: 1
PRINTER NAME:Epson Stylus Photo 750 ESC/P 2
PRINT RANGE: 2
LANDSCAPE: 270
```

To set the orientation of the printout, you must click the Properties button on the Print dialogbox.

The RichTextBox Control

It provides all the functionality of a TextBox control; in addition, it gives you the capability to mix different fonts, sizes, and attributes; and it gives you precise control over the margins of the text.

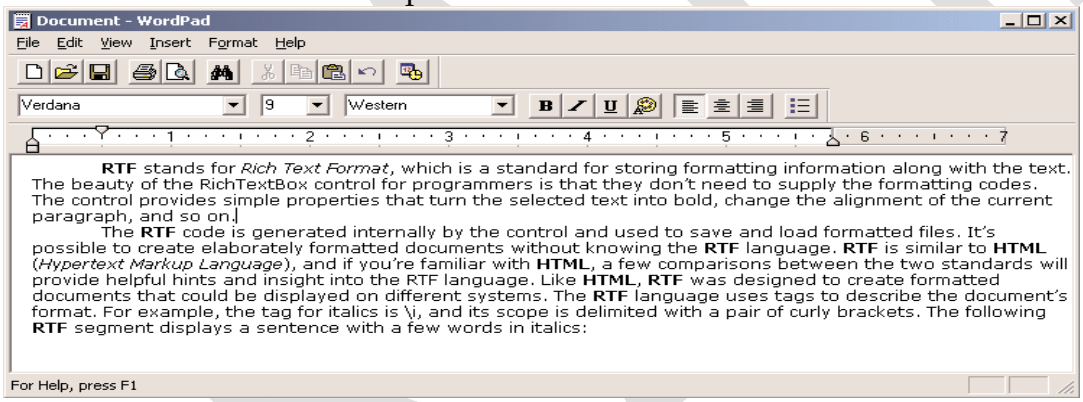


The fundamental property of the RichTextBox control is its *RTF* property. RTF stands for *Rich Text Format,* which is a standard for storing formatting information along with the text.

The RTF Language

RTF is a language that uses simple commands to specify the formatting of a document. These commands, or *tags*, are ASCII strings, such as \par and \b. RTF documents don't contain special characters and can be easily exchanged among different operating systems and computers, as long as there is an RTF-capable application to read the document.

Open the WordPad application (choose Start ➢ Programs ➢ Accessories ➢ WordPad) and enter a few lines of text. Select a few words or sentences and format them in different ways with any of WordPad's formatting commands. Then save the document in RTF format: Choose File ➢ Save As, select Rich Text Format, and then save the file as Document.rtf. If you open this file with a text editor such as Notepad, you'll see the actual RTF code that produced the document.




```
{\rtf1\ansi\ansicpg1252\deff0\deflang1033
{\fonttbl{\f0\fnil\fcharset0 Verdana;}{\f1\fswiss\fcharset0 Arial;}}
\viewkind4\uc1\pard\nowidctlpar\fi720\b\f0\fs18 RTF \b0 stands for \i Rich Text
Format\i0 , which is a standard for storing formatting information along with the
text. The beauty of the RichTextBox control for programmers is that they don\rquote
t need to supply the formatting codes. The control provides simple properties that
turn the selected text into bold, change the alignment of the current paragraph,
and so on.
```

All formatting tags are prefixed with the backslash (\) symbol. To display the \ symbol itself, insert an additional slash. Paragraphs are marked with the \par tag, and the entire document is enclosed in a pair of curly brackets. The \li and \ri tags followed by a numeric value specify the amount of the left and right indentation.

RTF is similar to HTML. RTF was designed to create formatted documents that could be displayed on different systems. The RTF language uses tags to describe the document's format. For example, the tag for italics is \i, and its scope is delimited with a pair of curly brackets. The following RTF segment displays a sentence with a few words in italics:

{{\b RTF} (which stands for Rich Text Format) is a {\i document formatting language} that uses simple commands to specify the formatting of the document.}

The following is the equivalent HTML code:

RTF (which stands for Rich Text Format) is a document formatting language that uses simple commands to specify the formatting of the document.

The RTF Code

If you click the Show RTF button, you'll see the actual RTF code that produced the formatted document. This is all the information the RichTextBox control requires to render the document. The control is responsible for generating the RTF code and for rendering the document. You simply manipulate a few properties (the recurring theme in Visual Basic programming), and the control does the rest.

The RichTextBox's Properties

Property	What It Manipulates
SelectedText	The selected text
SelectedRTF	The RTF code of the selected text
SelectionStart	The position of the selected text's first character
SelectionLength	The length of the selected text
SelectionFont	The font of the selected text
SelectionColor	The color of the selected text
SelectionIndent, SelectionRightIndent, SelectionHangingIndent	The indentation of the selected text
RightMargin	The distance of the text's right margin from the left edge of the control, which is in effect the length of each text line
SelectionBullet	Whether the selected text is bulleted

BulletIndent	The amount of bullet indent for the selected text

SelectedText

The SelectedText property represents the selected text. To assign the selected text to a variable, use the following statement:

```
SText=RichTextbox1.SelectedText
RichTextbox1.SelectedText=UCase(RichTextbox1.SelectedText)
```

SelectionStart, SelectionLength

SelectionStart and SelectionLength, report the position of the first selected character in the text and the length of the selection, respectively.

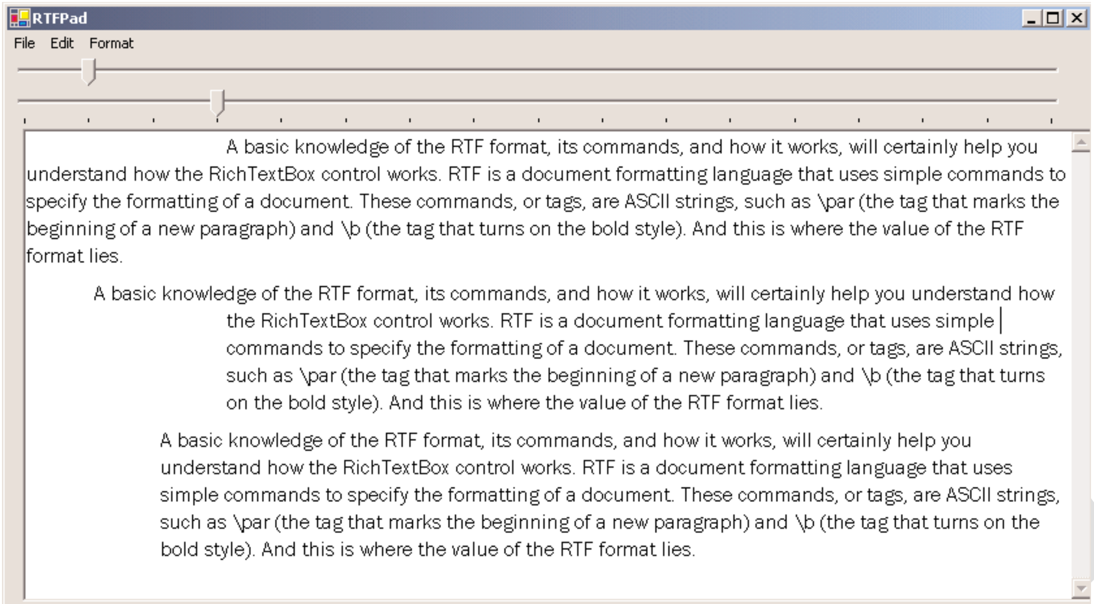
```
RichTextBox1.SelectionStart = 0
RichTextBox1.SelectionLength = Len(RichTextBox1.Text)
```

SelectionAlignment

Use this property to read or change the alignment of one or more paragraphs.

SelectionIndent, SelectionRightIndent, SelectionHangingIndent

These properties allow you to change the margins of individual paragraphs. The SelectionIndent property sets (or returns) the amount of the text's indentation from the left edge of the control. The SelectionRightIndent property sets (or returns) the amount of the text's indentation from the right edge of the control. The SelectionHangingIndent property is the distance between the left edge of the first line and the left edge of the following lines.



SelectionBullet, BulletIndent

You use these properties to create a list of bulleted items. If you set the SelectionBullet property to True, the selected paragraphs are formatted with a bullet style, similar to the <ul> tag in HTML. To create a list of bulleted items, assign the value True to the SelectionBullet property. To change a list of bulleted items back to normal text, make the same property False.

The paragraphs formatted with the SelectionBullet property set to True are also indented from the left by a small amount. To set the amount of the indentation, use the BulletIndent property, whose syntax is

RichTextBox1.BulletIndent

Methods

The first two methods of the RichTextBox control you will learn about are SaveFile and LoadFile:

SaveFile saves the contents of the control to a disk file.

LoadFile loads the control from a disk file.

SaveFile

The syntax of the SaveFile method is

RichTextBox1.SaveFile(path, filetype)

where *path* is the path of the file in which the current document will be saved.

Format	Effect
PlainText	Stores the text on the control without any formatting
RichNoOLEObjs	Stores the text without any formatting and ignores any embedded OLE objects
RichText	Stores the formatted text
TextTextOLEObjs	Stores the text along with the embedded OLE objects
UnicodePlainText	Stores the text in Unicode format

LoadFile

Similarly, the LoadFile method loads a text or RTF file to the control. Its syntax is identical to the syntax of the SaveFile method:

RichTextBox1.LoadFile(path, filetype)

The *filetype* argument is optional and can have one of the values of the RichTextBoxStreamType enumeration.

Select, SelectAll

The Select method selects a section of the text on the control, similar to setting the SelectionStart and SelectionLength properties.

RichTextBox1.Select(start, length)

The SelectAll method accepts no arguments and selects all the text on the control.

Advanced Editing Features

CanUndo, CanRedo

These two properties are Boolean values you can read to find out whether an operation can be undone or redone. The following statements disable the Undo command if there's no action to be undone at the time, where *EditUndo* is the name of the Undo command on the Edit menu:

```
If RichTextBox1.CanUndo Then
    EditUndo.Enabled = True
Else
    EditUndo.Enabled = False
End If
```

UndoActionName, RedoActionName

These two properties return the name of the action that can be undone or redone. The most common value of both properties is the string "typing," which indicates that the Undo command will delete a number of characters..

The following statement sets the caption of the Undo command to a string that indicates the action to be undone:

EditUndo.Text = "Undo " & Editor.UndoActionName

Undo, Redo

These two methods undo or redo an action. The Undo method cancels the effects of the last action of the user on the control. The Redo method redoes the last action that was undone.

Cutting and Pasting

To cut, or copy, and paste text on the RichTextBox control, you can use the same techniques as with the regular TextBox control. For example, you can replace the current selection by assigning a string to the SelectedText property.

```
If Clipboard.GetDataObject.GetDataPresent(DataFormats.Text) Then
    RichTextBox.Paste(DataFormats.Text)
End If
```

Searching in a RichTextBox Control

The Find method locates a string in the control's text and is similar to the InStr() function. You can use InStr() with the control's Text property to locate a string in the text, but the Find method is optimized for the RichTextBox control and supports a couple of options that the InStr() function doesn't. The simplest form of the Find method is the following:

```
RichTextBox1.Find(string)
```

The *string* argument is the string you want to locate in the RichTextBox control.

```
RichTextBox1.Find(string, searchMode)
```

The *searchMode* argument is a member of the RichTextBoxFinds enumeration, which are shown in Table.

Table : The RichTextBoxFinds Enumeration

Value	Effect
MatchCase	Performs a case-sensitive search.

NoHighlight	The text found will not be highlighted.
None	Locates instances of the specified string even if they're not whole words.
Reverse	The search starts at the end of the document.
WholeWord	Locate only instances of the specified string that are whole words.

Two more forms of the Find method allow you specify the range of the text in which the search will take place:

RichTextBox1.Find(string, start, searchMode)
RichTextBox1.Find(string, start, end, searchMode)

The arguments *start* and *end* are the starting and ending locations of the search (use them to search for a string within a specified range only). If you omit the *end* argument, the search will start at the location specified by the *start* argument and will extend to the end of the text.

The String Class

The String class implements the String data type.

Properties

The String class exposes only two properties, the Length and Chars properties, which return a string's length and its characters respectively. Both properties are read-only.

Length

The Length property returns the number of characters in the string, and it's read-only. To find out the number of characters in a string vaiable, use the following statement:

chars = myString.Length

Chars

The Chars property is an array of characters that holds all the characters in the string.

Listing : Validating a Password

```
Private Function ValidatePassword(ByVal password As String) As Boolean
        If password.Length < 6 Then
            MsgBox("The password must be at least 6 characters long")
            Return False
        End If
        Dim i As Integer
```

```
        Dim valid As Boolean = False
        For i = 0 To password.Length - 1
            If Not Char.IsLetterOrDigit(password.Chars(i)) Then Return True
        Next
        MsgBox("The password must contain at least one " & _
                "character that is not a letter or a digit.")
        Return False
    End Function
```

Methods

Compare

This method compares two strings and returns a negative value if the first string is less than the second, a positive value if the second string is less than the first, and zero if the two strings are equal. The Compare method is overloaded, and the first two arguments are always the two strings to be compared.

The simplest form of the method accepts two strings as arguments:

```
String.Compare(str1, str2)
```

The following form of the method accepts a third argument, which is a True/False value and determines whether the search will be case-sensitive (if True) or not:

```
String.Compare(str1, str2, case)
```

Another form of the Compare method allows you to compare segments of two strings; its syntax is

```
String.Compare(str1, index1, str2, index2, length)
```

index1 and *index2* are the starting locations of the segment to be compared in each string.

CompareOrdinal

The CompareOrdinal method compares two strings similar to the Compare method, but it doesn't take into consideration the current locale. This method returns zero if the two strings are the same, but a positive or negative value if they're different. These values are not 1 and –1.

Concat

This method concatenates two or more strings and forms a new string. The simpler form of the Concat method has the following syntax, and it's equivalent to the & operator:

```
newString = String.Concat(string1, string2)
```

This statement is equivalent to the following:

```
newString = string1 & string2
```

A more useful from of the same method concatenates a large number of strings stored in an array:

```
newString = String.Concat(strings)
```

To use this form of the method, store all the strings you want to concatenate into a string array and then call the Concat method, as shown in this code segment:

```
Dim strings() As String = {"string1", "string2", "string3", "string4"}
Dim longString As String
longString = String.Concat(strings)
```

Copy

The Copy method copies the value of one String variable to another. The last two statements in the following sample code are equivalent:

```
Dim s1, s2 As String
s1 = "some text"
 s2 = s1
s2 = String.Copy(s1)
```

The following syntax will also work, because the *s1* variable is an instance of the String class, but it's awkward—almost annoying:

```
s2 = s1.Copy(s1)
```

However, the Copy method doesn't return a copy of the string to which it's applied. The following statement is invalid:

```
s2 = s1.Copy ' INVALID STATEMENT
```

EndsWith, StartsWith

These two methods return True if the string ends or starts with a user-supplied substring. The syntax of these methods is

```
found = str.EndsWith(string)
found = str.StartsWith(string)
```

IndexOf, LastIndexOf

The IndexOf method starts searching from the beginning of the string, and the LastIndexOf method starts searching from the end of the string.

To locate a single character in a string, use the following forms of the IndexOf method:

String.IndexOf(ch)
String.IndexOf(ch, startIndex)
String.IndexOf(ch, startIndex, count)

The *startIndex* argument is the location in the string, where the search will start, and the *count* argument is the number of characters that will be examined.

IndexOfAny

This method accepts as argument an array of characters and returns the first occurrence of any of the array's characters in the string:

str.IndexOfAny(chars)

where *chars* is an array of characters. This method attempts to locate the first instance of any member of *chars* in the string.

Insert

The Insert method inserts one or more characters at a specified location in a string and returns the new string. The syntax of the Insert method is

newString = str.Insert(startIndex, subString)

startIndex is the position in the *str* variable, where the string specified by the second argument will be inserted.

Join

This method joins two or more strings and returns a single string with a separator between the original strings. Its syntax is

newString = String.Join(separator, strings)

where *separator* is the string that will be used as separator and *strings* is an array with the strings to be joined.

Split

You can split a long string into smaller ones with the Split method, whose syntax is

strings() = String.Split(delimiters, string)

where *delimiters* is an array of characters and *string* is the string to be split.

Remove

The Remove method removes a given number of characters from a string, starting at a specific location, and returns the result as a new string. Its syntax is

newString = str.Remove(startIndex, count)

where *startIndex* is the index of the first character to be removed in the *str* string variable and *count* is the number of characters to be removed.

Replace

This method replaces all instances of a specified character in a string with a new one. It creates a new instance of the string, replaces the characters as specified by its arguments, and returns this string:

newString = str.Replace(oldChar, newChar)

where *oldChar* is the character in the *str* variable to be replaced and *newChar* is the character to replace the occurrences of *oldChar*.

PadLeft, PadRight

These two methods align the string left or right in a specified field. They both return a fixed-length string with spaces to the right or to the left. Both methods accept the length of the field as argument and return a new string:

```
Dim LPString, RPString As String
LPString = "[" & "Mastering VB".PadRight(20) & "]"
RPString = "[" & "Mastering VB".PadLeft(20) & "]"
```

After the execution of these statements, the values of the *LPString* and *RPString* variables are:

[Mastering VB]
[Mastering VB]

Another form of these methods allows you to specify the character to be used in padding the strings. If you change the calls to the PadLeft and PadRight methods in the last example with the following:

LPString = "Mastering VB".PadRight(20, "@")
RPString = "Mastering VB".PadLeft(20, ".")
then the two strings will be:
Mastering VB@@@@@@@@
.....Mastering VB

The StringBuilder Class

The StringBuilder class stores dynamic strings and exposes methods to manipulate them much faster than the String class. To use the StringBuilder class in an application, you must import the System.Text class (unless you want to fully qualify each instance of the StringBuilder class in your code). Assuming you have imported the System.Text class in your code module, you can create a new instance of the class with the following statement:

Dim txt As New StringBuilder

There are many ways to initialize an instance of the StringBuilder class, but first I must explain the capacity of a StringBuilder object. Since the StringBuilder handles dynamic strings, it's good to declare in advance the size of the string you intend to store in the current instance of the class. The default capacity is 16 characters, and it's doubled automatically every time you exceed it. To set the initial capacity of the StringBuilder class, use the Capacity property. A related property is the Max-Capacity property, which is read-only and returns the maximum length of a string you can store in a StringBuilder variable. This value is approximately 2 billion characters; it's the length of the longest string you can store to an instance of the StringBuilder class.

To create a new instance of the StringBuilder class, you can call its constructor without any arguments, as I did in the preceding example. You can also initialize it by passing a string as argument:

Dim txt As New StringBuilder("some string")

If you can estimate the maximum length of the string you'll store in the variable, you can specify this value with the following form of the constructor, so that the variable need not be resized as you add to it:

Dim txt As New StringBuilder(initialCapacity)

The size you specify is not a hard limit; the variable may grow longer at runtime, and the String-Builder will adjust its capacity.

If you want to specify a *maximum* capacity for your StringBuilder variable, use the following constructor:

```
Dim txt As New StringBuilder(initialCapacity, maxCapacity)
```

Finally, you can initialize a new instance of the StringBuilder class using both an initial and a maximum capacity, as well as its initial value, with the following form of the constructor:

```
Dim txt As New StringBuilder(string, initialCapacity, maxCapacity)
```

All the members of the StringBuilder class are instance members. In other words, you must create an instance of the StringBuilder class before calling any of its properties or methods.

Properties

You have already seen the two basic properties of the StringBuilder class, the Capacity and Max-Capacity properties. In addition, the StringBuilder class provides the Length and Chars properties, which are the same as the corresponding properties of the String class.

Length

This property returns the number of characters in the current instance of the StringBuilder and is an integer value smaller than (or, at most, equal to) the Capacity property.

Chars

This property gets or sets the character at a specified location in the string, and it's an array of characters. Note that the index of the first character is zero.

```
ch = SB.Chars(index)
```

where *ch* is a properly declared Char variable and *SB* is an instance of the StringBuilder class. To set a character's value in the string, use the following statement:

```
SB.Chars(index) = ch
```

Methods

Many of the methods of the StringBuilder class are equivalent to the methods of the String class, but they act directly on the string to which they're applied, and they don't return a new, separate string.

Append

The Append method appends a base type to the current instance of the StringBuilder class, and its syntax is

SB.Append(value)

where the *value* argument can be a single character, a string, a date, or any numeric value. When you append numeric values to a StringBuilder, they're converted to strings; the value appended is the string returned by the type's ToString method. You can also append an object to the String-Builder—the actual string that will be appended is the object's ToString property.

Another form of the Append method allows you to append an array of characters, and it has the following syntax:

SB.Append(chars, startIndex, count)

Or, you can append a segment of a string by specifying the starting location of the substring in the longer string and the number of characters to be copied:

SB.Append(string, startIndex, count)

AppendFormat

The AppendFormat method is similar to the Append method. Before appending the string, however, it formats it. The string to be appended contains format specifications and the appropriate values. The syntax of the AppendFormat method is

SB.AppendFormat(string, values)

The first argument is a string with embedded format specifications and *values* is an array with values (objects, in general), one for each format specification in the *string*. If you have a small number of values to format, up to four, you can supply them as separate arguments separated by commas:

SB.AppendFormat(string, value, value, value, value)

The following statement appends the string "Your balance as of Thursday, May 16, 2002 is $19,950.40" to a StringBuilder variable:

```
Dim statement As New StringBuilder
statement.AppendFormat("Your balance as of {0:D} is ${1: #,###.00}", _
#5/16/2002#, 19950.40)
```

Each format specification is enclosed in a pair of curly brackets, and they're numbered sequentially (from zero). Then there's a colon followed by the actual specification. The D format specification tells the AppendFormat method to format the specified string in long date format. The second format specification, "#,###.00", uses the thousands separator and two decimal digits.

The following statements append the same string, but they pass the values through an array:

```
Dim statement As New StringBuilder
Dim values() As Object = {"5/16/2002", 19950.4}
statement.AppendFormat("Your balance as of {0:D} is ${1:#,###.00}", values)
```

In both cases, the *statement* variable will hold a string like this one:

Your balance as of Thursday, May 16, 2002 is $19,950.40

The format specifications in the original string usually contain formatting characters. For more information on date and time formatting options, see the section on the ToString method of the Date type, later in this chapter.

Insert

This method inserts a string into the current instance of the StringBuilder class, and its syntax is

SB.Insert(index, value)

The *index* argument is the location where the new string will be inserted in the current instance of the StringBuilder, and *value* is the string to be inserted.

Remove

This method removes a number of characters from the current StringBuilder, starting at a specified location; its syntax is

SB.Remove(startIndex, count)

where *startIndex* is the position of the first character to be removed from the string and *length* is the number of characters to be removed.

Replace

This method replaces all instances of a string in the current StringBuilder with another string. The syntax of the Replace method is

SB.Replace(oldValue, newValue)

where the two arguments can be either strings or characters. Another form of the Replace method limits the replacements to a specified segment of the StringBuilder instance:

SB.Replace(oldValue, newValue, startIndex, count)

This method will replace all instances of *oldValue* with *newValue* in the section starting at location *startIndex* and extending *count* characters.

Handling Dates

The DateTime Class

The DateTime class is used for storing date and time values.

```
Dim date1 As Date = #4/15/2001#
Dim date2 As Date = #4/15/2001 14:01:59#
```

Properties

The Date type exposes the following properties, which are straightforward.

Date

The Date property returns the date from a date/time value and sets the time to midnight. The statements:

```
Dim date1 As Date
date1 = Now()
Console.WriteLine(date1)
Console.WriteLine(date1.Date)
```

will print something like the following values in the Output window:

5/29/2001 2:30:17 PM
5/29/2001 12:00:00 AM

DayOfWeek, DayOfYear

These two properties return the day of the week (a number from 1 to 7) and the number of the day in the year (an integer from 1 to 365, or 366 for leap years).

Hour, Minute, Second, Millisecond

These properties return the corresponding time part of the Date value passed as arguments. If the current time is 1:35:22 P.M., the three properties of the DateTime class will return the following values when applied on the current date and time:

```
Console.WriteLine("The current time is " & Date.Now.TimeOfDay.ToString)
Console.WriteLine("The hour is " & Date.Now.Hour)
Console.WriteLine("The minute is " & Date.Now.Minute)
Console.WriteLine("The second is " & Date.Now.Second)
```

Methods

The DateTime class exposes several methods for manipulating dates. The most practical methods add and subtract time intervals to and from an instance of the DateTime class.

Compare

Compare is a shared method that compares two date/time values and returns an integer value indicating the relative order of the two values. The syntax of the Compare method is

```
order = System.DateTime.Compare(date1, date2)
```

where *date1* and *date2* are the two values to be compared. The method returns an integer, which is –1 if *date1* is less than *date2*, 0 if they're equal, and 1 if *date1* is greater than *date2*.

DaysInMonth

This shared method returns the number of days in a specific month. Because February contains a variable number of days depending on the year, the DaysInMonth method accepts as arguments both the month and the year:

```
monDays = System.DateTime.DaysInMonth(year, month)
```

To find out the number of days in February 2009, use the following expression:

```
FebDays = System.DateTime.DaysInMonth(2, 2009)
```

FromOADate

This shared method creates a date/time value from an OLE Automation Date.

```
newDate = System.DateTime.FromOADate(dtvalue)
```

The argument *dtvalue* must be a Double value in the range from –657,434 (first day of year 100) to 2,958,465 (last day of year 9999).

IsLeapYear

This shared method returns a True/False value that indicates whether the specified year is leap or not.

```
Dim leapYear As Boolean
leapYear = System.DateTime.IsLeapYear(year)
```

Add

This method adds a TimeSpan object to the current instance of the Date class. The TimeSpan object represents a time interval and there are many methods to create a TimeSpan object, which are all discussed in the section "The TimeSpan Class" later in this chapter. The following statements create a new TimeSpan object that represents 3 days, 6 hours, 2 minutes, and 50 seconds, and add this TimeSpan to the current date and time. Depending on when these statements are executed, the two date/time values will differ, but the difference between them will always be 3 days, 6 hours, 2 minutes, and 50 seconds:

```
Dim TS As New TimeSpan()
Dim thisMoment As Date = Now()
TS = New TimeSpan(3, 6, 2, 50)
Console.WriteLine(thisMoment)
Console.WriteLine(thisMoment.Add(TS))
```

The values printed in the Output window when I tested this code segment were:

```
2001-04-13 16:26:38
2001-04-16 22:29:28
```

Subtract

This method is the counterpart of the Add method; it subtracts a TimeSpan object from the current instance of the Date class and returns another Date value. The following statements create a new Time-Span object that represents 3 days, 6 hours, 2 minutes, and 50 seconds, and subtracts this TimeSpan from the current date and time. Depending on when these statements are executed, the two date/time values will differ, but the difference between them will always be the same:

```
Dim TS As New TimeSpan()
Dim thisMoment As Date = Now()
TS = New TimeSpan(3, 6, 2, 50)
Console.WriteLine(thisMoment)
Console.WriteLine(thisMoment.Subtract(TS))
```

The values printed in the Output window when I tested this code segment were:

5/29/2001 2:52:03 PM
5/26/2001 8:49:13 AM

Adding Intervals to Dates

Various methods add specific intervals to a date/time value. Each method accepts the number of intervals to add (days, hours, milliseconds, and so on). These methods are simply listed: AddYears, AddMonths, AddDays, AddHours, AddMinutes, AddSeconds, AddMilliseconds, and AddTicks. A tick is 100 nanoseconds and is used for really fine timing operations.

To add 3 years and 12 hours to the current date, use the following statements:

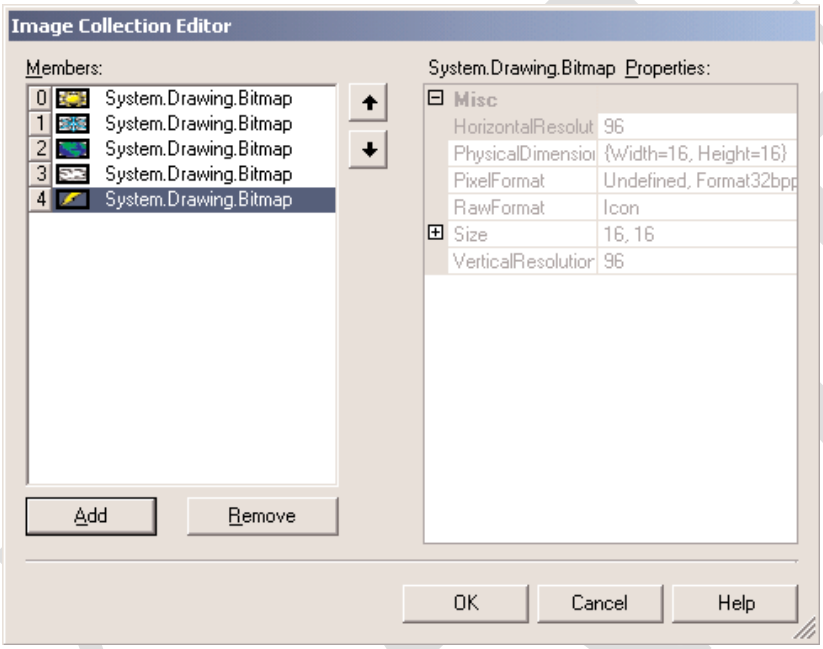
```
Dim aDate As Date
aDate = Now()
aDate = aDate.AddYears(3)
aDate = aDate.AddHours(12)
```

If the argument is a negative value, the corresponding intervals are subtracted from the current instance of the class. The following statement subtracts 2 minutes from a Date variable:

```
aDate = aDate.AddMinutes(-2)
```

The ImageList Control

The ImageList control is a really simple control that stores a number of images used by other controls at runtime. For example, a TreeView control may use a number of icons to identify its nodes. The simplest and quickest method of preparing the images to be used with the TreeView control is to create an ImageList with icons. The ImageList control maintains a series of bitmaps in memory that the TreeView control can access very quickly at runtime.



The other method of adding images to an ImageList control is to call the Add method of the Images collection, which contains all the images stored in the control. The Images collection provides the usual Add and Remove methods. To add an image at runtime, you must first create an Image object with the image (or icon) you want to add to the control and then call the Add method as follows:

```
ImageList1.Images.Add(image)
```

where *image* is an Image object with the desired image. You will usually call this method as follows:

```
ImageList1.Images.Add(Image.FromFile(path))
```

where *path* is the full path of the file with the image. *Images* is a collection of Image objects, not the files where the pictures are stored.

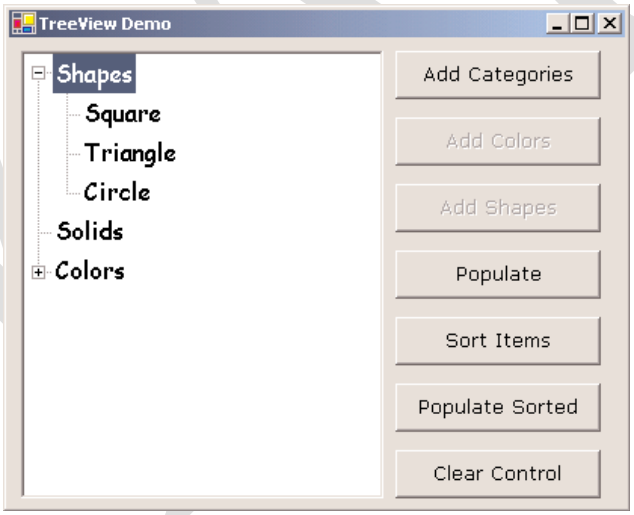
The TreeView Control

CheckBoxes If this option is enabled, a check box appears in front of each item. If the control displays check boxes, you can also select multiple items. If not, you're limited to a single selection.

FullRowSelect This True/False value determines whether the entire row of the selected item will be highlighted and whether an item will be selected even if the user clicks outside the item's text.

HideSelection This property determines whether the selected item will remain highlighted when the focus is moved to another control.

HotTracking This True/False value determines whether items are highlighted as the pointer hovers over them. When this property is True, the TreeView control behaves like a Web document with the items acting as hyperlinks—they turn blue while the pointer hovers over them. However, you can't capture this action from within your code. There's no event to report that the pointer is hovering over an item.



Indent This property indicates the indentation level in pixels. The same indentation applies to all levels of the tree—each level is indented by the same amount of pixels with respect to its parent level.

ShowLines The ShowLines property is a True/False value that determines whether the items on the control will be connected to their parent items with lines.

ShowPlusMinus The ShowPlusMinus property is a True/False value that determines whether the plus/minus button is shown next to tree nodes that have children.

ShowRootLines This is another True/False property that determines whether there will be lines between each node and root of the tree view.

Sorted This property determines whether the items in the control will be automatically sorted or not. The control sorts each level of nodes separately.

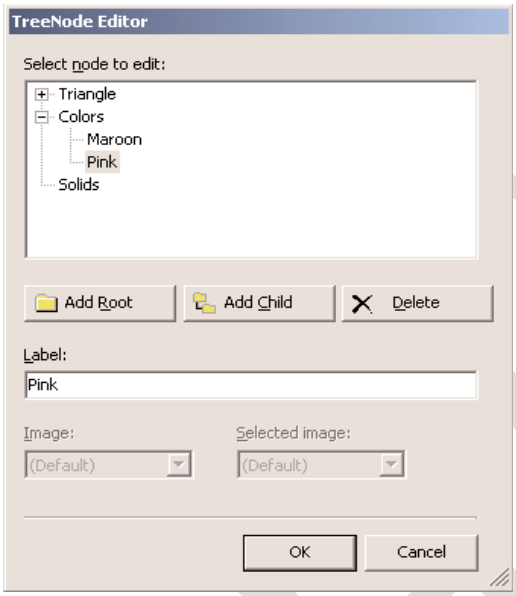
Text This is the text of the currently selected node. Use this property to retrieve the user's selection.

TopNode This is the first visible node in the TreeView control. It's the first node in the control if its contents haven't been scrolled, but it can be any node, at any level, if the control has been scrolled. The TreeNode property returns a TreeNode object, which you can use to manipulate the node from within your code.

VisibleCount This property returns the number of items that are visible on the control.

Adding New Items at Design Time

To add the root item, just click the Add Root button. The new item will be named Node0 by efault. You can change its name by selecting the item in the list. When its name appears in the Label box, change the item's name to anything you like.



You can add items at the same level as the selected one by clicking the Add Root button, or you can add items under the selected node by clicking the Add Child button. The Add Child button adds a new node under the selected node. Follow these steps to enter the root node GLOBE, a child node for Europe, and two more nodes under Europe: Germany and Italy.

Adding New Items at Runtime

Adding items to the control at runtime is a bit more involved. All the items belong to the control's Nodes collection, which is made up of TreeNode objects. To access the Nodes collection, use the following expression, where *TreeView1* is the control's name and *Nodes* is a collection of TreeNode objects:

```
TreeView1.Nodes
```

This expression returns a collection of TreeNode objects, which is called TreeNodeCollection, and it exposes the proper members for accessing and manipulating the individual nodes. The control's Nodes property is the collection of all root nodes.

The following statements will print the strings shown below them in bold (these strings are not part of the statements; they're the output the statements produce).

```
Console.WriteLine(TreeView1.Nodes(0).Text)
GLOBE
Console.WriteLine(TreeView1.Nodes(0).Nodes(0).Text)
 Europe
Console.WriteLine(TreeView1.Nodes(0).Nodes(0).Nodes(1).Text)
 Italy
```

The Nodes.Add Method

The Add method adds a new node to the Nodes collection. The Add method accepts as an argument a string or a TreeNode object. The simplest form of the Add method is

```
newNode = Nodes.Add(nodeCaption)
```

where *nodeCaption* is a string that will be displayed on the control (you can't add objects to the Tree-View control). Another form of the Add method allows you to add a TreeNode object directly:

```
newNode = Nodes.Add(nodeObj)
```

To use this form of the method, you must first declare and initialize a TreeNode object:

```
Dim nodeObj As New TreeNode
nodeObj.Text = "Tree Node"
nodeObj.ForeColor = Color.BlueViolet
TreeView1.Nodes.Add(nodeObj)
```

The TreeNode object exposes a number of properties for setting its appearance. You can change its foreground and background colors, the image to be displayed in front of the node (ImageIndex property), the image to be displayed in front of the node when the node is selected (SelectedImage-Index property), and more, including the NodeFont property. You will see shortly how to assign images to the nodes of a TreeView control.

The last overloaded form of the Add method allows you to specify the index in the current Nodes collection, where the node will be added:

```
newNode = Nodes.Add(index, nodeObj)
```

The *nodeObj* Node object must be initialized as usual. The Add method inserts the new node into the current Nodes collection.

If you call the Add method on the TreeView1.Nodes collection, as we've done in the last few examples, you'll add a root item. If you call it on a child's Nodes collection, you'll add another item to the existing collection of child items. If your control contains a root item already, then this item is given by the expression

```
TreeView1.Nodes(0)
```

To add a child node to the root node, use a statement like the following:

```
TreeView1.Nodes(0).Nodes.Add("Asia")
```

The expression TreeView1.Nodes(0) is the first root node. Its Nodes property represents the nodes under the root node, and the Add method of the Nodes property adds a new node to this collection.

To add another element on the same level as the previous one, just use the same statement with a different argument. To add a country under Asia, use a statement like the following:

```
TreeView1.Nodes(0).Nodes(1).Nodes.Add("Japan")
```

This can get quite complicated, as you can understand. The proper way to add child items to a node is to create a TreeNode variable that represents the parent node, under which the child nodes will be added. The *ContinentNode* variable, for example, represents the node Europe:

```
Dim ContinentNode As TreeNode
ContinentNode = TreeView1.Nodes(0).Nodes(0)
```

The expression TreeView1.Nodes(0) is the first root node. The property Nodes(0) is the third child of the previous node (in our case, the Europe node). Then, you can add child nodes to the *ContinentNode* node:

```
ContinentNode.Nodes.Add("France")
ContinentNode.Nodes.Add("Germany")
```

To add yet another level of nodes, the city nodes, create a new variable that represents the country of the city. The Add method actually returns a TreeNode object, so you can add a country and a few cities with the following statements:

```
Dim CountryNode As TreeNode
CountryNode = ContinentNode.Nodes.Add("Germany")
CountryNode.Nodes.Add("Berlin")
CountryNode.Nodes.Add("Frankfurt")
```

Then, you can continue adding countries through the *ContinentNode* variable:

```
CountryNode = ContinentNode.Nodes.Add("Italy")
CountryNode.Nodes.Add("Rome")
```

The Count Property

This property returns the number of nodes in the Nodes collection. The expression

```
TreeView1.Nodes.Count
```

returns the number of all nodes in the first level of the control. In the case of the Globe example, it returns the value 1. The expression

```
TreeView1.Nodes(0).Nodes.Count
```

returns the number of continents in the Globe example. Again, you can simplify this expression with an intermediate TreeNode object:

```
Dim Continents As TreeNode
Continents = TreeView1.Nodes(0)
Console.WriteLine("There are " & Continents.Nodes.Count.ToString & _
```

" continents on the control")

The Clear Method

The Clear method removes all the child nodes from the current node. To remove all the countries under the Germany node, use a statement like the following:

```
TreeView1.Nodes(0).Nodes(2).Nodes(1).Nodes.Clear
```

The Item Property

The Item property retrieves a node specified by an index value. The expression

```
Nodes.Item(1)
```

is equivalent to the expression

```
Nodes(1)
```

The Remove Method

The Remove method removes a node from the Nodes collection. Its syntax is

```
Nodes.Remove(index)
```

where *index* is the order of the node in the current Nodes collection. To remove the selected node, call the Remove method on the SelectedNode object without arguments:

```
TreeView1.SelectedNode.Remove
```

The FirstNode, NextNode, PrevNode, and LastNode Properties

These four properties allow you to retrieve any node at the current segment of the tree. The FirstNode property will return the first city under Germany and the LastNode will return the last city under Germany. PrevNode and NextNode allow you to iterate through the nodes of the current segment:

Assigning Images to Nodes

To display an image in front of a node's caption, you must first initialize an ImageList control and populate it with all the images you plan to use with the TreeView control. The Node object exposes two image-related properties: ImageIndex and SelectedImageIndex. Both properties are the indices of an image in an ImageList control, which contains the images to be used with the control. To connect the

ImageList control to the TreeView object, you must assign the name of the ImageList control to the ImageList property of the TreeView control. Then you can specify images by their index in the ImageList control.

The ImageIndex property is the index of the image you want to display in front of the node's caption. The SelectedImageIndex is the index of the image you want to display when the node is selected (expanded). Windows Explorer, for example, uses the icon of a closed folder for all collapsed nodes and the icon of an open folder for all expanded nodes. If you don't specify a value for the SelectedImageIndex property, then the image specified with the ImageIndex property will be displayed. If you haven't specified a value for this property either, then no image will be displayed for this node.

The ListView Control

The ListView control is similar to the ListBox control except that it can display its items in many forms, along with any number of subitems for each item. To use the ListView control in your project, place an instance of the control on a form and then sets its basic properties, which are described in the following sections.

The View and Arrange properties There are two properties that determine how the various items will be displayed on the control: the View property, which determines how the items will appear, and the Arrange property, which determines how the items will be aligned on the control's surface.

Table: Settings of the View Property

Setting	Description
LargeIcon	Each item is represented by an icon and a caption below the icon.
SmallIcon	Each item is represented by a small icon and a caption that appears to the right of the icon.
List	Each item is represented by a caption.
Report	Each item is displayed in a column with its subitems in adjacent columns.

The Arrange property determines how the items will be arranged on the control, and its possible settings are show in Table 16.2.

Table : Settings of the Arrange Property

Setting	Description
Default	When an item is moved on the control, it remains where it is dropped.
Left	Items are aligned to the left side of the control.
Snap ToGrid.	Items are aligned to an invisible grid on the control. When the user moves an item, the item moves to the closest grid point on the control.
Top	Items are aligned to the top of the control.

HeaderStyle This property determines the style of the headers in Report view. It has no meaning when the View property is set to something else, because only the Report view has columns. The possible settings for the HeaderStyle property are shown in the following table.

Table : Settings of the HeaderStyle Property

Setting	Description
Clickable	Visible column header that responds to clicking
Nonclickable	Visible column header that does not respond to clicking
None	No visible column header

AllowColumnReorder This property is a True/False value that determines whether the user can reorder the columns at runtime.

Activation This property specifies the action that will activate an item on the control.

Table : Settings of the Activation Property

Setting	Description
OneClick	Items are activated with a single click. When the cursor is over an item, it changes shape and the color of the item's text changes.
Standard.	Items are activated with a double-click. No change in the selected item's text color takes place.
TwoClick	Items are activated with a double-click and their text changes color as well

FullRowSelect This property is a True/False value indicating whether the user can select an entire row or just the item's text, and it's meaningful only in Report view.

GridLines Another True/False property. If True, then grid lines between items and subitems are drawn. This property is meaningful only in Report view.

LabelEdit The LabelEdit property lets you specify whether the user will be allowed to edit the text of the items. The default value of this property is False.

MultiSelect A True/False value indicating whether the user can select multiple items on the control or not. To select multiple items, click them with the mouse while holding down the Shift or the Control key.

Scrollable A True/False value that determines whether the scrollbars are visible or not. Even if the scrollbars are invisible, users will still be able to bring any item into view.

Sorting This property determines how the items will be sorted, and as usual it's meaningful only in Report view.

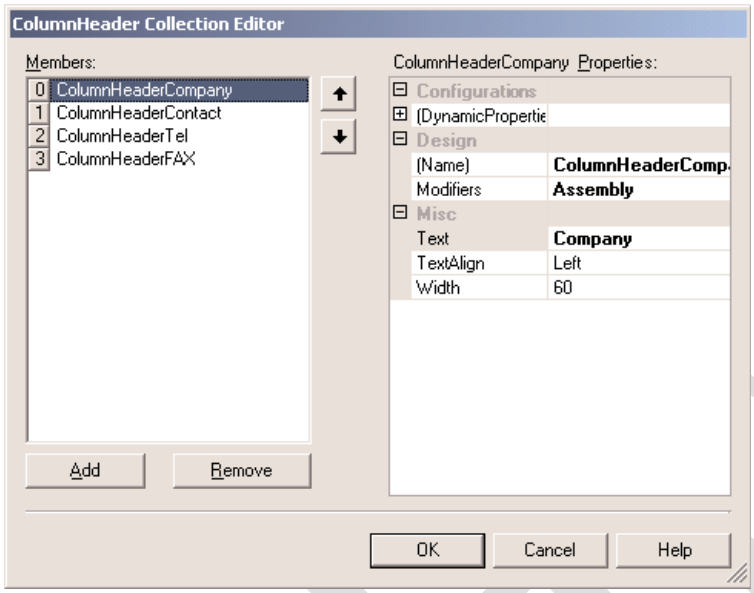
The Columns Collection

To display items in Report view, you must first set up the appropriate columns. The first column corresponds to the item, and the following columns correspond to its subitems.

It is also possible to manipulate the Columns collection from within your code, with the methods and properties discussed here.

Add method Use the Add method of the Columns collection to add a new column to the control. The syntax of the Add method is

TreeView.Columns.Add(header, width, textAlign)



The *header* argument is the column's header. The *width* argument is the column's width in pixels, and the last argument determines how the text will be aligned. The *textAlign* argument can be Center, Justify, Left, NotSet, or Right. The NotSet setting specifies that horizontal alignment is not set.

The Add method returns a ColumnHeader object, which you can use later in your code to manipulate the corresponding column. The ColumnHeader object exposes a Name property, which can't be set with the Add method.

```
Header1 = TreeView1.Add("Column 1", 60, ColAlignment.Left)
Header1.Name = "COL1"
```

After the execution of these statements, the first column can be accessed not only by index but by name as well.

Clear method This method removes all columns.

Count property This property returns the number of columns in the ListView control. You can add more subitems than there are columns in the control, but the excess subitems will not be displayed.

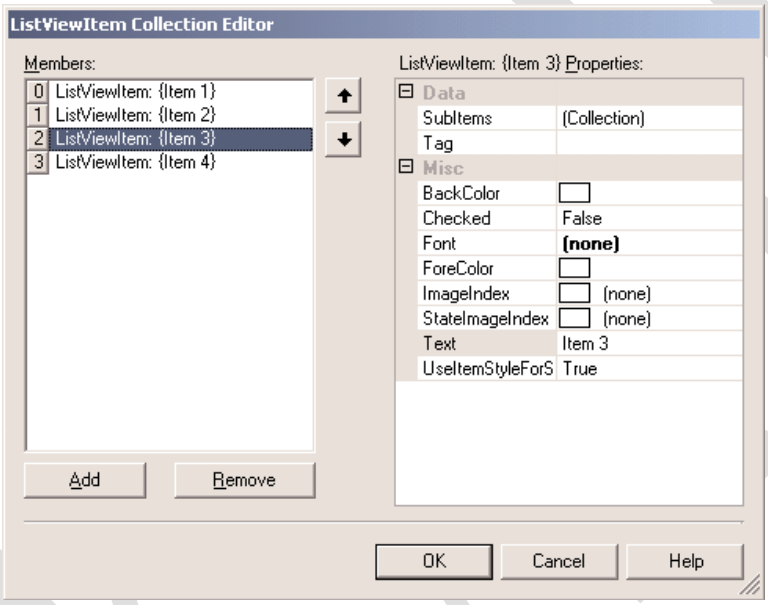
Remove method This method removes a column by its index:

ListView1.Columns(3).Remove

The indices of the following columns are automatically decreased by one.

The ListItem Object

To add items at design time, click the button with the ellipsis next to the ListItems property in the Properties window. When the ListViewItem Collection Editor window pops up, you can enter the items, including their subitems.



Click the Add button to add a new item. Each item has subitems, which you can specify as members of the SubItems collection. To add an item with three subitems, you can populate the SubItems collection with the appropriate elements. Click the button with the ellipsis in the SubItems property on the ListViewItem Collection Editor, and the ListViewSubItem Collection Editor will appear.

BackColor property This property sets or returns the background color of the current item.

Checked property This property controls the status of an item. If it's True, then the item has been selected. You can also select an item from within your code by setting its Checked property to True. The check boxes in front of each item won't be visible unless you set the control's CheckBoxes property to True.

Font property This property sets the font of the current item. Subitems can be displayed in a different font if you specify one with the SetSubItemFont method (see the section "The SubItems Collection," later in this chapter).

Text property This property is the caption of the current item.

SubItems collection This property holds the subitems of the current ListViewItem. To retrieve a specific subitem, use a statement like the following:

```
sitem = ListView1.Items(idx1).SubItems(idx2)
```

where *idx1* is the index of the item and *idx2* the index of the desired subitem.

To add a new subitem to the SubItems collection, use the Add method, passing the text of the subitem as argument:

```
LItem.SubItems.Add("subitem's caption")
```

The argument of the Add method can also be a ListViewItem object. If you want to add a subitem at a specific location, use the Insert method. The Insert method of the SubItems collection accepts two arguments: the index of the subitem before which the new subitem will be inserted and a string or ListViewItem to be inserted:

```
LItem.SubItems.Insert(idx, subitem)
```

Like the ListViewItem objects, each subitem can have its own font, which is set with the Font property.

Remove method This method removes an item by index. When you remove an item, it takes with it all of its subitems.

SetSubItemBackColor method This method sets the background color of the current subitem.

SetSubItemForeColor method This method sets the foreground color of the current subitem. The items of the ListView control can be accessed through the ListItems property, which is a collection. As such, it exposes the standard members of a collection, which are described in the following section.

The Items Collection

All the items on the ListView control form a collection, the Items collection. This collection exposes the typical members of a collection that let you manipulate the control's items. These members are discussed next:

Add method This method adds a new item to the Items collection. The syntax of the Add method is

```
ListView1.Items.Add(caption)
```

You can also specify the index of the image to be used along with the item and a collection of subitems to be appended to the new item, with the following form of the Add method:

```
ListView1.Items.Add(caption, imageIndex)
```

where *imageIndex* is the index of the desired image on the associated ImageList control.

Finally, you can create a ListViewItem object in your code and then add it to the ListView control with the following form of the Add method:

```
ListView1.Items.Add(listItemObj)
```

The following statements create a new item, set its individual subitems, and then add the newly created ListViewItem object to the control:

```
Dim LItem As New ListViewItem()
LItem.Text = "new item"
```

```
LItem.SetSubItem(0, "sub item 1a")
LItem.SetSubItem(1, "sub item 1b")
LItem.SetSubItem(2, "sub item 1c")
ListView1.ListItems.Add(LItem)
```

Count property Returns the number of items in the collection.
Clear method Removes all the items from the collection.
Item property Retrieves an item specified by an index value.
Remove method Removes an item from the Items collection.

The SubItems Collection

Each item in the ListView control may also have subitems. You can think of the item as the key of a record and the subitems as the other fields of the record. The subitems are displayed only in Report mode, but they are available to your code in any view. For example, you can display all items as icons, and when the user clicks on an icon, show the values of the selected item's subitems on other controls.

To access the subitems of a given item, use its SubItems collection. The following statements add an item and three subitems to the ListView1 control:

```
Dim LItem As ListViewItem
Set LItem = ListView1.Items.Add(, , "Alfreds Futterkiste")
LItem.SubItems(1) = "Maria Anders"
LItem.SubItems(2) = "030-0074321"
LItem.SubItems(3) = "030-0076545"
```

Part B

1. Explain following terms:
   Assemblies
   Metadata
   Microsoft Intermediate Language

2. Explain Garbage Collection.
3. What is Console Application? Write a sample console application for input and output of data.

4. Discuss Error and exception handling in .NET
5. What is event? Explain various important .NET Events
6. Discuss about Visual Inheritance

Part C

1) Why windows applications are beneficial? Discuss benefits of Windows Forms
2) Compare ADO.NET with classic ADO
3) Give Motivation for XML Web Services and Discuss how to Create an XML Web Service with Visual Studio.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
Coimbatore-21

DEPARTMENT OF CS, CA & IT

UNIT- II (Objective Type/ Multiple Choice Questions Each Question Carries One Mark)

.NET

PROGRAMMING (17CAP502)

S.No	Question	Option1	Option2	Option3	Option4	Answer
1	The ----- event happens when the mouse pointer hovers over the form/control	MouseWheel	MouseUp	MouseDown	MouseHover	MouseHover
2	the ____occurs when a mouse button is pressed	MouseDown	MouseUp	MouseWheel	MouseHover	MouseDown
3	------ specifies number of times the mouse button is pressed and released	Button	Click	Delta	X	Click
4	If the number of items exceed the value that can be displayed, _____ bars will automatically appear on the control	icon	option button	command button	scroll bars	scroll bars
5	_____ provides easy navigation through a list of items or a large amount of information	scroll bar	command button	tool bar	tool box	scroll bar

6	the _____ToolWindow, but is resizable. In addition, its caption font is smaller than the usual.	Sizable	non sizable	large	small	Sizable
7	The _____ is an one example of breaking a large application into smaller tasks.	Event Handler	Coding	function	subroutine	Event Handler
8	when a mouse button is pressed _____ event will fired	Mouse Enter	Mouse Up	Mouse down	MouseHover	Mouse down
9	_____ is a segment of the code that is executed each time an external condition triggers the event	Event Handler	function	Coding	built-in function	Event Handler
10	Which property has to be set to minimize maximize ot restore a form in code?	Windows Applications	WindowState	FormBorderStyle	WindowSize	WindowState
11	Which property is used to specify the tab order of the various controls	tab order	Accept return	Control Box	Auto tab	tab order
12	The tab order command will appear in which menu	File	Format	View	Edit	View
13	Which property is used to not move the controls around the forms.	Control	Top	Locked	Name	Locked

14	The user action like key press, clicks, mouse movements are called __	Handlers	Triggers	Events	Methods	Events
15	_____ event is fired when a key is released while the control has focus	Key Up	Key press	Key Down	Key Enter	Key Up
16	___ allows the user to open the menu by pressing the Alt key and a letter	Access key	shortcut key	accept key	Tab key	Access key
17	The _____ property is one that automatically activated when you press Enter	Accept button	Cancel button	Control box	Border style	Accept button
18	A ------- is a component used to accept input from the user or display the information on the form	text	container	control	counter	control
19	Which controls do not have events?	TextBox	Label	ToolTip	ImageList	ImageList
20	_____ refers to the position a control has relative to the edge of the form	Anchor	Dock	Key stokes	Key preview	Anchor
21	_____ refers to how much space the control to take up on the form	Anchor	Dock	Key stokes	Key preview	Dock

22	The _____ event takes place every time the form must be refreshed	Resize	Paint	Close	refersh	Paint
23	The default value of FormBorderStyle property is	FixedSingle	FixedToolWindow	Sizable	SizableToolWindow	Sizable
24	The _____ property determines the initial position of the form when its first displayed	initial position	Start position	sizedripstyle	none	Start position
25	the _____ value position the form at the default location and size determined by windows	WindowsDefault Location	WindowsDefault Bounds	Fixed Dialog	Fixed 3D	WindowsDefault Bounds
26	To attach the scroll bar automatically to the form, which property to set true.	Auto Scale	Auto scroll	Auto scroll bar	Auto accept	Auto scroll
27	Without the _____ the form cannot be repositioned by the user	Minimize / Maximize button	Border	Title bar	Control Menu	Title bar
28	_____ method does not simply hides the form, but destroy it completely	Close()	Hide()	Distroy()	Remove()	Close()
29	The simplest method for two forms to communicate with each other is via ____ variables	Private	Common	public	form	public

30	How many parent form will be in MDI	2	0	1	many	1
31	Which class is used to run the EXE application file in VB.NET	Process	Application	Exe	Execute	Process
32	What is the property used to enlarge the inmage in picture box?	Size	SizeMode	Mode	Stretch	SizeMode
33	What is the default event for Picture Box?	Click	Disposed	Layout	Resize	Click
34	The textbox can accept a maximum of ------ characters	1024.00	2048.00	156.00	1028.00	2048.00
35	The -------- property allows you to display multiple lines of text in a textbox control	Text	Multiline	PasswordChar	Autosize	Multiline
36	The -------- property allows automatic resizing of the label control according to the length of its caption	Text	Multiline	PasswordChar	Autosize	Autosize
37	The ---------- is used to display the text as a link	label	textbox	linklabel	listview	linklabel

38	The ---------- property is used to get or set the color used to display the active link	LinkColor	DisabledLinkColor	ActiveLinlColor	LinkVisited	ActiveLinlColor
39	The ---------- property is used to get or set a value indicating whether a link should be displayed as though it was vivited	LinkColor	DisabledLinkColor	LinkVisited	ActiveLinlColor	LinkVisited
40	The ------- property is used to get or set the mode behavior of the listbox control	Sorted	SelectionMode	SelectedIndex	SelectedItem	SelectionMode
41	The ------- property is used to set or retrieve the currently selected item in the combobox control	Sorted	SelectionMode	SelectedIndex	SelectedItem	SelectedItem
42	The --------- control is used to set Yes/No options	CheckBox	RadioButton	GroupBox	Button	CheckBox
43	The --------- control is used to group related controls together	RadioButton	StatusBar	GroupBox	CheckBox	GroupBox
44	The -------- property is used to specify whether or not the statusbar should display panels	Text	Checked	SelectedIndex	ShowPanels	ShowPanels
45	The -------- property is used to specify the location of a control in terms of X and Y coordinates	Name	Visible	Location	Enabled	Location

46	In VB.Net the -------- class is the base class for displaying common dialog boxes.	Inherits	String	CommonDialog	MyBase	CommonDialog
47	The classes that are inherited from the CommonDialog class are categorized as ----------	4.00	6.00	5.00	3.00	5.00
48	Button class is based on ------- class	Stirng	TextBoxBase	ButtonBase	Windows	ButtonBase
49	The ------ property doesn't allow the user to enter	Enabled	Multiline	ReadOnly	TextAllign	ReadOnly
50	The default event of the CheckBox is ----------	Click	CheckedChange	Changed	DoubleClick	CheckedChange
51	RadioButton control is based on the -------- class	Stirng	TextBoxBase	ButtonBase	Windows	ButtonBase
52	The ListBox control is based on the -------- class	Stirng	TextBoxBase	ButtonBase	ListControl	ListControl
53	To display the list as multiple columns in list box --------- property is used	SelectionMode	SelectedIndex	SelectedItem	MultiColumn	MultiColumn

54	The default event of ListBox is the ---------	Click	CheckedChange	DoubleClick	SelectedIndexChanged	SelectedIndexChanged
55	The ---------- property in the Appearance section of the properties window	TextAllign	ReadOnly	Enabled	DropDownStyle	DropDownStyle
56	The ---------- Gets/Sets whether the tree node is checked	ReadOnly	Checked	IsEditing	IsSelected	Checked
57	The --------- Gets the collection of nodes in the current node	Checked	IsEditing	IsSelected	Nodes	Nodes
58	Default event of the Tree View control is the ----------	Click	Selected	AfterSelect	Load	AfterSelect
59	---------- is a combination of a ListBox and a CheckBox	DropDownBox	CheckedListBox	LinkBox	TreeView	CheckedListBox
60	To assign ToolTip's with controls ------- is used	SetTip	SetToolTip	GetTip	SetTool	SetToolTip

UNIT-III

SYLLABUS

The Multiple Document Interface-Databases: Architecture and Basic Concepts-Building Database Application with ADO.NET-Programming with ADO.NET

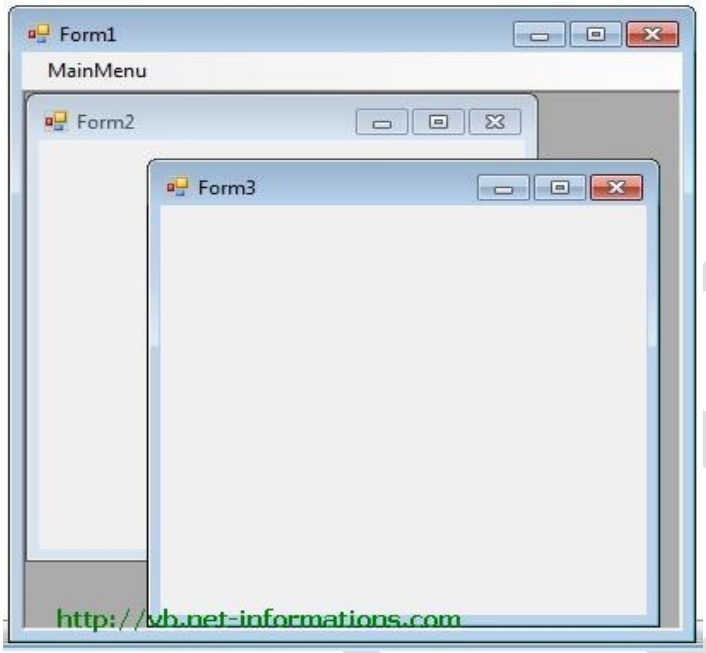
The Multiple Document Interface

A **multiple document interface** (**MDI**) is a graphical user interface in which multiple windows reside under a single parent window. Such systems often allow child windows to embed other windows inside them as well, creating complex nested hierarchies.

MDI applications must have at least two forms, the parent form and one or more child forms. There may be many child forms contained within the parent form, but there can be only one parent form. The parent form is the MDI form, or MDI container, because it contains all child forms.

The parent form may not contain any controls. While the parent form is open in design mode, the icons on the Toolbox aren't disabled, but you can't place any controls on the form. The parent form can, and usually does, have its own menu. While one or more child forms are displayed, the menu of the child forms takes over and it's displayed on the MDI form's menu bar.

A Multiple Document Interface (MDI) programs can display multiple child windows inside them.



This is in contrast to single document interface (SDI) applications, which can manipulate only one document at a time. Visual Studio Environment is an example of Multiple Document Interface (MDI) and notepad is an example of an SDI application, opening a document closes any previously opened document. Any windows can become an MDI parent, if you set the IsMdiContainer property to True.

IsMdiContainer = True

Accessing Child Forms

There are two different methods to access the child forms.

The **first method** is to use the Me keyword. This keyword refers to the form in which the code resides, and since the bulk of the code is on the child form, you can use the Me keyword to access the controls on the child form. The MDI child forms of a text editor, for example, contain a TextBox control where the user can enter and edit text. The following expression returns the text on the active child form:

Me.TextBox1.Text

To select all the text on the active child window, call the TextBox control's SelectAll method with the following statements:

Me.TextBox1.SelectAll

To access the child form from within the MDI parent form's code, you can use the ActiveMdiChild property, which represents the active child form. The following statement returns the caption of the active child form:

Me.ActiveMdiChild.Text

To access the contents of a TextBox control on the child form from within the MDI form's code, use the following statement:

Me.ActiveMdiChild.TextBox1.Text

Using the second method, access all child windows through the MdiChildren property of the parent form. This property returns an array whose elements represent the child forms open on the MDI form at any one time. To find out the number of open child forms, read the Length property of this array:
Console.WriteLine("There are " & Me.MdiChildren.Length & " child Forms open")

This statement works only if it appears in the MDI parent form's code. To access a child form from within another child form's code, you must first access the parent form (Me.ParentForm) and then the parent form's MdiChildren property. The following statements return the values shown in bold if executed from within a child form's code:

Console.WriteLine(Me.ParentForm.MdiChildren.GetLength(0))
3
Console.WriteLine(Me.ParentForm.ActiveMdiChild)

Tracking the Active Child Form

On an MDI form, all child forms are usually instances of one basic form. All forms all have the same behavior, but the operation of each one doesn't affect the others. When a child form is loaded, for example, it will have the same background color as its prototype, but you can change this from within your code by setting the form's BackColor property. No other child form will be affected by that change.

Each child form is totally independent of any other child form, and you can access it from within your code through the Me keyword. This keyword identifies the current form, as long as you program it from within its own form.

Database

A database is an object for storing complex, structured information. It is maintained by special programs, such as Access and SQL Server. These programs are called database management systems (DBMS).

It isolates much of the complexity of the database from the developer. To access the data stored in the database and to update the database, you use a special language, Structured Query Language (SQL).

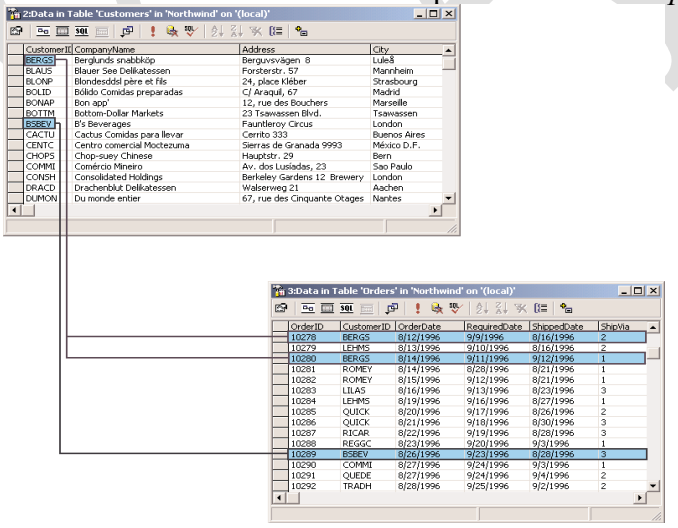
Data are stored in tables, and each table contains entities of the same type.

Relational Database (RDB)

A relational database (RDB) is a collective set of multiple data sets organized by tables, records and columns. RDBs establish a well-defined relationship between database tables. Tables communicate and share information, which facilitates data searchability, organization and reporting.

RDBs use Structured Query Language (SQL), which is a standard user application that provides an easy programming interface for database interaction.

Entities are not independent of each other. For example, orders are placed by specific customers, so the rows of the Customers table must be linked to the rows of the Orders table that store the orders of the customers. Figure shows a segment of a table with customers (top left) and the rows of a table with orders that correspond to one of the customers (bottom right). Thelines that connect the rows of the two tables represent *relationships*.



Relationships are implemented by inserting columns with matching values in the two related tables; the CustomerID column is repeated in both tables. The rows with a common value in their CustomerID field are related. In other words, the lines that connect the two tables simply indicate that there are two fields, one on each side of the relationship, with a common value.

These two fields used in a relationship are called *key fields*. The CustomerID field of the Customers table is the *primary key*, because it identifies a single customer. The CustomerID field of the Orders table is the *foreign key* of the relationship. A CustomerID value appears in a single row of the Customers table; it's the table's primary key. However, it may appear in multiple rows of the Orders table, because in this table the CustomerID field is the foreign key. In fact, it will appear in as many rows of the Orders table as there are orders for the specific customer. The operation of matching rows in two tables based on their primary and foreign keys is called a *join*.

The Visual Database Tools

The Server Explorer. The Server Explorer is the Toolbox for database applications, in the sense that it contains all the basic tools for connecting to databases and manipulating their objects.

The Query Builder This is a tool for creating SQL queries. The Query Builder specify the operations to perform on the tables of a database with point-and-click operations. In the background, the Query Builder builds the appropriate SQL statement and executes it against the database.

The Database Designer and Tables Designer These tools allow to work with an entire database or its tables. In the database, we can add new tables, establish relationships between the tables, and so on. When you work with individual tables, you can manipulate the structure of the tables, edit their data, and add constraints.

The Server Explorer

The Server Explorer is a new development tool in Visual Studio .NET or in Visual Studio 2005 that is shared across development languages and projects. With the Server Explorer, you can connect to servers, as well as view and access their resources.

- Database Connections

- Servers
- Crystal Reports
- Event Logs
- Message Queues
- Performance Counters
- Windows Services

Add a data connection

1. In the Server Explorer window, right-click the **Data Connections** node, and then click **Add Connection**.
2. In the **Data Link Properties** dialog box, type or select a server name in the combo box. For example, if SQL Server is installed on the local computer, type **local**. **Note** In Visual Studio 2005, the **Add Connection** dialog box appears.
3. Type the logon information as necessary for your environment.
4. Select a database to use. For example, if the connection is to a SQL Server, you can click **Northwind**.
5. Click **Test Connection** to verify that the data connection is valid. After a few seconds, the following message appears:
   Test connection succeeded

   If you encounter an error during this test, check the settings, and make any necessary changes.
6. Click **OK**.

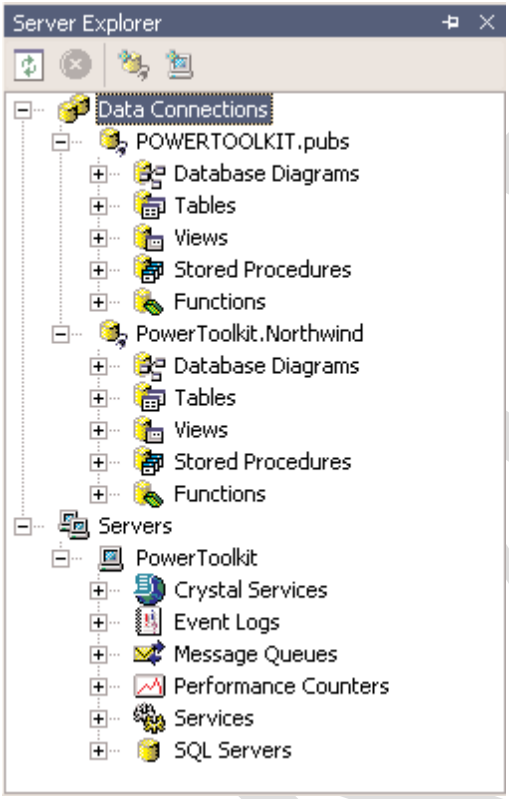
   Notice that the new data connection appears as a child node below the **Data Connections** node.
7. Click to expand the data connection node that you just created.

Add a server

1. In the Server Explorer window, right-click the **Servers** node, and then click **Add Server**.
2. In the **Add Server** dialog box, type a server name that differs from your local server, or type an IP address that differs from your Internet Protocol (IP) address.
3. If you use a different user name to log on to the server, click **Connect using a different user name**, and then type your user name and password.
4. Click **OK**.
   Notice that a new server node appears below the top-level **Servers** node.

5. Click to expand the server node that you just created.

Server Explorer displays the resources that are available to you, such as Crystal Services, Event Logs, Message Queues, Performance Counters, Services and SQL Servers. You receive the same information for your local computer when you click to expand your local server name.



Database Diagrams: A database diagram is a visual representation of a set of related tables, with the relations between the tables. Relations are indicated with line segments between two related tables, and you can quickly learn a lot about the structure of a database by looking at a database diagram.

Tables : We can select a table and edit it, or add a new table to the database. You can edit the table itself (change its design by adding/removing rows or change the data types of one or more columns). Finally, you can view the table's rows and edit them, add new rows, or delete existing rows.

Views This is where you specify the various views you want to use in your applications. Sometimes, the tables are not the most convenient, or even the most expedient, method of looking at your data.
Views are created with SQL SELECT statements. A SELECT statement basically allows you to specify the information you want to retrieve from the database. This information can be stored in a View object, which is just like another table to your application.

Stored Procedures Stored procedures are programs that are stored in the database and perform very specific, and often repeated, tasks. By coding many of the operations you want to perform against the database as stored procedures, you won't have to access the database directly. Moreover, you can call the same stored procedure from several places in your VB code, and you can be sure that the same action is performed every time. Once created, the stored procedure becomes part of the database, and programmers (as well as users) can call it by name, passing the appropriate arguments if necessary. A typical example is a stored procedure for removing orders. The stored procedure must remove the order details first, then remove the order.

Functions The functions of SQL Server are just like the VB functions. They perform specific tasks on the database taking into consideration the arguments passed to the functions when they were called.

Structured Query Language

SQL (Structured Query Language) is a universal language for manipulating tables, and every database management system (DBMS) supports it, so you should invest the time and effort to learn it. You can generate SQL statements with point-and-click operations (the Query Builder is a visual tool for generating SQL statements), but this is no substitute for understanding SQL and writing your own statements.

SQL is a *nonprocedural* language. This means that SQL doesn't provide traditional programming structures like IF statements or loops. Instead, it's a language for specifying the operation you want to perform at an unusually high level. The details of the implementation are left to the DBMS. This is good news for nonprogrammers, but many programmers new to SQL wish it had the structure of a more traditional language. You will get used to SQL and soon be able to combine the best of both worlds: the programming model of VB and the simplicity of SQL.

Example

SELECT StudentName FROM mca

SQL statements are categorized into two major categories, which are actually considered separate languages: the statements for manipulating the data, which form the Data Manipulation Language (DML); and the statements for defining database objects, such as tables or their indexes, which form the Data Definition Language (DDL). The DDL is not of interest to every database developer, and we will not discuss it in this book. The DML is covered in depth, because you'll use these statements to retrieve data, insert new data to the database, and edit or delete existing data.

The statements of the DML part of the SQL language are also known as *queries*, and there are two types of queries: selection queries and action queries. *Selection queries* retrieve information from the database. The queries return a set of rows with identical structure. The columns may come from different tables, but all the rows returned by the query have the same number of columns. *Action queries* modify the database's objects, or create new objects and add them to the database

Selection Queries

We'll start our discussion of SQL with the SELECT statement. Once you learn how to express the criteria for selecting the desired rows with the SELECT statement, you'll be able to apply this information to other data-manipulation statements.
The simplest form of the SELECT statement is

Syntax:

SELECT fields FROM tables

Executing SQL Statements

you have two options, the Query Analyzer and the Query Builder. The Query Analyzer executes SQL statements you design. The Query Builder lets you build the statements with visual tools. After a quick overview of the SQL statements, I will describe the Query Builder and show you how to use its interface to build fairly elaborate queries.

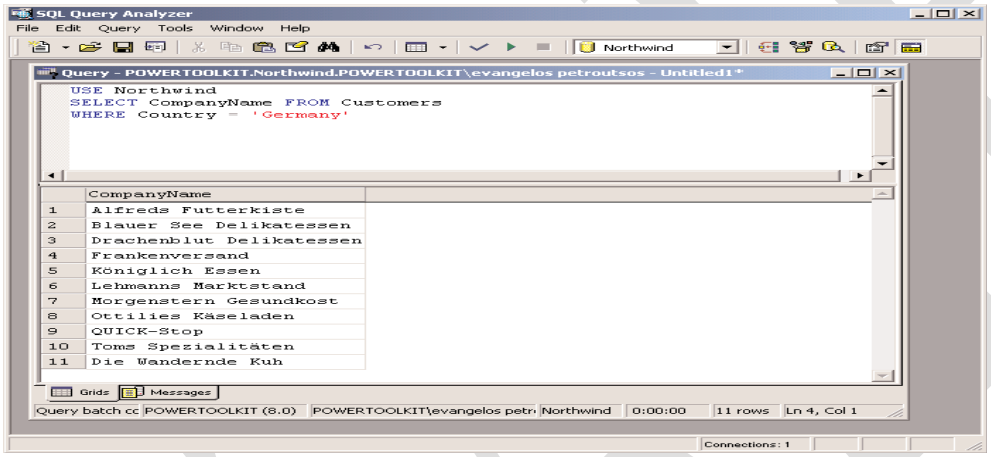
Using the Query Analyzer

One of the applications installed with SQL Server is the Query Analyzer.

To start it, select Start -> Programs ➢ SQL Server ➢ Query Analyzer. Initially, its window will be empty.

First, select the desired database's name in the Database drop-down list and then enter the SQL statement you want to execute in the upper pane. The SQL statement will be executed against the selected database when you press Ctrl+E, or click the Run button

USE Northwind
SELECT CompanyName FROM Customers
WHERE Country = 'Germany'



Selection Queries

We'll start our discussion of SQL with the SELECT statement. Once you learn how to express the criteria for selecting the desired rows with the SELECT statement, you'll be able to apply this information to other data-manipulation statements.

The simplest form of the SELECT statement is
SELECT fields FROM tablesf

WHERE Clause

The unconditional form of the SELECT statement we used in last few examples is quite trivial. You rarely retrieve data from all rows in a table To restrict the rows returned by the query, use the WHERE clause of the SELECT statement. The most common form of the SELECT statement is the following:

SELECT fields FROM tables WHERE condition.

AS Keyword

By default, each column of a query is labeled after the actual field name in the output. If a table contains two fields named CustLName and CustFName, you can display them with different labels using the AS keyword.

The SELECT statement
SELECT CustLName, CustFName

will produce two columns labeled CustLName and CustFName. The query's output will look much better if you change the labels of these two columns with a statement like the following one:

SELECT CustLName AS [Last Name],
CustFName AS [First Name]

It is also possible to concatenate two fields in the SELECT list with the concatenation operator. Concatenated fields are not labeled automatically, so you must supply your own header for the combined field. The following statement creates a single column for the customer's name and labels it.

Customer Name:
SELECT CustFName + ', ' + CustLName AS [Customer Name]

TOP Keyword

The TOP keyword is used only when the rows are ordered according to some meaningful criteria. Limiting a query's output to the alphabetically top *N* rows isn't very practical. When the rows are sorted according to items sold, revenue generated, and so on, it makes sense to limit the query's output to *N* rows. You'll see many examples of the TOP keyword later in this chapter, after you learn how to order a query's rows

DISTINCT Keyword

The DISTINCT keyword eliminates any duplicates from the cursor retrieved by the SELECT statement. Let's say you want a list of all countries with at least one customer. If you retrieve all country names from the Customers table, you'll end up with many duplicates. To eliminate them, use the DISTINCT keyword, as shown in the following statement:

USE NORTHWIND

SELECT DISTINCT Country FROM Customers

LIKE Operator

The LIKE operator uses pattern-matching characters, like the ones you use to select multiple files in DOS. The LIKE operator recognizes several pattern-matching characters (or *wildcard* characters) to match one or more characters, numeric digits, ranges of letters, and so on;

SQL Wildcard Characters

Wildcard

Character	**Description**
%	Matches any number of characters. The pattern program% will find program, programming, programmer, and so on. The pattern %program% will locate strings that contain the words *program, programming, nonprogrammer,* and so on.
_ (Underscore character)	Matches any single alphabetic character. The pattern b_y will find *boy* and *bay,* but not *boysenberry*
[ ]	Matches any single character within the brackets. The pattern Santa [YI] nez will find both *Santa Ynez* and *Santa Inez*
[^]	Matches any character not in the brackets. The pattern %q[^u]% will find words that contain the character *q* not followed by *u* (they are misspelled words).
[-]	Matches any one of a range of characters. The characters must be consecutive in the alphabet and specified in ascending order (A to Z, not Z to A). The pattern [a-c]% will find all words that begin with *a, b,* or *c* (in lowercase or uppercase).
#	Matches any single numeric character. The pattern D1## will find *D100* and *D139,* but not *D1000* or *D10*

You can use the LIKE operator to retrieve all titles about Windows from the Pubs database, with a statement like the following one.

```
USE PUBS
SELECT titles.title FROM titles WHERE titles.title LIKE '%WINDOWS%'
```

Null Values

A very common operation in manipulating and maintaining databases is to locate Null values in fields. The expressions IS NULL and IS NOT NULL find field values that are (or are not) Null. A zero-length string is not the same as a Null field. To locate the rows which have a Null value in their CompanyName column, use the following WHERE clause:

```
WHERE CompanyName IS NULL
```

ORDER Keyword

The rows of a query are not in any particular order. To request that the rows be returned in a specific
Order, use the ORDER BY clause, whose syntax is

```
ORDER BY col1, col2…
```

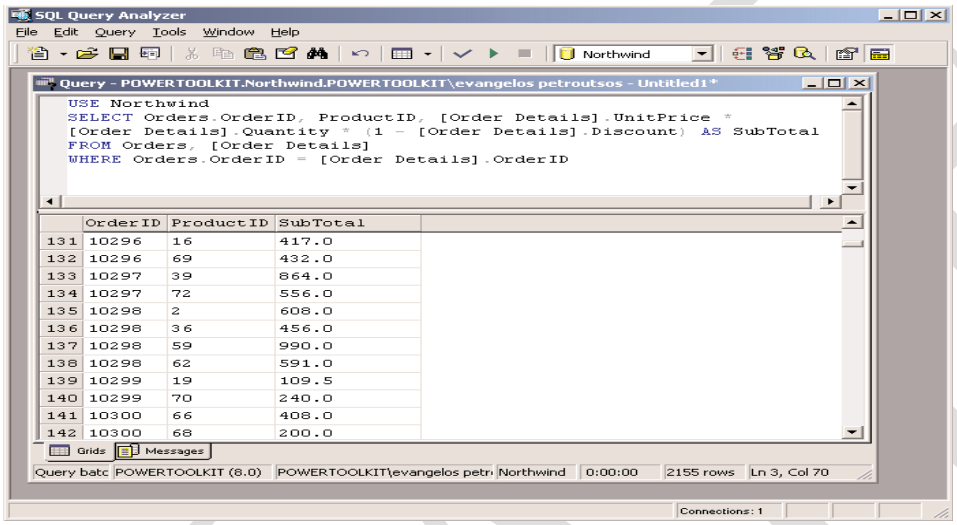
You can specify any number of columns in the ORDER list. The output of the query is ordered according to the values of the first column (col1). If two rows have identical values in this column, then they are sorted according to the second column, and so on. The statement

```
USE NORTHWIND
SELECT Company Name, Contact Name FROM Customers
ORDER BY Country, City
```

Calculated Fields

In addition to column names, you can specify calculated columns in the SELECT statement. as shown in the following statement. The Order Details table contains a row for each invoice line. Invoice #10248, for instance, contains four lines (four items sold), and each detail line appears in a separate row in the Order Details table. Each row holds the number of items sold, the item's price, and the corresponding discount. To display the line's subtotal, you must multiply the quantity by the price minus the discount, as shown in the following statement:

```
USE NORTHWIND
SELECT Orders.OrderID, ProductID,
[Order Details].UnitPrice * [Order Details].Quantity *
(1 - [Order Details].Discount) AS SubTotal
FROM Orders, [Order Details]
WHERE Orders.OrderID = [Order Details].OrderID
```



Totaling and Counting

SQL supports some aggregate functions, which act on selected fields of all the rows returned by the query. The aggregate functions, listed in Table 20.2, perform basic calculations like summing, counting, and averaging numeric values. Aggregate functions accept field names as arguments, and they return a single value, which is the sum of all values.

SQL's Aggregate Functions

Function	Returns
COUNT()	The number (count) of values in a specified column
SUM()	The sum of values in a specified column
AVG()	The average of the values in a specified column
MIN()	The smallest value in a specified column
MAX()	The largest value in a specified column

SQL Joins

Joins specify how you connect multiple tables in a query, and there are four types of joins:

- Left outer, or left join
- Right outer, or right join
- Full outer, or full join
- Inner join

A *join* operation combines all the rows of one table with the rows of another table. Joins are usually followed by a condition, which determines which records in either side of the join will appear in the result.

Left Joins

This join displays all the records in the left table and only those records of the table on the right that match certain user-supplied criteria.

This join has the following syntax:

FROM (primary table) LEFT JOIN (secondary table) ON (primary table).(field) (comparison) (secondary table).(field).

The following statement will retrieve all the titles from the Pubs database along with their publisher. If some titles have no publisher, they will be included to the result:

```
USE PUBS

SELECT title, pub_name
FROM titles LEFT JOIN publishers
ON titles.pub_id = publishers.pub_id
```

Right Joins

This join is similar to the left outer join, except that all rows in the table on the right are displayed and only the matching rows from the left table are displayed.
This join has the following syntax:
FROM (secondary table) RIGHT JOIN (primary table) ON (secondary table).(field) (comparison) (primary table).(field)

This join has the following example:

```
USE PUBS
SELECT title, pub_name
FROM titles RIGHT JOIN publishers
ON titles.pub_id = publishers.pub_id
```

Full Joins

The full join returns all the rows of the two tables, regardless of whether there are matching rows or not. In effect, it's a combination of left and right joins. To retrieve all the titles and all publishers, and match publishers to their titles, use the following join:

```
USE PUBS
SELECT title, pub_name
FROM titles FULL JOIN publishers
ON titles.pub_id = publishers.pub_id
```

Inner Joins

This join returns the matching rows of both tables, similar to the WHERE clause, and has the following syntax:

FROM (primary table) INNER JOIN (secondary table) ON (primary table).(field) (comparison) (secondary table).(field)

the following example:

```
USE PUBS
SELECT titles.title, publishers.pub_name FROM titles, publishers
WHERE titles.pub_id = publishers.pub_id
```

Grouping Rows

Sometimes you need to group the results of a query, so that you can calculate subtotals. Let's say you need not only the total revenues generated by a single product, but a list of all products and the revenues they generated. The example of the previous section "Totaling and Counting" calculates the total revenue generated by a single product. If you omit the WHERE clause, it will calculate the total revenue generated by all products. It is possible to use the SUM() function to break the calculations at each new product ID as demonstrated in the following statement. To do so, you must group the product IDs together with the GROUP BY clause.

```
USE NORTHWIND
SELECT ProductID,
SUM(Quantity * UnitPrice *(1 - Discount)) AS [Total Revenues]
FROM [Order Details]
GROUP BY ProductID
ORDER BY ProductID
```

Limiting Groups with HAVING

The HAVING clause limits the groups that will appear in the cursor. In a way, it is similar to the WHERE clause, but the HAVING clause allows you to use aggregate functions. The following statement will return the IDs of the products whose sales exceed 1,000 units:

```
USE NORTHWIND
SELECT ProductID, SUM(Quantity)
FROM [Order Details]
GROUP BY ProductID
HAVING SUM(Quantity) > 1000
```

the Order Details table with a WHERE clause:

```
USE                 NORTHWIND
SELECT        Products.ProductName,
              [Order Details].ProductID,
              SUM(Quantity) AS [Items Sold]
FROM           Products, [Order Details]
WHERE         [Order Details].ProductID = Products.ProductID
GROUP BY     [Order Details].ProductID, Products.ProductName
HAVING         SUM(Quantity) > 1000
ORDER BY      Products.ProductName
```

IN and NOT IN Keywords

The IN and NOT IN keywords are used in a WHERE clause to specify a list of values that a column must match (or not match). They are more of a shorthand notation for multiple OR operators. The following is statement that retrieves the names of the customers in all German-speaking countries:

```
USE NORTHWIND
SELECT CompanyName
FROM Customers
WHERE Country IN ('Germany', 'Austria', 'Switzerland')
```

The BETWEEN Keyword

The BETWEEN keyword lets you specify a range of values and limit the selection to the rows that have a specific column in this range. The BETWEEN keyword is a shorthand notation for an expression like column >= minValue AND column <= maxValue.

The following statement example:

```
USE NORTHWIND
```

```
SELECT OrderID, OrderDate, CompanyName
FROM Orders, Customers
WHERE Orders.CustomerID = Customers.CustomerID AND
(OrderDate BETWEEN '1/1/1997' AND '1/1/1998')
```

Action Queries

These queries are called *action queries,* and they're quite simple compared to the selection queries. There are three types of actions you can perform against a database: insertions of new rows, deletions of existing rows, and updates (edits) of existing rows. For each type of action there's a SQL statement, appropriately named INSERT, DELETE, and UPDATE. Their syntax is very simple, and the only complication is how you specify the affected rows (for deletions and updates).

They return the number of rows affected, but you disable this feature by calling the statement:
SET NOCOUNT ON

Deleting Rows

The DELETE statement deletes one or more rows from a table, and its syntax is:
DELETE table_name WHERE criteria

Inserting New Rows

The syntax of the INSERT statement is:
INSERT table_name (column_names) VALUES (values)

Example
INSERT Customers (CustomerID, CompanyName) VALUES ('FRYOG', 'Fruit & Yogurt')

Editing Existing Rows

The UPDATE statement edits a row's fields, and its syntax is
UPDATE table_name SET field1 = value1, field2 = value2, … WHERE criteria
Example
UPDATE Customers SET Country='United Kingdom'
WHERE Country = 'UK'

The Query Builder

The Query Builder is a visual tool for building SQL statements. It's a highly useful tool that generates SQL statements.

Views are based on SQL statements, and you will see the Query Builder with the statement that

implements the view you selected.
You can also create new queries by creating a new view. A view is the result of a query: it's a virtual table that consists of columns from one or more tables selected with a SQL SELECT statement.

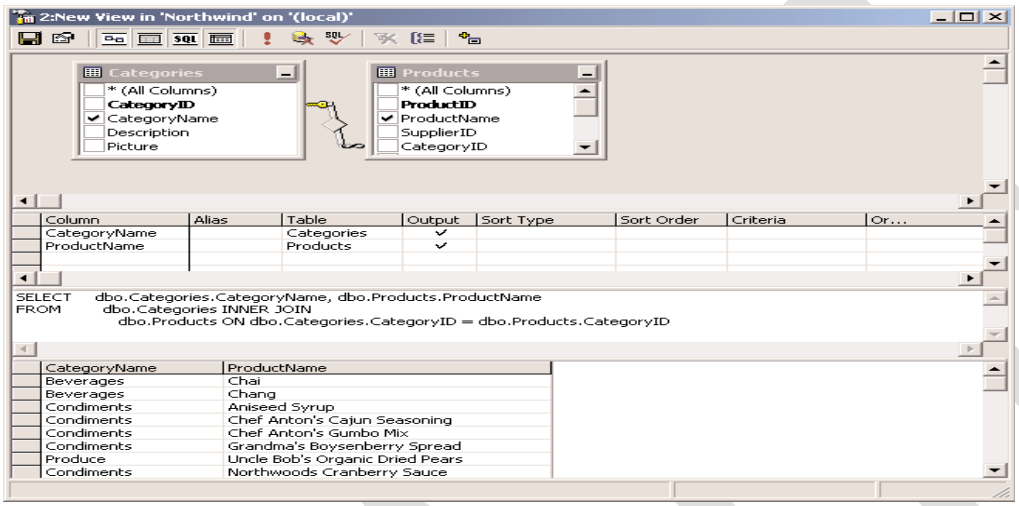


Diagram Pane

To select a table, right-click anywhere on the Diagram pane and you will see the Add Table dialog box. Select as many tables as you need and then close the Add Table dialog box. The little shape in the middle of the line indicates the type of join that must performed on the two tables, and it can take several shapes. To change the type of the relation, you can right-click the shape and select one of the options in the context menu. The diamond-shaped icon indicates an inner join, which requires that only rows with matching primary and foreign keys will be retrieved.By default, the Query Builder treats all joins as inner joins, but you can change the type of the join.

The first step in building a query is the selection of the fields that will be included in the result. Select the fields you want to include in your query by checking the box in front of their names, in the corresponding tables. As you select and deselect fields, their names appear in the Grid pane.

Grid Pane

The Grid pane contains the selected fields. Some fields may not be part of the output—you may use them only for selection purposes—but their names will appear on this pane. To exclude them from the output, clear the box in the Output column.
The Alias column contains a name for the field. By default, the column's name is the alias. This is the heading of each column in the output, and you can change the default name to any string that suits you.

SQL Pane
As you build the statement with point-and-click operations, the Query Builder generates the SQL statement that must be executed against the database to retrieve the specified data. The statement that retrieves product names along with their categories is shown next:

```
SELECT dbo.Products.ProductName, dbo.Categories.CategoryName
FROM dbo.Categories INNER JOIN dbo.Products
ON dbo.Categories.CategoryID = dbo.Products.CategoryID
```

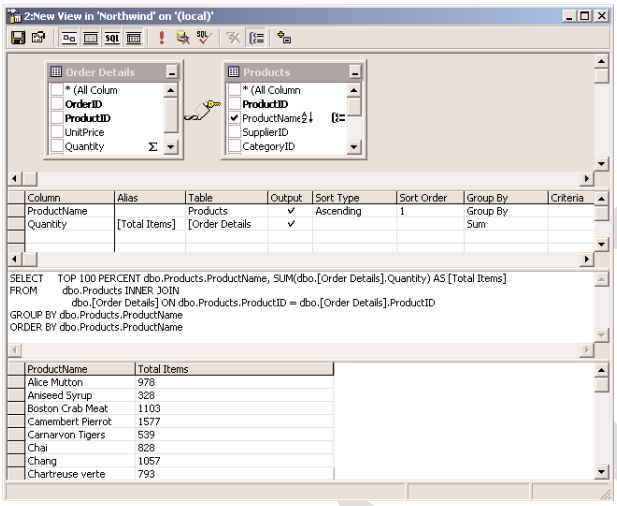
Results Pane
To execute a query, right-click somewhere on the SQL pane and select Run from the context menu. The Query Builder will execute the statement it generated and will display the results in the Results pane at the bottom of the window. The heading of each column is the column's name, unless you've specified an alias for the column.

SQL at Work: Calculating Sums
we'll build a query that retrieves all the products, along with the quantities sold. The names of the products will come from the Products table, while the quantities must be retrieved from the Order Details table. Because the same product appears in multiple rows of the tables (each product appears in multiple invoices with different quantities), we must sum the quantities of all rows that refer to the same product.

Create a new view in the Server Explorer to start the Query Builder, right-click the upper pane, and select Add Table. On the Add Table dialog box, select the tables Products and Order Details, then close the dialog box. The two tables will appear on the Diagram pane with a line connecting them. This is their relation. Now check the fields you want to include in the query: Select the field ProductName in the Products table and the field Quantity in the Order Details table. Expand the options in the Sort Type box in the ProductName row and select Ascending. The Query Builder will generate the following SQL statement:

```
SELECT dbo.Products.ProductName, dbo.[Order Details].Quantity
```

```
FROM dbo.Products INNER JOIN dbo.[Order Details]
ON dbo.Products.ProductID = dbo.[Order Details].ProductID
ORDER BY dbo.Products.ProductName
```



Stored Procedures

Stored procedures are short programs that are executed on the server and perform very specific tasks. Any action you perform against the database frequently should be coded as a stored procedure, so that you can call it from within any application or from different parts of the same application. A stored procedure that retrieves customers by name is a typical example, and you'll call this stored procedure from many different placed in your application.

Stored procedures isolate programmers from the database and minimize the risk of impairing the database's integrity. When all programmers access the same stored procedure to add a new invoice to the database. They simply call the stored procedure passing the invoice's fields as arguments.

Another advantage of using stored procedures is that they're compiled by SQL Server and they're executed faster. Stored procedures contain traditional programming statements to validate arguments, use default argument values, and so on. The language you use to write stored procedure is called T-SQL, and it's a superset of SQL.

```
CREATE PROCEDURE dbo.StoredProcedure1
/*
```

```
(
@parameter1 datatype = default value,
@parameter2 datatype OUTPUT
)
*/
AS
/* SET NOCOUNT ON */
RETURN
```

ADO .NET

Data access is making the application interact with a database, where all the data is stored. Different applications have different requirements for database access. VB .NET uses ADO .NET (Active X Data Object) as it's data access and manipulation protocol which also enables us to work with data on the Internet.

Evolution of ADO.NET

The first data access model, DAO (data access model) was created for local databases with the built-in Jet engine which had performance and functionality issues. Next came RDO (Remote Data Object) and ADO (Active Data Object) which were designed for Client Server architectures but soon ADO took over RDO. ADO was a good architecture but as the language changes so is the technology. With ADO, all the data is contained in a recordset object which had problems when implemented on the network and penetrating firewalls.

ADO was a connected data access, which means that when a connection to the database is established the connection remains open until the application is closed. Leaving the connection open for the lifetime of the application raises concerns about database security and network traffic. Also, as databases are becoming increasingly important and as they are serving more people, a connected data access model makes us think about its productivity.

Example : an application with connected data access may do well when connected to two clients, the same may do poorly when connected to 10 and might be unusable when connected to 100 or more. Also, open database connections use system resources to a maximum extent making the system performance less effective.

ADO.NET

When an application interacts with the database, the connection is opened to serve the request of the application and is closed as soon as the request is completed. Likewise, if a database is Updated, the connection is opened long enough to complete the Update operation and is closed.

By keeping connections open for only a minimum period of time, ADO .NET conserves system resources and provides maximum security for databases and also has less impact on system performance. Also, ADO .NET when interacting with the database uses XML and converts all the data into XML format for database related operations making them more efficient.

The ADO.NET Data Architecture

Data Access in ADO.NET relies on two components: DataSet and Data Provider.

DataSet

The dataset is a disconnected, in-memory representation of data. It can be considered as a local copy of the relevant portions of the database. The DataSet is persisted in memory and the data in it can be manipulated and updated independent of the database. When the use of this DataSet is finished, changes can be made back to the central database for updating. The data in DataSet can be loaded from any valid data source like Microsoft SQL server database, an Oracle database or from a Microsoft Access database.

Data Provider

The Data Provider is responsible for providing and maintaining the connection to the database. A DataProvider is a set of related components that work together to provide data in an efficient and performance driven manner. The .NET Framework currently comes with two DataProviders: the SQL Data Provider which is designed only to work with Microsoft's SQL Server 7.0 or later and the OleDb DataProvider which allows us to connect to other types of databases like Access and Oracle. Each DataProvider consists of the following component classes:

1. The Connection object which provides a connection to the database
2. The Command object which is used to execute a command
3. The DataReader object which provides a forward-only, read only, connected recordset
4. The DataAdapter object which populates a disconnected DataSet with data and performs update

Data access with ADO.NET can be summarized as follows:

1. A connection object establishes the connection for the application with the database.
2. The command object provides direct execution of the command to the database. If the command returns more than a single value, the command object returns a DataReader to provide the data. Alternatively, the DataAdapter can be used to fill the Dataset object. The database can be updated using the command object or the DataAdapter.

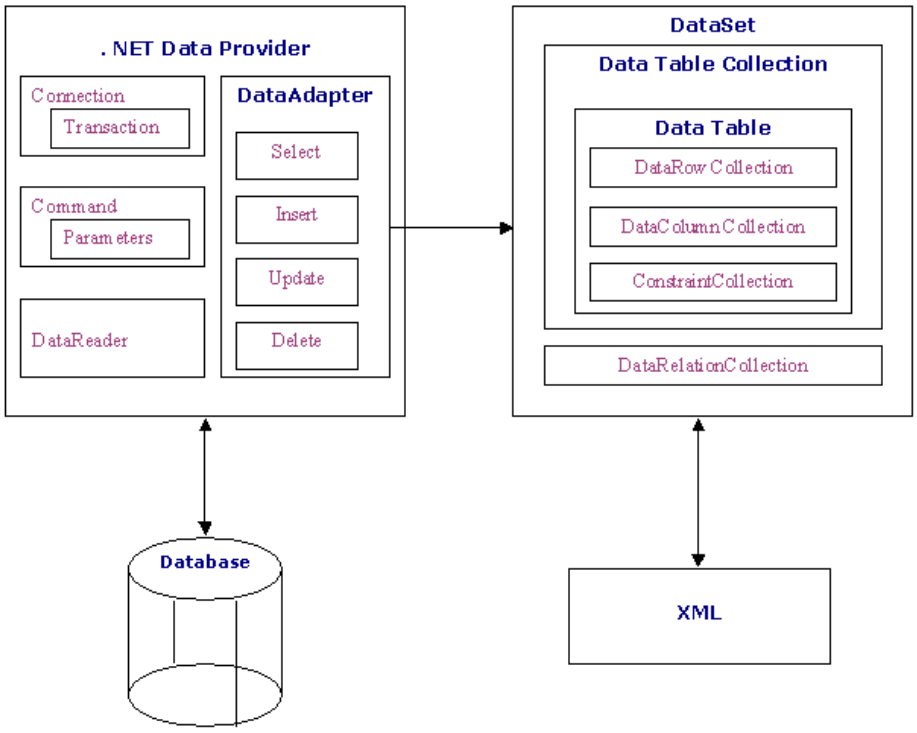
Component classes that make up the Data Providers

The Connection Object

The Connection object creates the connection to the database. Microsoft Visual Studio .NET provides two types of Connection classes: the SqlConnection object, which is designed specifically to connect to Microsoft SQL Server 7.0 or later, and the OleDbConnection object, which can provide connections to a wide range of database types like Microsoft Access and Oracle. The Connection object contains all of the information required to open a connection to the database.

The Command Object

The Command object is represented by two corresponding classes: SqlCommand and OleDbCommand. Command objects are used to execute commands to a database across a data connection. The Command objects can be used to execute stored procedures on the database, SQL commands, or return complete tables directly. Command objects provide three methods that are used to execute commands on the database:



ADO .NET Data Architecture

1. ExecuteNonQuery: Executes commands that have no return values such as INSERT, UPDATE or DELETE
2. ExecuteScalar: Returns a single value from a database query
3.
4. ExecuteReader: Returns a result set by way of a DataReader object

The DataReader Object

The DataReader object provides a forward-only, read-only, connected stream recordset from a database. Unlike other components of the Data Provider, DataReader objects cannot be directly instantiated. Rather, the DataReader is returned as the result of the Command object's ExecuteReader method. The SqlCommand.ExecuteReader method returns a SqlDataReader object, and the OleDbCommand.ExecuteReader method returns an OleDbDataReader object.

The DataReader can provide rows of data directly to application logic when you do not need to keep the data cached in memory. Because only one row is in memory at a time, the DataReader provides the lowest overhead in terms of system performance but requires the exclusive use of an open Connection object for the lifetime of the DataReader.

The DataAdapter Object

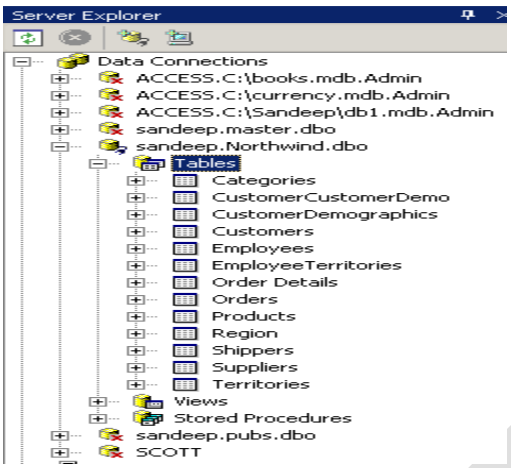
The DataAdapter is the class at the core of ADO .NET's disconnected data access. It is essentially the middleman facilitating all communication between the database and a DataSet. The DataAdapter is used either to fill a DataTable or DataSet with data from the database with it's Fill method. After the memory-resident data has been manipulated, the DataAdapter can commit the changes to the database by calling the Update method. The DataAdapter provides four properties that represent database commands:

1. SelectCommand
2. InsertCommand
3. DeleteCommand
4. UpdateCommand

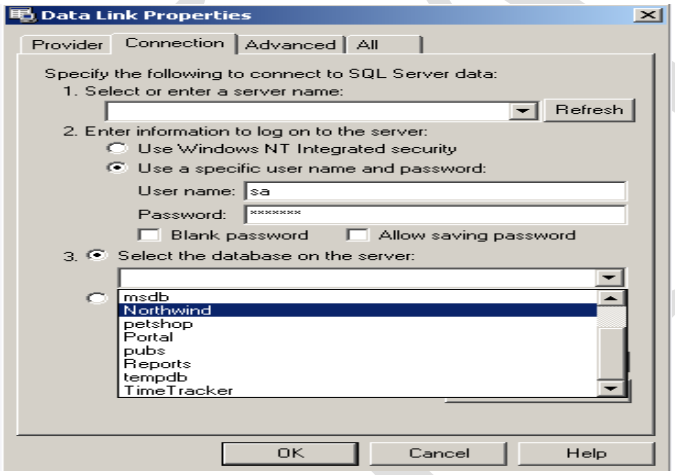
When the Update method is called, changes in the DataSet are copied back to the database and the appropriate InsertCommand, DeleteCommand, or UpdateCommand is executed.

Data Access with Server Explorer

Visual Basic allows us to work with databases in two ways, visually and code. In Visual Basic, Server Explorer allows us to work with connections across different data sources visually. Lets see how we can do that with Server Explorer. Server Explorer can be viewed by selecting View->Server Explorer from the main menu or by pressing Ctrl+Alt+S on the keyboard. The window that is displayed is the Server Explorer which lets us create and examine data connections. The Image below displays the Server Explorer.

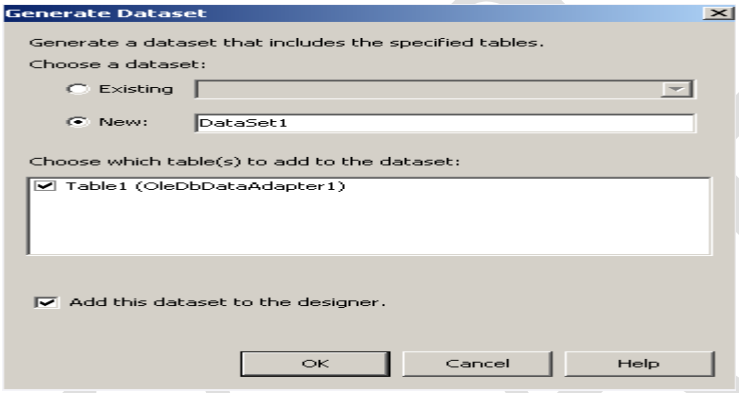


First, we need to establish a connection to this database. To do that, right-click on the Data Connections icon in Server Explorer and select Add Connection item. Doing that opens the Data Link Properties dialog which allows you to enter the name of the server you want to work along with login name and password. The Data Link properties window can be viewed in the Image below.



Since we are working with a database already on the server, select the option "select the database on the server". Selecting that lists the available databases on the server, select Northwind database from the list. Once you finish selecting the database, click on the Test Connection tab to test the connection. If the connection is successful, the message "Test Connection Succeeded" is

displayed. When connection to the database is set, click OK and close the Data Link Properties. Closing the data link properties adds a new Northwind database connection to the Server Explorer and this connection which we created just now is part of the whole Visual Basic environment which can be accessed even when working with other applications. When you expand the connection node ("+" sign), it displays the Tables, Views and Stored Procedures in that Northwind sample database. Expanding the Tables node will display all the tables available in the database. In this example we will work with Customers table to display its data.
Now drag Customers table onto the form from the Server Explorer. Doing that creates SQLConnection1 and SQLDataAdapter1 objects which are the data connection and data adapter objects used to work with data. They are displayed on the component tray. Now we need to generate the dataset that holds data from the data adapter. To do that select Data->Generate DataSet from the main menu or right-click SQLDataAdapter1 object and select generate DataSet menu.



Once the dialogbox is displayed, select the radio button with New option to create a new dataset. Make sure Customers table is checked and click OK. Clicking OK adds a dataset, DataSet11 to the component tray and that's the dataset with which we will work. Now, drag a DataGrid from toolbox. We will display Customers table in this data grid. Set the data grid's DataSource property to DataSet11 and it's DataMember property to Customers. Next, we need to fill the dataset with data from the data adapter. The following code does that:

```
Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As System.EventArgs)_
Handles MyBase.Load
DataSet11.Clear()
SqlDataAdapter1.Fill(DataSet11)
'filling the dataset with the data adapter’s fill method
End Sub
```

Once the application is executed, Customers table is displayed in the data grid. That's one of the simplest ways of displaying data using the Server Explorer window.

Creating a DataSet

To create a new database application, start a new project as usual. When the project's form appears, open the Server Explorer and expand one of the databases. Select the Northwind database and expand its icon to see the objects of the database. In the Tables section, select the Customers table, drag it with the mouse, and drop it on the form. VB will add two new objects in the Components tray: *SqlConnection1* and *SqlDataAdapter1*.

The first object, *SqlConnection1,* is the application's connection to the database. This object contains all the information needed to connect to the database. If you look at its properties, you will see that its ConnectionString property is:

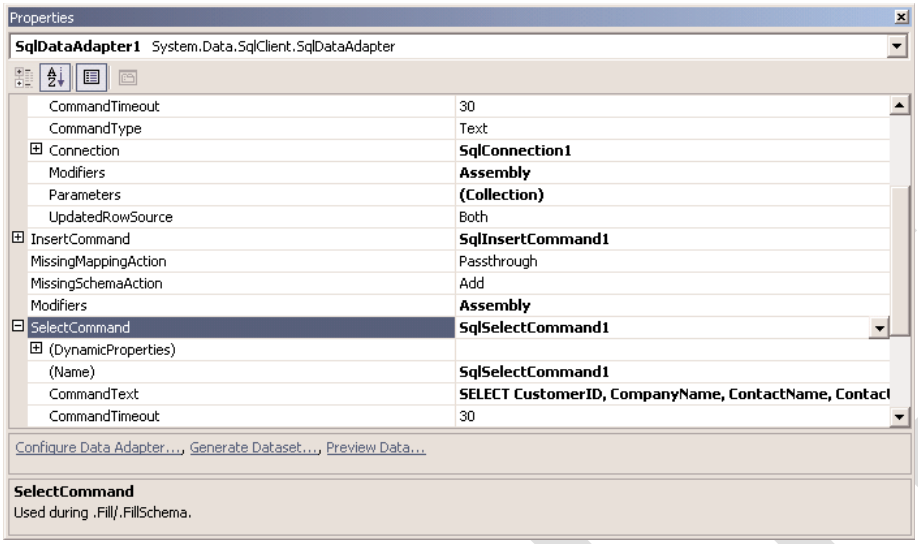
data source=PowerToolkit;initial catalog=Northwind;integrated security=SSPI;
persist security info=False;workstation id=POWERTOOLKIT;packet size=4096

SqlDataAdapter1 is the channel between your application and the database. The DataSet doesn't know anything about the database—it's not its job to know about the database. The application can request the data through the DataAdapter object, process them and then rely on the DataAdapter to update the database.

The SelectCommand object has a property called CommandText, which is a SELECT SQL statement:
SELECT CustomerID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, Country, Phone, Fax FROM dbo.Customers
This statement was generated automatically when you dropped the Customers table on the form. VB picked up the information from the table's structure in the database and create a SELECT statement to retrieve all the columns of all rows.

If you select the SelectCommand item in the Properties window and then click the button with the ellipsis that appears next to the item's setting, the Query Builder window will pop up and you can edit the SELECT statement. You can also edit the SELECT statement by selecting the s*qlDataAdapter1* object on the designer and clicking the Configure Data Adapter command at the bottom of the Properties window.

The *SqlDataAdapter1* also has an InsertCommand property, which is shown next:
INSERT INTO dbo.Customers(CustomerID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, Country, Phone, Fax)

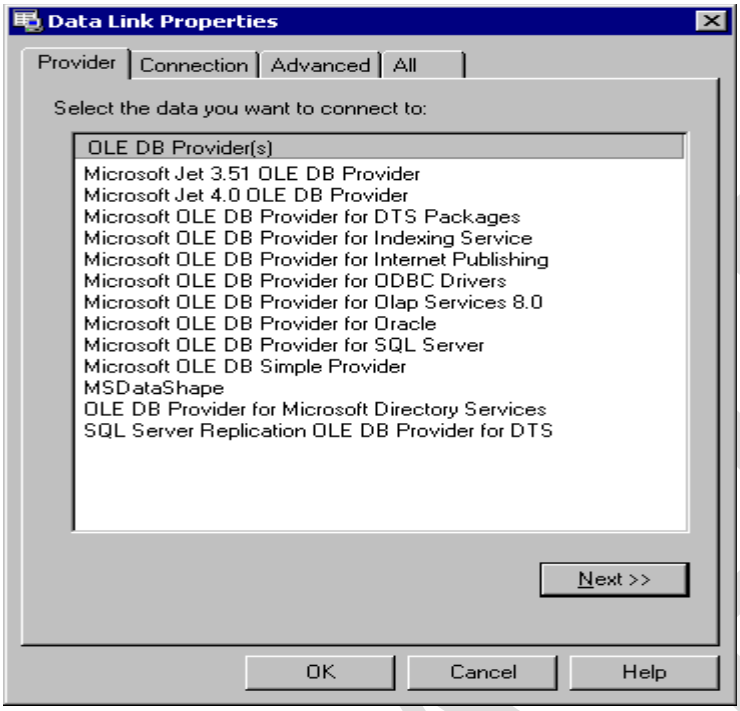


VALUES (@CustomerID, @CompanyName, @ContactName, @ContactTitle, @Address, @City, @Region, @PostalCode, @Country, @Phone, @Fax);
SELECT CustomerID, CompanyName, ContactName, ContactTitle, Address, City, Region, PostalCode, Country, Phone, Fax FROM dbo.Customers
WHERE (CustomerID = @Select_CustomerID)

Any string that starts with the @ symbol is a variable. The DataAdapter sets the values of all these variables to the values of the new row to be inserted and then executes the InsertCommand against the database. The INSERT statement will add a new row to the Customers table. The SELECT statement following the INSERT statement selects the newly added row from the table and returns it to the application. There are two more commands in the SqlDataAdapter object, UpdateCommand and DeleteCommand.

Microsoft Access and Oracle Database

The process is same when working with Oracle or MS Access but with some minor changes. When working with Oracle you need to select Microsoft OLE DB Provider for Oracle from the Provider tab in the DataLink dialog. You need to enter the appropriate Username and password. The Data Link Properties window can be viewed in the Image below.



When working with MS Access you need to select Microsoft Jet 4.0 OLE DB provider from the Provider tab in DataLink properties.
Using OleDb Provider

The Objects of the OleDb provider with which we work are:

1. The OleDbConnection Class : The OleDbConnection class represents a connection to OleDb data source. OleDb connections are used to connect to most databases.
2. The OleDbCommand Class: The OleDbCommand class represents a SQL statement or stored procedure that is executed in a database by an OLEDB provider.
3. The OleDbDataAdapter Class : The OleDbDataAdapter class acts as a middleman between the datasets and OleDb data source. We use the Select, Insert, Delete and Update commands for loading and updating the data.
4. The OleDbDataReader Class :The OleDbDataReader class creates a data reader for use with an OleDb data provider. It is used to read a row of data from the database. The data is read as forward-only stream which means that data is read sequentially, one row after another not allowing you to choose a row you want or going backwards.

The Command Objects

Each DataAdapter has four command objects, which provide the information needed to interact with the database: DeleteCommand, InsertCommand, UpdateCommand, and SelectCommand.

The Command and DataReader Objects

The DataReader is an object that lets you iterate through the rows retrieved by a query. It's faster than storing all the rows to a DataSet, but you can't move back and forth in the rows.

Once the Command object has been set up, you can execute it by calling one of the following methods:

ExecuteReader Executes the command and returns a DataReader object, which you can use to read the results, one row at a time.

ExecuteXMLReader Executes the command and returns a XMLDataReader object, which you can use to read the results, one row at a time.

ExecuteScalar Executes the command, returns the first column of the first row in the result, and ignores all other rows.

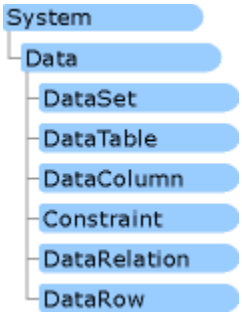
ExecuteNonQuery Executes a SQL command against the database and returns the number of rows affected. Use this method to execute a command that updates the database.

```
Private Sub Button1_Click(ByVal sender As System.Object,_
ByVal e As System.EventArgs) Handles Button1.Click
SqlConnection1.Open()
Dim SQLReader As System.Data.SqlClient.SqlDataReader
SQLReader = SqlCommand1.ExecuteReader()
While SQLReader.Read
ListBox1.Items.Add(SQLReader.Item("CategoryID") & vbTab & _
SQLReader.Item("CategoryName"))
End While
SqlConnection1.Close()
End Sub
```

Dataset

Datasets store data in a disconnected cache. The structure of a dataset is similar to that of a relational database; it exposes a hierarchical object model of tables, rows, and columns. In addition, it contains constraints and relationships defined for the dataset

Dataset Namespace



The fundamental parts of a dataset are exposed to you through standard programming constructs such as properties and collections. For example:

The DataSet class includes the Tables collection of data tables and the Relations collection of DataRelation objects.

The DataTable class includes the Rows collection of table rows, the Columns collection of data columns, and the ChildRelations and ParentRelations collections of data relations.

The DataRow class includes the RowState property, whose values indicate whether and how the row has been changed since the data table was first loaded from the database. Possible values for the RowState property include Deleted, Modified, New, and Unchanged

Transactions

A *transaction* is a series of actions that must either succeed, or fail, as a whole.

The following pseudo-code is the skeleton of a transaction:
Begin Transaction
Try
{ statements to complete transaction }
Commit Transaction
Catch Exception
Rollback Transaction
End Try

A DataRow's States
In addition to versions, rows have states, too; a row can be in one of the following states:

Added The row has been added to the DataTable, but it hasn't been accepted yet (rows are

accepted after they're written to the database as well).

Deleted The row has been deleted. However, it remains in the DataSet marked as Deleted, so that the Update method can delete the matching row of the underlying table.

Detached The row has been created but it has not been added to a DataTable yet. A row is in this state while you set its fields and before you actually add it to a table.

Modified The row has been modified, but it hasn't been accepted yet.

Unchanged The row hasn't been changed yet.

```
Public Function UpdateDataSource(ByVal dataSet As EditProducts.DSProducts) _
As System.Int32
Me.OleDbConnection1.Open()
Dim UpdatedRows As System.Data.DataSet
Dim InsertedRows As System.Data.DataSet
Dim DeletedRows As System.Data.DataSet
Dim AffectedRows As Integer = 0
UpdatedRows = DataSet.GetChanges(System.Data.DataRowState.Modified)
InsertedRows = DataSet.GetChanges(System.Data.DataRowState.Added)
DeletedRows = DataSet.GetChanges(System.Data.DataRowState.Deleted)
Try
If (Not (UpdatedRows) Is Nothing) Then
AffectedRows = OleDbDataAdapter1.Update(UpdatedRows)
AffectedRows = (AffectedRows + OleDbDataAdapter2.Update(UpdatedRows))
AffectedRows = (AffectedRows + OleDbDataAdapter3.Update(UpdatedRows))
End If
If (Not (InsertedRows) Is Nothing) Then
AffectedRows = (AffectedRows + OleDbDataAdapter1.Update(InsertedRows))
AffectedRows = (AffectedRows + OleDbDataAdapter2.Update(InsertedRows))
AffectedRows = (AffectedRows + OleDbDataAdapter3.Update(InsertedRows))
End If
If (Not (DeletedRows) Is Nothing) Then
AffectedRows = (AffectedRows + OleDbDataAdapter1.Update(DeletedRows))
AffectedRows = (AffectedRows + OleDbDataAdapter2.Update(DeletedRows))
AffectedRows = (AffectedRows + OleDbDataAdapter3.Update(DeletedRows))
End If
Catch updateException As System.Exception
Throw updateException
```

```
Finally
Me.OleDbConnection1.Close()
End Try
End Function
```

Coding

```
Public Class Form1
```

DECLERATION

```
   Dim con As ADODB.Connection
   Dim cmd As ADODB.Command
   Dim str, cnstr, sql As String
   Dim cn As OleDb.OleDbConnection
```

FORM LOAD

```
   Private Sub Form1_Load(ByVal sender As System.Object, ByVal e As
   System.EventArgs) Handles MyBase.Load
      con = New ADODB.Connection
      con.Open("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\new
      folder\employee.mdb")
   End Sub
```

ADDING A RECORD

```
   Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
   System.EventArgs) Handles Button1.Click
      str = "insert into table1 values('" + TextBox1.Text + "','" +
      TextBox2.Text + "','" + TextBox3.Text + "','" + TextBox4.Text +
      "','" + TextBox5.Text + "','" + TextBox6.Text + "','" +
      TextBox7.Text + "')"
      cmd = New ADODB.Command
      cmd.ActiveConnection = con
      cmd.CommandText = str
      cmd.Execute(MsgBox("Add"))
      cmd.Cancel()

   End Sub
```

DELETING A RECORD

```
   Private Sub Button2_Click(ByVal sender As System.Object, ByVal e As
   System.EventArgs) Handles Button2.Click
```

```
    str = "delete * from table1 where empno=" + ComboBox1.Text + ""
    cmd = New ADODB.Command
    cmd.ActiveConnection = con
    cmd.CommandText = str
    cmd.Execute()
    MsgBox("Delete")
    cmd.Cancel()
  End Sub
```

ADDING ITEMS IN COMBO BOX

```
Private Sub ComboBox1_GotFocus(ByVal sender As Object, ByVal e As
  System.EventArgs) Handles ComboBox1.GotFocus
    ComboBox1.Items.Clear()
    cnstr = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\new
    folder\employee.mdb"
    cn = New OleDb.OleDbConnection(cnstr)
    cn.Open()
    sql = "select empno from table1"
    Dim ocmd As New OleDb.OleDbCommand(sql, cn)
    Dim odatareader As OleDb.OleDbDataReader = ocmd.ExecuteReader
    While odatareader.Read
      ComboBox1.Items.Add(odatareader.GetValue(0).ToString())
    End While
    odatareader.Close()
    cn.Close()
  End Sub
```

SELECTING ITEMS IN COMBO BOX & DISPLAYING IN TEXT BOX

```
Private Sub ComboBox1_SelectedIndexChanged(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles ComboBox1.SelectedIndexChanged
cnstr = "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=c:\new folder\employee.mdb"
    cn = New OleDb.OleDbConnection(cnstr)
    cn.Open()
    sql = "select * from table1 where empno=" & CInt(ComboBox1.SelectedItem) & ""
    Dim ocmd As New OleDb.OleDbCommand(sql, cn)
    Dim odatareader As OleDb.OleDbDataReader = ocmd.ExecuteReader
    While odatareader.Read
      TextBox1.Text = odatareader.GetValue(0).ToString
```

```
            TextBox2.Text = odatareader.GetValue(1).ToString
            TextBox3.Text = odatareader.GetValue(2).ToString
            TextBox4.Text = odatareader.GetValue(3).ToString
            TextBox5.Text = odatareader.GetValue(4).ToString
            TextBox6.Text = odatareader.GetValue(5).ToString
            TextBox7.Text = odatareader.GetValue(6).ToString
        End While
        odatareader.Close()
        cn.Close()
    End Sub
```

UPDATING A RECORD

```
  Private Sub Button3_Click(ByVal sender As System.Object, ByVal e As
    System.EventArgs) Handles Button3.Click
        str = "delete * from table1 where empno=" + ComboBox1.Text + ""
        cmd = New ADODB.Command
        cmd.ActiveConnection = con
        cmd.CommandText = str
        cmd.Execute()
        cmd.Cancel()
            str = "insert into table1 values('" + TextBox1.Text + "','" + TextBox2.Text + "','" +
TextBox3.Text + "','" + TextBox4.Text + "','" + TextBox5.Text + "','" + TextBox6.Text + "','" +
TextBox7.Text + "')"
        cmd = New ADODB.Command
        cmd.ActiveConnection = con
        cmd.CommandText = str
        cmd.Execute()
        MsgBox("Update")
        cmd.Cancel()
    End Sub
End Class
```

Part B

1. List and explain various configuration sections of web.config file.
2. Give differences between connected and disconnected data access.
3. Explain important namespaces and classes of ADO .NET
4. Explain various data bound controls with syntax
5. Explain versioning and side by side execution with example.

Part C

6. Explain problem of DLL Hell and how to overcome from it.
7. Explain following keywords.
8. Namespace ,Class, Dim, Nothing, Module
9. Explain various string manipulation functions in .NET

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
Coimbatore-21

DEPARTMENT OF CS, CA & IT

UNIT-

III (Objective Type/ Multiple Choice Questions Each Question Carries One Mark)

.NET

PROGRAMMING (17CAP502)

S.No	Question	Option1	Option2	Option3	Option4	Answer
1	ADO Refers to _____	ActiveX Data object	Active Data Object	Application Development object	None	ActiveX Data object
2	Which of this is not a server component	Counter Component	Permission Checker Component	Distributed component	Content Linking	Distributed component
3	The Provider to access MS Access database is	OLEDB Data Provider	SQL Data Provider	ADO Data Provider	DOA Data Provider	OLEDB Data Provider
4	Which object is used to perform retrieve Operation	Connection Object	Command Object	Data object	Request Object	Command Object
5	_____ provides a language for describing Web Services	UDDI	WSDL	DDT	XML	WSDL

6	Drive,Folder,File Objects allbelong to _____	TextStream Object	FileSystem Object	Dictionary Object	NetworkSystem Object	FileSystem Object
7	Which of these properties belong to TextStream Object	Drive	Line	Path	Size	Line
8	CTS Refers to _____	Common type system	Common type service	Central type system	None	Common type system
9	What data type is the output of a Web Service?	Any data type we choose	Only numberic values	Text Strings	None of the Above	None of the Above
10	SQL Data Provider is used For	MS Access	SQL Server	Oracle	All the above	SQL Server
11	Which of the following operations can you NOT perform on an ADO.NET DataSet?	Development	A DataSet can be synchronised with a RecordSet	A DataSet can be converted to XML	You can infer the schema from a DataSet	A DataSet can be synchronised with a RecordSet
12	Which of this not a OLE DB Provider	ODBC drivers	DTP Packages OLAP services		MSDataShape	DTP Packages OLAP services
13	_____ Object is specifically designed to run commands against a data store	Connection	Command	Dataset Object	DataReader Object	Command

14	_____ Object allows to connect to the data stores	Connection	Command	Dataset Object	DataReader Object	Connection
15	_____ ADO Object is to handle data not formatted in structured rows and colums	Connection	Command	Dataset Object	Record	Record
16	Which of these is a Access Data Type	Text	String	Char	Long	Text
17	RDO stands for _____	Remote data object	remote development object	Remote data oriented	Real Data Object	Remote data object
18	Data set is a _____ architecture	connected	disconnected	self constructed	locally connected	disconnected
19	_____is responsible for providing and maintaining connection to database	Data reader	Data adapter	data set	Data provider	Data provider
20	DAO stands for _____	Data access object	Data adapter object	Data available object	Data provider oriented	Data access object
21	Local copy of database is called as _____	Data base copy	Dataset	Data provider	Data tables	Dataset

22	ADO NET comes with _____ providers	2	3	4	6	2
23	OLEDB Data Provider is used For	MS Access	SQL Server	Oracle	SQL Server and Oracle	a
24	_____ was a connected data access	RDO	ADO.NET	ASP.NET	DAO	ADO
25	In ADO.NET, The data are converted in to _____ Format	XML	XAML	HTML	CSS	XML
26	_____is a disconnected, in-memory representation of data.	Dataset	Data Reader	Data Adapter	Data Provider	Dataset
27	The _____object which provides a forward-only, read only, connected recordset	Dataset	Data Reader	Data Adapter	Data Provider	Data Reader
28	The _____ object which populates a disconnected DataSet with data and performs update	Dataset	Data Reader	Data Adapter	Data Provider	Data Adapter
29	A _____object establishes the connection for the application with the database.	Connection	Command	Dataset Object	Record	Connection
30	_____is the Extension for Access Database	.MDB	.RTF	.XML	.GCC	.MDB

31	_____Returns a single value from a database query	ExecuteNonQuery	ExecuteScalar	ExecuteReader	Non ExecuteReader	ExecuteScalar
32	_____Returns a result set by way of a DataReader object	ExecuteNonQuery	ExecuteScalar	ExecuteReader	Non ExecuteReader	ExecuteReader
33	_____ is essentially the middleman facilitating all communication between the database and a DataSet.	Dataset	Data Reader	Data Adapter	Data Provider	Data Adapter
34	Which Command is used to insert the New Record to the Database Table	Insert	Add	Update	New	Insert
35	_____ is a collection of Record	Database	Table	File	Document	Table
36	Which of the Component is not a Component of ADO.NET	DataReader	DataProvider	DataAdapter	DataChanger	DataChanger
37	Which of the Following does not support Client Server Technology	DAO	ADO	RDO	ADO.NET	DAO
38	Which object is used to perform Update Operation	Connection Object	Command Object	Data Adapter	Request Object	Data Adapter
39	In Access, the Image data type are stored using _____ data type	Text	BLOB	Memo	Number	BLOB
40	In SQL, the Image data type are stored in _____ format	Text	Unicode	Binary	Special	Binary

41	Which of these is not a Access Data Type	Text	String	memo	date	String

UNIT-IV

SYLLABUS

Goal of ASP.NET –ASP.NET Web Server Control-Validation Server Controls-Themes and Skins -Content Page Holder

ASP.NET

- ASP-Active server page
- Active Server Page program is a Server-Side Script (a program running on a Server) to create the widely used Microsoft operating system limit on Microsoft Windows by using the Web.
- Writing ASP.NET provides a family file ending with .aspx ASP.NET which language can be used to control database programming and many other languages are ASP.NET Microsoft has focused to develop a language.
- Generation of the world's next generation Internet-called new generation or that generation Internet Web2.0 a new generation to replace the older generation Internet (Web1.0).
- ASP is a language created by starting from the Active Server Page 3.0 (ASP 3.0) and develop a ASP.NET 1.0 / 1.1 / 2.0 / to ASP.NET 3.5, respectively, which is an optimization web development of applications.

PURPOSE OF ASP.NET:

- The main purpose of ASP.NET 3.5 is to allow users to create web applications that are secure, convenient and easy to reduce the number of write less code because version control is added.
- Accessing more functions.

Capabilities of ASP.NET:

ASP.NET can take the technology. NET Framework to be able to use apple pin with applications such as computer hardware any Palm PDA Notebook and mobile phones etc..

Make a web page developed. Technology in the form of ASP.NET Web Form is divided into 2 sections of the tag is used and the display processing program used. The screen is similar to that used tools. In

developing programs such as Visual Basic and C + +, etc.

Web Browser with all the different types of orders due. Defined in the Web Form, it is converted to HTML tag with an appropriate Web Browser, which differs from ASP in some of the traditional command does not work in certain Web Browser.

Support for working with the program. Developed from the language . NET such as VB.NET and C #. NET is.

The goal of ASP.NET 3.5

- Microsoft has targeted the development team of ASP.NET 3.5 with focus on what is useful to developers, administrators and management with the added performance.
- Code to eliminate the surplus in the previous version of ASP.NET. Make it easy to retrieve data such as shown in the table format control for ASP.NET 3.5 is called "**GridView Server Control**" .
- The GridView Server Control is capable of formatting the page [Paging] Sort of. order [Sorting] to correct this. To reduce the steps of writing code.
- One key goal of ASP.NET 3.5 is to speed up Web applications apple pin. One of ability is. The ability to cache [Cache] data with Microsoft SQL Server in ASP.NET 3.5 that includes critical characteristics called "**SQL Cache Invalidation**" helping to provide the information that a User would like to save the current time.
- This ASP.NET 3.5 to add support 64 - bit, which means users can apple Rapid Run ASP.NET applications on your Intel or AMD Processor Size 64 - bit and also compatible with ASP.NET 1.0,. 1.1 and 2.0 can be variations in apple Rapid applications of ASP.NET 1.0,1.1, and 2.0 to the. NET Framework 3.5.

Web Server Controls:

- Like HTML server controls, Web server controls are also created on the server and they require a runat="server" attribute to work.
- However, Web server controls do not necessarily map to any existing HTML elements and they may represent more complex elements.

- The syntax for creating a Web server control is:

```
<ASP:CONTROL_NAME ID="SOME_ID" RUNAT="SERVER" />
```

Web Server Control	Description
AdRotator	Displays a sequence of images
Button	Displays a push button
Calendar	Displays a calendar
CalendarDay	A day in a calendar control
CheckBox	Displays a check box
CheckBoxList	Creates a multi-selection check box group
DataGrid	Displays fields of a data source in a grid
DataList	Displays items from a data source by using templates
DropDownList	Creates a drop-down list
HyperLink	Creates a hyperlink
Image	Displays an image
ImageButton	Displays a clickable image
Label	Displays static content which is programmable (lets you apply styles to its content)
LinkButton	Creates a hyperlink button
ListBox	Creates a single- or multi-selection drop-down list
ListItem	Creates an item in a list
Literal	Displays static content which is programmable(does not let you apply styles to its content)
Panel	Provides a container for other controls
PlaceHolder	Reserves space for controls added by code
RadioButton	Creates a radio button
RadioButtonList	Creates a group of radio buttons
BulletedList	Creates a list in bullet format

Repeater	Displays a repeated list of items bound to the control
Style	Sets the style of controls
Table	Creates a table
TableCell	Creates a table cell
TableRow	Creates a table row
TextBox	Creates a text box
Xml	Displays an XML file or the results of an XSL transform

AdRotator Control

The AdRotator control is used to display a sequence of ad images.

This control uses an XML file to store the ad information. The XML file must begin and end with an <Advertisements> tag. Inside the <Advertisements> tag there may be several <Ad> tags which defines each ad.

The predefined elements inside the <Ad> tag are listed below:

Element	Description
<ImageUrl>	Optional. The path to the image file
<NavigateUrl>	Optional. The URL to link to if the user clicks the ad
<AlternateText>	Optional. An alternate text for the image
<Keyword>	Optional. A category for the ad
<Impressions>	Optional. The display rates in percent of the hits

The Button Control

The Button control is used to display a push button. The push button may be a submit button or a command button. By default, this control is a submit button.

A submit button does not have a command name and it posts the Web page back to the server when it is clicked. It is possible to write an event handler to control the actions performed when the submit button is clicked.

A command button has a command name and allows you to create multiple Button controls on a page. It is possible to write an event handler to control the actions performed when the command button is clicked.

Properties

Property	Description
CausesValidation	Specifies if a page is validated when a button is clicked
CommandArgument	Specifies additional information about the command to perform
CommandName	Specifies the command associated with the Command event
OnClientClick	Specifies the name of the function to be executed when a button is clicked

PostBackUrl	Specifies the URL of the page to post to from the current page when a button is clicked
runat	Specifies that the control is a server control. Must be set to "server"
Text	Specifies the text on a button
UseSubmitBehavior	Specifies whether or not a button uses the browser's submit mechanism or the ASP.NET postback mechanism
ValidationGroup	Specifies the group of controls a button causes validation, when it posts back to the server

The Calendar control

The Calendar control is used to display a calendar in the browser.

This control displays a one-month calendar that allows the user to select dates and move to the next and previous months.

SelectedDate	The selected date
SelectedDates	The selected dates
SelectedDayStyle	The style for selected days
SelectionMode	How a user is allowed to select dates
SelectMonthText	The text displayed for the month selection link
SelectorStyle	The style for the month and weeks selection links
SelectWeekText	The text displayed for the week selection link

CheckBox Control

The CheckBox control is used to display a check box.

Properties

Property	Description
AutoPostBack	Specifies whether the form should be posted immediately after the Checked property has changed or not. Default is false
CausesValidation	Specifies if a page is validated when a Button control is clicked
Checked	Specifies whether the check box is checked or not
InputAttributes	Attribute names and values used for the Input element for the CheckBox control
LabelAttributes	Attribute names and values used for the Label element for the CheckBox control
runat	Specifies that the control is a server control. Must be set to "server"
Text	The text next to the check box
TextAlign	On which side of the check box the text should appear (right or left)
ValidationGroup	Group of controls for which the Checkbox control causes validation when it posts back to the server
OnCheckedChanged	The name of the function to be executed when the Checked property has changed

CheckBoxList Control

The CheckBoxList control is used to create a multi-selection check box group.

Each selectable item in a CheckBoxList control is defined by a ListItem element!

Properties

Property	Description
CellPadding	The amount of pixels between the border and the contents of the table cell
CellSpacing	The amount of pixels between table cells
RepeatColumns	The number of columns to use when displaying the check box group
RepeatDirection	Specifies whether the check box group should be repeated horizontally or vertically
RepeatLayout	The layout of the check box group
runat	Specifies that the control is a server control. Must be set to "server"
TextAlign	**On which side of the check box the text should appear**

DropDownList Control

The DropDownList control is used to create a drop-down list.

Each selectable item in a DropDownList control is defined by a ListItem element!

Properties

Property	Description
SelectedIndex	The index of a selected item
OnSelectedIndexChanged	The name of the function to be executed when the index of the selected item has changed
runat	Specifies that the control is a server control. Must be set to "server"

ListBox Control

The ListBox control is used to create a single- or multi-selection drop-down list.

Each selectable item in a ListBox control is defined by a ListItem element!

Properties

Property	Description
Rows	The number of rows displayed in the list
SelectionMode	Allows single or multiple selections

RadioButton Control

The RadioButton control is used to display a radio button.

Properties

Property	Description
AutoPostBack	A Boolean value that specifies whether the form should be posted immediately after the Checked property has changed or not. Default is false
Checked	A Boolean value that specifies whether the radio button is checked or not
id	A unique id for the control
GroupName	The name of the group to which this radio button belongs
OnCheckedChanged	The name of the function to be executed when the Checked property has changed
runat	Specifies that the control is a server control. Must be set to "server"
Text	The text next to the radio button
TextAlign	On which side of the radio button the text should appear (right or left)

RadioButtonList Control

The RadioButtonList control is used to create a group of radio buttons.

Each selectable item in a RadioButtonList control is defined by a ListItem element.

Property	Description
CellPadding	The amount of pixels between the border and the contents of the table cell
CellSpacing	The amount of pixels between table cells

RepeatColumns	The number of columns to use when displaying the radio button group
RepeatDirection	Specifies whether the radio button group should be repeated horizontally or vertically
RepeatLayout	The layout of the radio button group
runat	Specifies that the control is a server control. Must be set to "server"
TextAlign	On which side of the radio button the text should appear

Table Control

The Table control is used in conjunction with the TableCell control and the TableRow control to create a table.

Property	Description
BackImageUrl	A URL to an image to use as an background for the table
Caption	The caption of the table
CaptionAlign	The alignment of the caption text
CellPadding	The space between the cell walls and contents
CellSpacing	The space between cells
GridLines	The gridline format in the table
HorizontalAlign	The horizontal alignment of the table in the page

Rows	A collection of rows in the Table
runat	Specifies that the control is a server control. Must be set to "server"

TextBox Control

The TextBox control is used to create a text box where the user can input text.

Property	Description
AutoCompleteType	Specifies the AutoComplete behavior of a TextBox
AutoPostBack	A Boolean value that specifies whether the control is automatically posted back to the server when the contents change or not. Default is false
CausesValidation	Specifies if a page is validated when a Postback occurs
Columns	The width of the textbox
MaxLength	The maximum number of characters allowed in the textbox
ReadOnly	Specifies whether or not the text in the text box can be changed
Rows	The height of the textbox (only used if TextMode="Multiline")
runat	Specifies that the control is a server control. Must be set to "server"
TagKey	
Text	The contents of the textbox

TextMode	Specifies the behavior mode of a TextBox control (SingleLine, MultiLine or Password)
ValidationGroup	The group of controls that is validated when a Postback occurs
Wrap	A Boolean value that indicates whether the contents of the textbox should wrap or not
OnTextChanged	The name of the function to be executed when the text in the textbox has changed

Validation Server Controls:

- A Validation server control is used to validate the data of an input control.
- If the data does not pass validation, it will display an error message to the user.

- The syntax for creating a Validation server control is:

```
<asp:control_name id="some_id" runat="server" />
```

Validation Server Control	Description
CompareValidator	Compares the value of one input control to the value of another input control or to a fixed value
CustomValidator	Allows you to write a method to handle the validation of the value entered
RangeValidator	Checks that the user enters a value that falls between two values
RegularExpressionValidator	Ensures that the value of an input control matches a specified pattern
RequiredFieldValidator	Makes an input control a required field
ValidationSummary	Displays a report of all validation errors occurred in a Web page

The RequiredFieldValidator:

The RequiredFieldValidator control ensures that the required field is not empty. It is generally tied to a text box to force input into the text box.

The syntax for the control:

```
<asp:RequiredFieldValidator ID="rfvcandidate"
      runat="server" ControlToValidate ="ddlcandidate"
      ErrorMessage="Please choose a candidate"
      InitialValue="Please choose a candidate">
</asp:RequiredFieldValidator>
```

The RangeValidator:

The RangeValidator control verifies that the input value falls within a predetermined range.

It has three specific properties:

Properties	Description
Type	it defines the type of the data; the available values are: Currency, Date, Double, Integer and String
MinimumValue	it specifies the minimum value of the range
MaximumValue	it specifies the maximum value of the range

The syntax for the control:

```
<asp:RangeValidator ID="rvclass"
     runat="server"
     ControlToValidate="txtclass"
     ErrorMessage="Enter your class (6 - 12)"
     MaximumValue="12"
     MinimumValue="6" Type="Integer">
</asp:RangeValidator>
```

The CompareValidator:

The CompareValidator control compares a value in one control with a fixed value, or, a value in another control.

It has the following specific properties:

Properties	Description
Type	it specifies the data type
ControlToCompare	it specifies the value of the input control to compare with
ValueToCompare	it specifies the constant value to compare with
Operator	it specifies the comparison operator, the available values are: Equal, NotEqual, GreaterThan, GreaterThanEqual, LessThan, LessThanEqual and DataTypeCheck

The basic syntax for the control:

```
<asp:CompareValidator ID="CompareValidator1"
    runat="server"
    ErrorMessage="CompareValidator">
</asp:CompareValidator>
```

The RegularExpressionValidator

The RegularExpressionValidator allows validating the input text by matching against a pattern against a regular expression. The regular expression is set in the ValidationExpression property.

The following table summarizes the commonly used syntax constructs for regular expressions:

Character Escapes	Description
\b	Matches a backspace
\t	Matches a tab
\r	Matches a carriage return
\v	Matches a vertical tab
\f	Matches a form feed
\n	Matches a new line
\	Escape character

Apart from single character match, a class of characters could be specified that can be matched, called the metacharacters.

Metacharacters	Description
.	Matches any character except \n
[abcd]	Matches any character in the set
[^abcd]	Excludes any character in the set
[2-7a-mA-M]	Matches any character specified in the range
\w	Matches any alphanumeric character and underscore
\W	Matches any non-word character
\s	Matches whitespace characters like, space, tab, new line etc.

\S	Matches any non-whitespace character
\d	Matches any decimal character
\D	Matches any non-decimal character

Quantifiers could be added to specify number of times a character could appear

Quantifier	**Description**
*	Zero or more matches
+	One or more matches
?	Zero or one matches
{N}	N matches
{N,}	N or more matches
{N,M}	Between N and M matches

The syntax for the control:

```
<asp:RegularExpressionValidator ID="string"
    runat="server"
    ErrorMessage="string"
    ValidationExpression="string"
    ValidationGroup="string">
</asp:RegularExpressionValidator>
```

The CustomValidator:

The CustomValidator control allows writing application specific custom validation routines for both the client side and the server side validation.

The client side validation is accomplished through the ClientValidationFunction property. The client side validation routine should be written in a scripting language, like JavaScript or VBScript, which the browser can understand.

The server side validation routine must be called from the control.s ServerValidate event handler. The server side validation routine should be written in any .Net language, like C# or VB.Net.

The basic syntax for the control

```
<asp:CustomValidator ID="CustomValidator1"
    runat="server"
    ClientValidationFunction=.cvf_func.
    ErrorMessage="CustomValidator">
</asp:CustomValidator>
```

The ValidationSummary Control

The ValidationSummary control does not perform any validation but shows a summary of all errors in the page. The summary displays the values of the ErrorMessage property of all validation controls that failed validation.

The following two mutually inclusive properties list out the error message:

- **ShowSummary:** shows the error messages in specified format.
- **ShowMessageBox:** shows the error messages in a separate window.

The syntax for the control:

<asp:ValidationSummary ID="ValidationSummary1"

```
    runat="server"
    DisplayMode = "BulletList"
    ShowSummary = "true"
    HeaderText="Errors:" />
```

Validation Groups:

Complex pages have different groups of information provided in different panels. In such a situation a need for performing validation separately for separate group, might arise. This kind of situation is handled using validation groups.

Themes and Skins:

- A theme is a collection of property settings that allow you to define the look of pages and controls, and then apply the look consistently across pages in a Web application, across an entire Web application, or across all Web applications on a server.
- Themes are made up of a set of elements: skins, cascading style sheets (CSS), images, and other resources.

- At a minimum, a theme will contain skins. Themes are defined in special directories in your Web site or on your Web server.

To apply a theme to a Web site

1. In the application's Web.config file, set the <pages> element to the name of the theme, either a global theme or a page theme, as shown in the following example:

```
<configuration>
    <system.web>
        <pages theme="ThemeName" />
    </system.web>
</configuration>
```

Skins:

- A skin file has the file name extension .skin and contains property settings for individual controls such as Button, Label, TextBox, or Calendar controls.
- Control skin settings are like the control markup itself, but contain only the properties you want to set as part of the theme.

- For example, the following is a control skin for a Button control:

  Copy

```
<asp:button runat="server" BackColor="lightblue" ForeColor="black" />
```

- You create .skin files in the Theme folder. A .skin file can contain one or more control skins for one or more control types. You can define skins in a separate file for each control or define all the skins for a theme in a single file.
- There are two types of control skins:

  1)default skins

  2)named skins

Default skins:

A default skin automatically applies to all controls of the same type when a theme is applied to a page.

- A control skin is a default skin if it does not have a SkinID attribute.
- For example, if you create a default skin for a Calendar control, the control skin applies to all Calendar controls on pages that use the theme. (Default skins are matched exactly by control type, so that a Button control skin applies to all Button controls, but not to LinkButton controls or to controls that derive from the Button object.)

Named skins:

- A named skin is a control skin with a SkinID property set.
- Named skins do not automatically apply to controls by type.
- Instead, you explicitly apply a named skin to a control by setting the control's SkinID property.
- Creating named skins allows you to set different skins for different instances of the same control in an application.

Applying Skins to Controls

Skins defined in your theme apply to all control instances in the application or pages to which the theme is applied. In some cases, you might want to apply a specific set of properties to an individual control. You can do that by creating a named skin (an entry in a .skin file that has a **SkinID** property set) and then applying it by ID to individual controls.

To apply a named skin to a control

- Set the control's SkinID property, as shown in the following example:
  <asp:Calendar runat="server" ID="DatePicker" SkinID="SmallCalendar" />

To disable themes for a page

- Set the **EnableTheming** attribute of the @ Page directive to **false**, as in this example:
- <%@ Page EnableTheming="false" %>

To disable themes for a control

- Set the **EnableTheming** property of the control to **false**, as in this example:
  <asp:Calendar id="Calendar1" runat="server" EnableTheming="false" />

To apply a style sheet theme programmatically

- In the page's code, override the StyleSheetTheme property and in the **get** accessor, return the name of the style sheet theme.
  The following code example shows how to set a theme named BlueTheme as the style sheet theme for a page:

```
Public Overrides Property StyleSheetTheme() As String
  Get
     Return "BlueTheme"
  End Get
  Set(ByVal value As String)
  End Set
End Property
```

To apply control skins programmatically

- In a handler for the page's PreInit method, set the control's **SkinID** property.
  The following code example shows how to set the SkinID property of a Calendar control. This example assumes that the the page's theme has already been set.

  VB

```
Sub Page_PreInit(ByVal sender As Object, _
      ByVal e As System.EventArgs) _
      Handles Me.PreInit
   Calendar1.SkinID = "CustomSkin"
End Sub
```

Cascading Style Sheets

- A theme can also include a cascading style sheet (.css file).
- When you put a .css file in the theme folder, the style sheet is applied automatically as part of the theme.

- You define a style sheet using the file name extension .css in the theme folder.

Theme Graphics and Other Resources

- Themes can also include graphics and other resources, such as script files or sound files. For example, part of your page theme might include a skin for a TreeView control.
- As part of the theme, you can include the graphics used to represent the expand button and the collapse button.

- Typically, the resource files for the theme are in the same folder as the skin files for that theme, but they can be elsewhere in the Web application, in a subfolder of the theme folder for example. To refer to a resource file in a subfolder of the theme folder, use a path like the one shown in this Image control skin:

```
<asp:Image runat="server" ImageUrl="ThemeSubfolder/filename.ext" />
```

- You can also store your resource files outside the theme folder.
- If you use the tilde (~) syntax to refer to the resource files, the Web application will automatically find the images. For example, if you place the resources for a theme in a

subfolder of your application, you can use paths of the form ~/SubFolder/filename.ext to refer to resource files, as in the following example.

<asp:Image runat="server" ImageUrl="~/AppSubfolder/filename.ext" />

Scoping Themes

- You can define themes for a single Web application, or as global themes that can be used by all applications on a Web server.
- After a theme is defined, it can be placed on individual pages using the Theme or StyleSheetTheme attribute of the @ Page directive, or it can be applied to all pages in an application by setting the <pages> element in the application configuration file.

- If the <pages> element is defined in the Machine.config file, the theme will apply to all pages in Web applications on the server.

Page Themes

- A page theme is a theme folder with control skins, style sheets, graphics files and other resources created as a subfolder of the \App_Themes folder in your Web site.
- Each theme is a different subfolder of the \App_Themes folder.

- The following example shows a typical page theme, defining two themes named BlueTheme and PinkTheme.

Global Themes

- A global theme is a theme that you can apply to all the Web sites on a server.
- Global themes allow you to define an overall look for your domain when you maintain multiple Web sites on the same server.

- Global themes are like page themes in that they include property settings, style sheet settings, and graphics.

- However, global themes are stored in a folder named Themes that is global to the Web server.

- Any Web site on the server, and any page in any Web site, can reference a global theme.

Theme Settings Precedence

- You can specify the precedence that theme settings take over local control settings by specifying how the theme is applied.
- If you set a page's Theme property, control settings in the theme and the page are merged to form the final settings for the control.
- If a control setting is defined in both the control and the theme, the control settings from the theme override any page settings on the control.
- This strategy enables the theme to create a consistent look across pages, even if controls on the pages already have individual property settings.
- For example, it allows you to apply a theme to a page you created in an earlier version of ASP.NET.
- Alternatively, you can apply a theme as a style sheet theme by setting the page's StyleSheetTheme property.
- In this case, local page settings take precedence over those defined in the theme when the setting is defined in both places.
- This is the model used by cascading style sheets. You might apply a theme as a style sheet theme if you want to be able to set the properties of individual controls on the page while still applying a theme for an overall look.
- Global theme elements cannot be partially replaced by elements of application-level themes.
- If you create an application-level theme with the same name as a global theme, theme elements in the application-level theme will not override the global theme elements.

Properties You Can Define Using Themes

- As a rule, you can use themes to define properties that concern a page or control's appearance or static content.
- You can set only those properties that have a ThemeableAttribute attribute set to true in the control class.

- Properties that explicitly specify control behavior rather than appearance do not accept theme values. For example, you cannot set a Button control's CommandName property by using a theme.

- Similarly, you cannot use a theme to set a GridView control's AllowPaging property or DataSource property.

Themes vs. Cascading Style Sheets

Themes are similar to cascading style sheets in that both themes and style sheets define a set of common attributes that can be applied to any page. However, themes differ from style sheets in the following ways:

- Themes can define many properties of a control or page, not just style properties. For example, using themes, you can specify the graphics for a TreeView control, the template layout of a GridView control, and so on.
- Themes can include graphics.

- Themes do not cascade the way style sheets do. By default, any property values defined in a theme referenced by a page's Theme property override the property values declaratively set on a control, unless you explicitly apply the theme using the StyleSheetTheme property. For more information, see the Theme Settings Precedence section above.

- Only one theme can be applied to each page. You cannot apply multiple themes to a page, unlike style sheets where multiple style sheets can be applied.

Security Considerations

Themes can cause security issues when they are used on your Web site. Malicious themes can be used to:

- Alter a control's behavior so that it does not behave as expected.
- Inject client-side script, therefore posing a cross-site scripting risk.

- Alter validation.

- Expose sensitive information.

- The mitigations for these common threats are:
- Protect the global and application theme directories with proper access control settings. Only trusted users should be allowed to write files to the theme directories.
- Do not use themes from an untrusted source. Always examine any themes from outside your organization for malicious code before using them on you Web site.
- Do not expose the theme name in query data. Malicious users could use this information to use themes that are unknown to the developer and thereby expose sensitive information.

Content Page Holder Control

- In the preceding tutorial we examined how master pages enable ASP.NET developers to create a consistent site-wide layout.
- Master pages define both markup that is common to all of its content pages and regions that are customizable on a page-by-page basis.
- In the previous tutorial we created a simple master page (Site.master) and two content pages (Default.aspx and About.aspx).
- Our master page consisted of two ContentPlaceHolders named head and MainContent, which were located in the <head> element and Web Form, respectively.
- While the content pages each had two Content controls, we only specified markup for the one corresponding to MainContent.
- As evidenced by the two ContentPlaceHolder controls in Site.master, a master page may contain multiple ContentPlaceHolders.
- What's more, the master page may specify default markup for the ContentPlaceHolder controls.
- A content page, then, can optionally specify its own markup or use the default markup.
- In this tutorial we look at using multiple content controls in the master page and see how to define default markup in the ContentPlaceHolder controls.

Step 1: Adding Additional ContentPlaceHolder Controls to the Master Page

- Many website designs contain several areas on the screen that are customized on a page-by-page basis. Site.master, the master page we created in the preceding tutorial, contains a single ContentPlaceHolder within the Web Form named MainContent. Specifically, this ContentPlaceHolder is located within the mainContent element.
- Figure 1 shows Default.aspx when viewed through a browser.

- The region circled in red is the page-specific markup corresponding to MainContent.

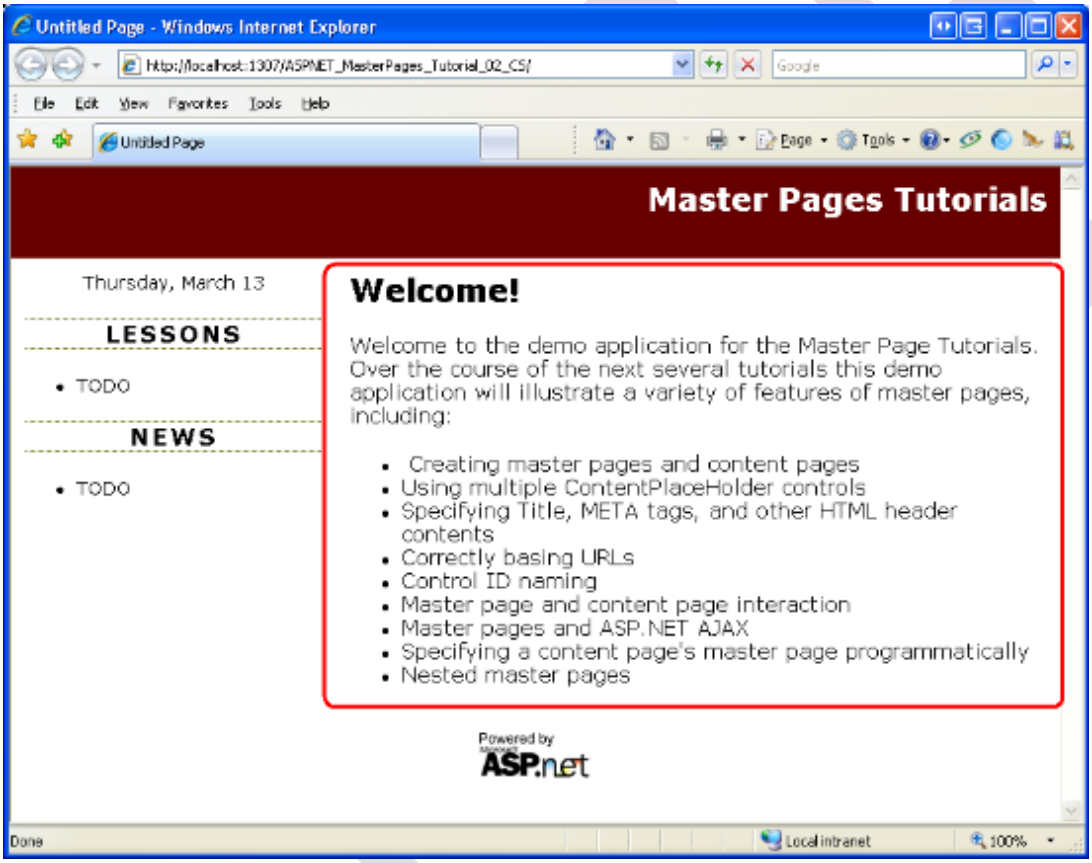


Figure 01: The Circled Region Shows the Area Currently Customizable on a Page-by-Page Basis (Click to view full-size image)

- Imagine that in addition to the region shown in Figure 1, we also need to add page-specific items to the left column beneath the Lessons and News sections.

- To accomplish this, we add another ContentPlaceHolder control to the master page.

- To follow along, open the Site.master master page in Visual Web Developer and then drag a ContentPlaceHolder control from the Toolbox onto the designer after the News section. Set the ContentPlaceHolder's ID to LeftColumnContent.

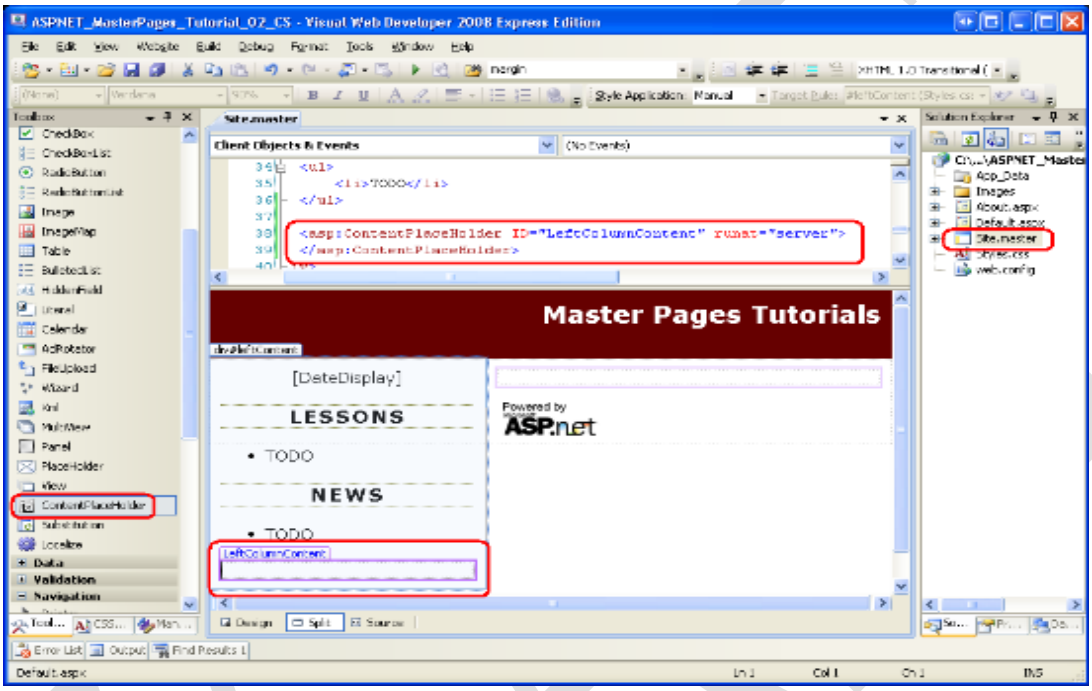


Figure 02: Add a ContentPlaceHolder Control to the Master Page's Left Column (Click to view full-size image)

- With the addition of the LeftColumnContent ContentPlaceHolder to the master page, we can define content for this region on a page-by-page basis by including a Content control in the page whose ContentPlaceHolderID is set to LeftColumnContent.
- We examine this process in Step 2.

Step 2: Defining Content for the New ContentPlaceHolder in the Content Pages

- When adding a new content page to the website, Visual Web Developer automatically creates a Content control in the page for each ContentPlaceHolder in the selected master page.

- Having added a the LeftColumnContent ContentPlaceHolder to our master page in Step 1, new ASP.NET pages will now have three Content controls.

- To illustrate this, add a new content page to the root directory named MultipleContentPlaceHolders.aspx that is bound to the Site.master master page.

Visual Web Developer creates this page with the following declarative markup:

```
<%@ Page Language="C#" MasterPageFile="/Site.master" AutoEventWireup="true"
CodeFile="MultipleContentPlaceHolders.aspx.cs" Inherits="MultipleContentPlaceHolders"
Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" Runat="Server">
</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="LeftColumnContent" Runat="Server">
</asp:Content>
```

- Enter some content into the Content control referencing the MainContent ContentPlaceHolders (Content2).

Next, add the following markup to the Content3 Content control (which references the LeftColumnContent ContentPlaceHolder):

```
<h3>Page-Specific Content</h3>
<ul>
 <li>This content is defined in the content page.</li>
 <li>The master page has two regions in the Web Form that are editable on a
 page-by-page basis.</li>
</ul>
```

- After adding this markup, visit the page through a browser.
- As Figure 3 shows, the markup placed in the Content3 Content control is displayed in the left column beneath the News section (circled in red).

- The markup placed in Content2 is displayed in the right portion of the page (circled in blue).

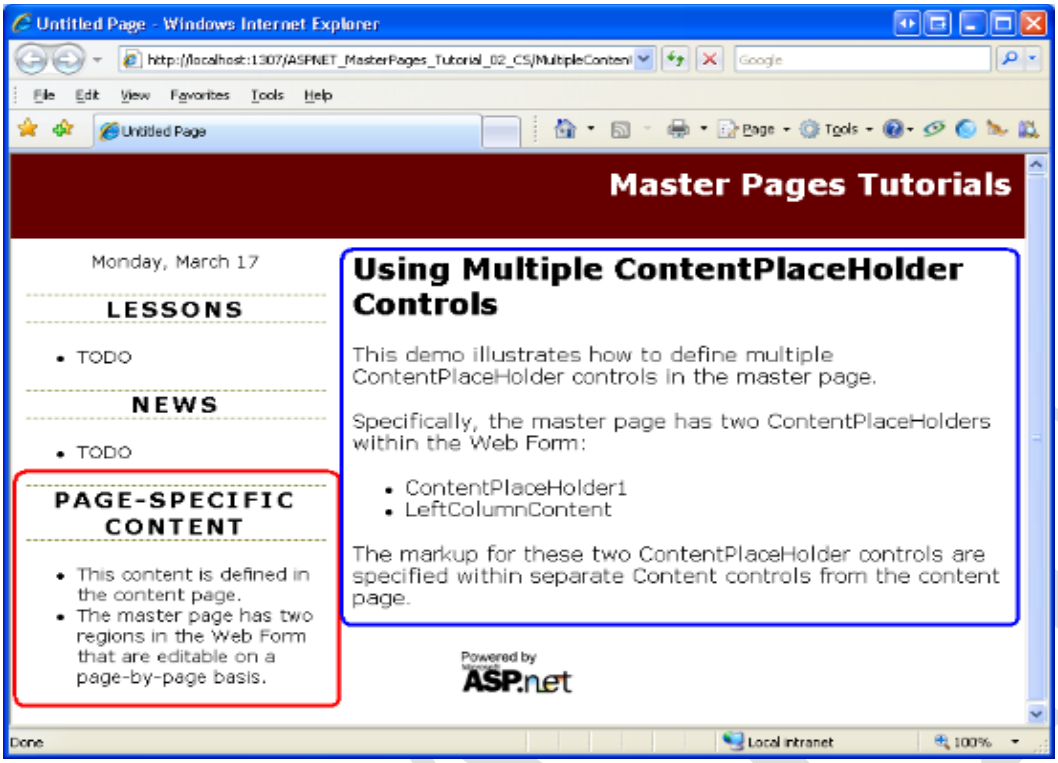


Figure 03: The Left Column Now Includes Page-Specific Content Beneath the News Section (Click to view full-size image)

Defining Content in Existing Content Pages

- Creating a new content page automatically incorporates the ContentPlaceHolder control we added in Step 1.
- But our two existing content pages - About.aspx and Default.aspx - don't have a Content control for the LeftColumnContent ContentPlaceHolder.

- To specify content for this ContentPlaceHolder in these two existing pages, we need to add a Content control ourselves.

- Unlike most ASP.NET Web controls, the Visual Web Developer Toolbox does not include a Content control item.

- We can manually type in the Content control's declarative markup into the Source view, but an easier and quicker approach is to use the Design view.

- Open the About.aspx page and switch to the Design view.

- As Figure 4 illustrates, the LeftColumnContent ContentPlaceHolder appears in the Design view; if you mouse over it, the title displayed reads: "LeftColumnContent (Master)." The inclusion of "Master" in the title indicates that there is no Content control defined in the page for this ContentPlaceHolder.

- If there exists a Content control for the ContentPlaceHolder, as in the case for MainContent, the title will read: "*ContentPlaceHolderID* (Custom)."

- To add a Content control for the LeftColumnContent ContentPlaceHolder to About.aspx, expand the ContentPlaceHolder's smart tag and click the Create Custom Content link.

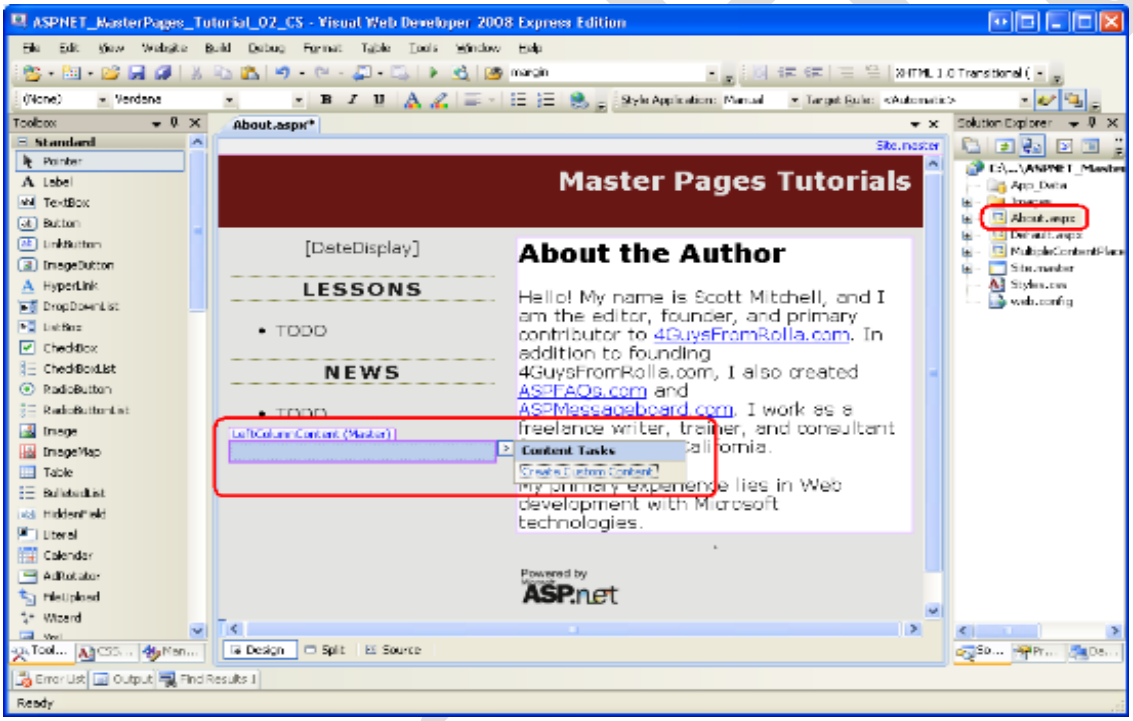


Figure 04: The Design View for About.aspx Shows the LeftColumnContent ContentPlaceHolder (Click to view full-size image)

- Clicking the Create Custom Content link generates the necessary Content control in the page and sets its ContentPlaceHolderID property to the ContentPlaceHolder's ID.

For example, clicking the Create Custom Content link for LeftColumnContent region in About.aspx adds the following declarative markup to the page:

<asp:Content ID="Content3" runat="server" contentplaceholderid="LeftColumnContent">

</asp:Content>

Omitting Content Controls

- ASP.NET does not require that all content pages include Content controls for each and every ContentPlaceHolder defined in the master page.
- If a Content control is omitted, the ASP.NET engine uses the markup defined within the ContentPlaceHolder in the master page.

- This markup is referred to as the ContentPlaceHolder's *default content* and is useful in scenarios where the content for some region is common among the majority of pages, but needs to be customized for a small number of pages.

- Step 3 explores specifying default content in the master page.

- Currently, Default.aspx contains two Content controls for the head and MainContent ContentPlaceHolders; it does not have a Content control for LeftColumnContent.

- Consequently, when Default.aspx is rendered the LeftColumnContent ContentPlaceHolder's default content is used.

- Because we have yet to define any default content for this ContentPlaceHolder, the net effect is that no markup is emitted for this region.

- To verify this behavior, visit Default.aspx through a browser.

- As Figure 5 shows, no markup is emitted in the left column beneath the News section.

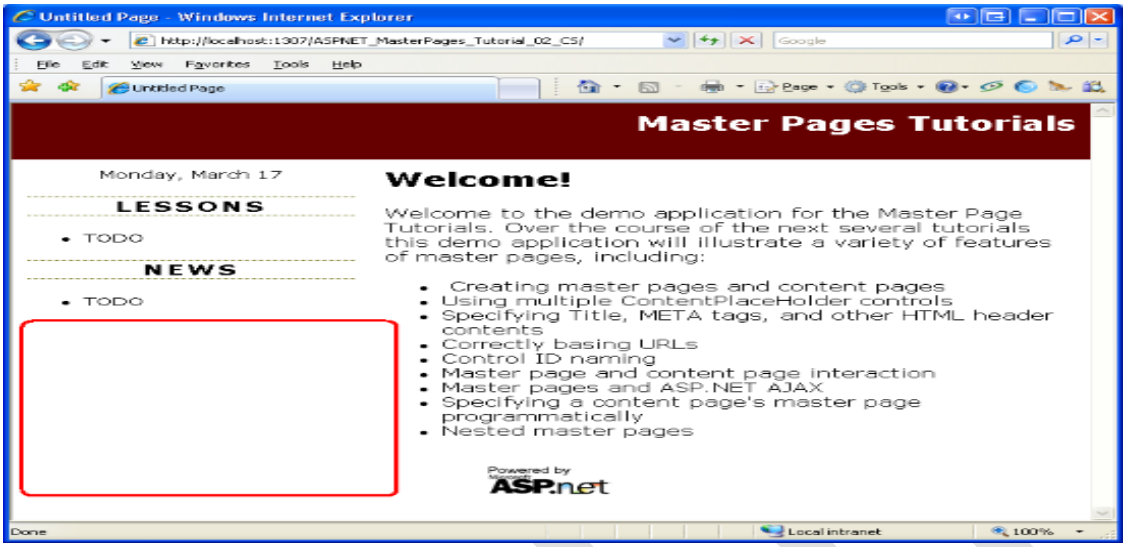


Figure 05: No Content is Rendered for the LeftColumnContent ContentPlaceHolder

Step 3: Specifying Default Content in the Master Page

- Some website designs include a region whose content is the same for all pages in the site except for one or two exceptions.
- Consider a website that supports user accounts.
- Such a site requires a login page where visitors can enter their credentials to sign into the site.
- To expedite the sign in process, the website designers might include username and password textboxes in the upper left corner of every page to allow users to sign in without having to explicitly visit the login page.
- While these username and password textboxes are helpful in most pages, they are redundant in the login page, which already contains textboxes for the user's credentials.
- To implement this design, you could create a ContentPlaceHolder control in the upper left corner of the master page.

- Each page that needed to display the username and password textboxes in their upper left corner would create a Content control for this ContentPlaceHolder and add the necessary interface.

- The login page, on the other hand, would either omit adding a Content control for this ContentPlaceHolder or would create a Content control with no markup defined.

- The downside of this approach is that we have to remember to add the username and password textboxes to every page we add to the site (except for the login page).

- This is asking for trouble.

- We're likely to forget to add these textboxes to a page or two or, worse, we might not implement the interface correctly (perhaps adding just one textbox instead of two).

- A better solution is to define the username and password textboxes as the ContentPlaceHolder's default content.

- By doing so, we only need to override this default content in those few pages that do not display the username and password textboxes (the login page, for instance).

- To illustrate specifying default content for a ContentPlaceHolder control, let's implement the scenario just discussed.

Adding a ContentPlaceHolder and Specifying Its Default Content

Open the Site.master master page and add the following markup to the left column between the DateDisplay Label and Lessons section:

```
<asp:ContentPlaceHolder ID="QuickLoginUI" runat="server">
 <asp:Login ID="QuickLogin" runat="server"
   TitleText="<h3>Sign In</h3>"
   FailureAction="RedirectToLoginPage">
 </asp:Login>
</asp:ContentPlaceHolder>
```

After adding this markup your master page's Design view should look similar to Figure 6.

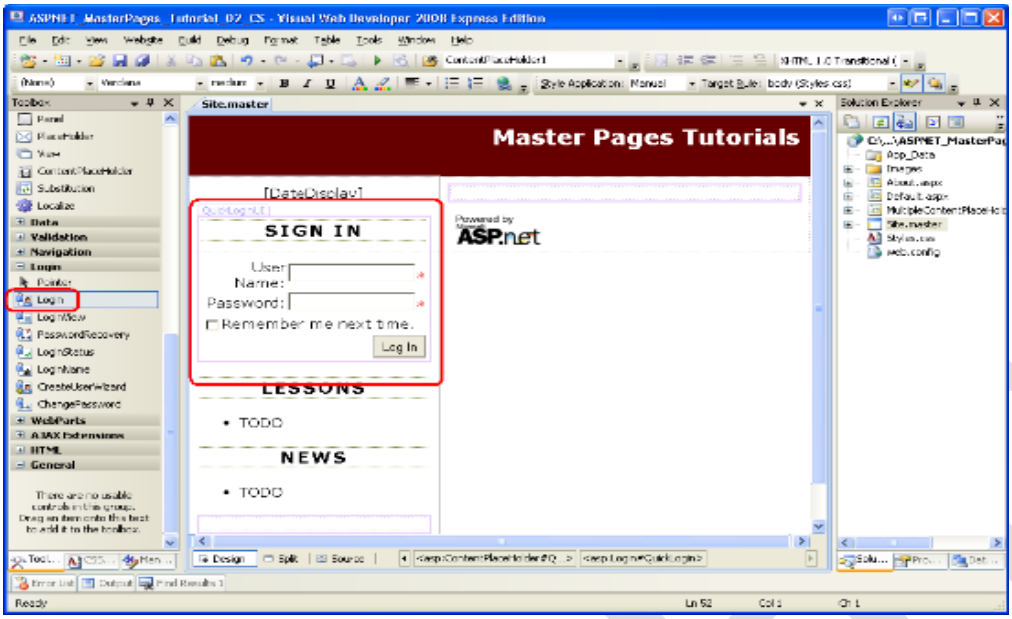


Figure 06: The Master Page Includes a Login Control (Click to view full-size image)

- This ContentPlaceHolder, QuickLoginUI, has a Login Web control as its default content. The Login control displays a user interface that prompts the user for their username and password along with a Log In button.
- Upon clicking the Log In button, the Login control internally validates the user's credentials against the Membership API.

- To use this Login control in practice, then, you need to configure your site to use Membership.

- This topic is beyond the scope of this tutorial; refer to my Forms Authentication, Authorization, User Accounts and Roles tutorials for more information on building a web application that supports user accounts.

- Feel free to customize the Login control's behavior or appearance.

- I have set two of its properties: TitleText and FailureAction.

- The TitleText property value, which defaults to "Log In", is displayed at the top of the control's user interface.

- I have set this property so that it displays the text "Sign In" as an <h3> element.
- The FailureAction property indicates what to do if the user's credentials are invalid.
- It defaults to a value of Refresh, which leaves the user on the same page and displays a failure message within the Login control.
- I've changed it to RedirectToLoginPage, which sends the user to the login page in the event of invalid credentials.
- I prefer to send the user to the login page when a user attempts to login from some other page, but fails, because the login page can contain additional instructions and options that would not easily fit into the left column.
- For example, the login page might include options to retrieve a forgotten password or to create a new account.

Creating the Login Page and Overriding the Default Content

- With the master page complete, our next step is to create the login page.
- Add an ASP.NET page to your site's root directory named Login.aspx, binding it to the Site.master master page.
- Doing so will create a page with four Content controls, one for each of the ContentPlaceHolders defined in Site.master.
- Add a Login control to the MainContent Content control.
- Likewise, feel free to add any content to the LeftColumnContent region.
- However, make sure to leave the Content control for the QuickLoginUI ContentPlaceHolder empty.
- This will ensure that the Login control does not appear in the left column of the login page.

After defining the content for the MainContent and LeftColumnContent regions, your login page's declarative markup should look similar to the following:

```
<%@ Page Language="C#" MasterPageFile="/Site.master" AutoEventWireup="true"
CodeFile="Login.aspx.cs" Inherits="Login" Title="Untitled Page" %>
<asp:Content ID="Content1" ContentPlaceHolderID="head" Runat="Server">
</asp:Content>
<asp:Content ID="Content2" ContentPlaceHolderID="MainContent" Runat="Server">
 <h2>
 Sign In</h2>
 <p>
 <asp:Login ID="Login1" runat="server" TitleText="">
 </asp:Login>
 </p>
</asp:Content>
<asp:Content ID="Content3" ContentPlaceHolderID="QuickLoginUI" Runat="Server">
</asp:Content>
<asp:Content ID="Content4" ContentPlaceHolderID="LeftColumnContent" Runat="Server">
 <h3>Sign In Tasks</h3>
 <ul>
 <li>Create a New Account</li>
 <li>Recover Forgotten Password</li>
 </ul>
 <p>TODO: Turn the above text into links...</p>
</asp:Content>
```

Figure 7 shows this page when viewed through a browser. Because this page specifies a Content control for the QuickLoginUI ContentPlaceHolder, it overrides the default content specified in the master page. The net effect is that the Login control displayed in the master page's Design view (see Figure 6) is not rendered in this page.

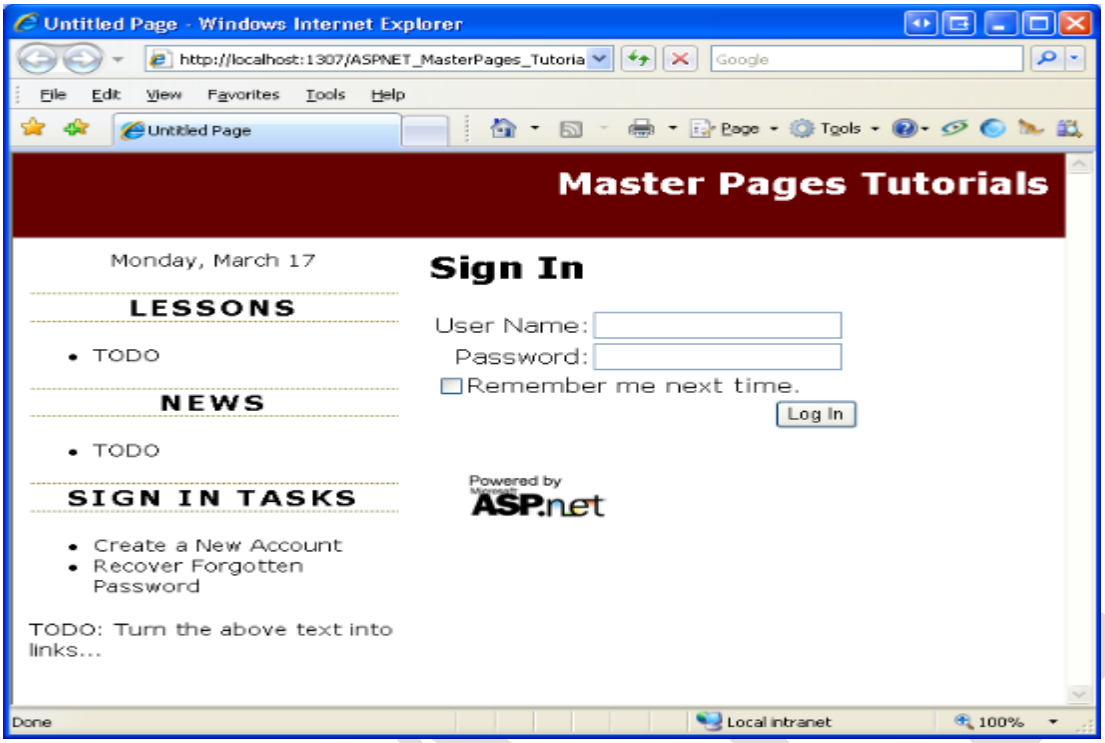


Figure 07: The Login Page Represses the QuickLoginUI ContentPlaceHolder's Default Content (Click to view full-size image)

Using the Default Content in New Pages

- We want to show the Login control in the left column for all pages except the Login page.
- To achieve this, all the content pages except for the login page should omit a Content control for the QuickLoginUI ContentPlaceHolder.

- By omitting a Content control, the ContentPlaceHolder's default content will be used instead.

- Our existing content pages - Default.aspx, About.aspx, and MultipleContentPlaceHolders.aspx - do not include a Content control for QuickLoginUI because they were created before we added that ContentPlaceHolder control to the master page.

- Therefore, these existing pages do not need to be updated. However, new pages added to the website include a Content control for the QuickLoginUI ContentPlaceHolder, by default.

- Therefore, we have to remember to remove these Content controls each time we add a new content page (unless we want to override the ContentPlaceHolder's default content, as in the case of the login page).

- To remove the Content control, you can either manually delete its declarative markup from the Source view or, from the Design view, choose the Default to Master's Content link from its smart tag.

- Either approach removes the Content control from the page and produces the same net effect.

Figure 8 shows Default.aspx when viewed through a browser. Recall that Default.aspx only has two Content controls specified in its declarative markup - one for head and one for MainContent. As a result, the default content for the LeftColumnContent and QuickLoginUI ContentPlaceHolders are displayed.

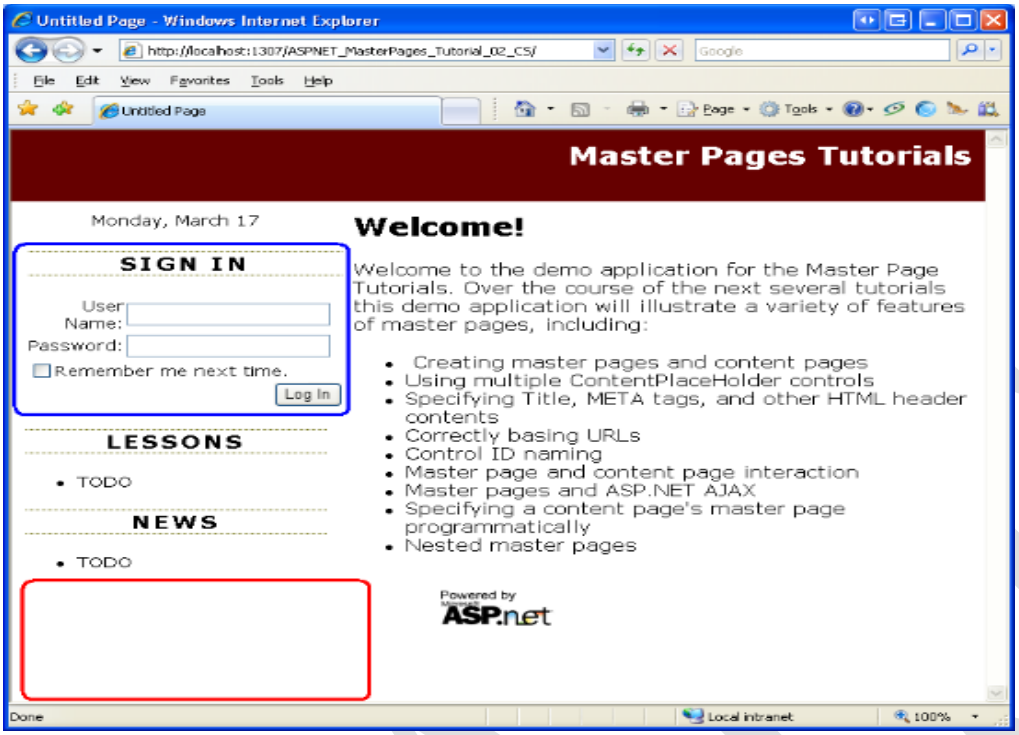


Figure 08: The Default Content for the LeftColumnContent and QuickLoginUI ContentPlaceHolders are Displayed.

Part B

1. Define ASP.NET. Differentiate ASP and ASP.NET.
2. Justify "ASP.NET is a server-side technology".
3. Write a note on inline code page and code behind page.
4. Explain importance of compiled code.
5. What do you understand by view state? Explain how ASP.NET maintains view state.
6. Differentiate InnerHtml and InnerText with example.
7. Differentiate Html server control and web server control.
8. What is a Namespace?

9. What is global.asax?
10. Explain CLR.
11. What is unstructured error handling?
12. Explain .net framework.
13. What is the use of causevalidation and validationgroup property of
14. asp.net control?
15. Write a code for bind data to list box control using any collection object.
16. What is the purpose of app_theme folder? How to apply theme to a particular page.

Part B

1. Write a note on CLR.
2. Explain the life cycle of a web page with relevant events.
3. What do you mean by post back? Explain various events occurred during post back. Also explain how will you apply view state at page level as well as control level.
4. What do you mean by literal control? Explain how it is useful in table server control with example.
5. What is ASP.Net? Explain feature of ASP.Net over ASP.
   Also Explain page-processing sequence of ASP.Net web page.
6. Write short note on:

Validation Server Control

OR

HTML Server Control

7. Explain ADRotator asp.net control.

8. Explain common properties of list control of ASP.Net.
9. Write design code for bind ename field of emp table to checklistbox control using sqldatasource asp.net control.
10. Write sort note on Event Driven Programming
11. Explain the featuresof ASP.Net.
12. Explain the various data types in vb.net.
13. Write the difference between the following:

(a) Static Web Page and Dynamic Web page.
(b) Html control and HTML Server Control.
(c) Range Validator and Compare Validator.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
Coimbatore-21

DEPARTMENT OF CS, CA & IT

UNIT- IV (Objective Type/ Multiple Choice Questions Each Question Carries One Mark)

.NET PROGRAMMING (17CAP502)

S.No	Question	Option1	Option2	Option3	Option4	Answer
1	What is the extension of a web user control file ?	.Asmx	.Ascx	.Aspx	.Aspz	. Ascx
2	Select the validation control used for “PatternMatching”	FieldValidator	RegularExpressio	RangeValidator	PatternValidator	RegularExpressionValidator
3	Which of the following server control shows data in a tab	ListBox	Repeater	Data Source	GridView	GridView
4	Skins with SkindID’s are known as _____	Application Skins	Named Skins	Default Skins	Reference Skins	Named Skins
5	Which of the following webserver control used as container for other server controls in a ASP.Net Page?	PlaceHolder	Table	Panel	IamgeMap	Panel
6	Attribute must be set on a validator control for the validat	ControlToValidate	ControlToBind	ValidateControl	Validate	ControlToValidate

7	If a developer of ASP.NET defines style information in a common location.Then that location is called as	Master Page	Theme	Customization	Skin	Theme
8	_____ checks the required fields have a value entere	RangeValidator	CustomValidato	CompareValidator	RequiredFieldVal idator	RequiredFieldVali dator
9	_____ forms a group of radio buttons that can be selected in a mutually exclusive manner	RadioButton	CheckBoxList	RadioButtonList	DropDownList	RadioButtonList
10	Which of the following does not have any visible i	Datagrid	Repeater	DropdownLi	Datalist	Repeater
11	In ASP.NET in form page the object which contain	Page.User.Identity	Page.User.IsInR	Page.User.Name	Page.User.Role	Page.User.Identity
12	Which of the following method must be overridden	The Paint() method	The Control_Bu	The default constru	The Render() meth	The Render() method
13	Which is the file extension used for an ASP.NET f	.asn	.asp	.aspn	.aspx	.aspx
14	Which property is used to name a web control?	ControlName	Designation	ID	Name	ID
15	Which user action will not generate a server-side event?	Mouse Move	Text Change	Button Click	Load	Mouse Move
16	What class does the ASP.Net Web Form class inhe	System.Web.Form	System.Web.UI.	System.Web.GUI.P	System.Web.UI.Pa	System.Web.UI.P age
17	Which of the following denote the web control associated with Table Control function of ASP.Net?	DataList	ListBox	TableRow	TableColumn	TableRow

18	ASP.Net separates the HTML output from progra	Exception	Code-behind	Code-front	Code-back	Code-behind
19	_____ Web Controls are used to achieve graphi	AdRotator	LinkButton	TextBox	Calendar	AdRotator
20	The Asp.net server control provides an alternative	<asp:listitem>	<asp:button>	<asp:label>	<asp:image>	<asp:label>
21	How will you add a TextBox control at runtime on the form? Choose the correct one.	TextBox obj = new TextBox();	form1.Controls.Add(TextBox);	this.FindControl.add(TextBox);	obj.ID = "txtUserName";	TextBox obj = new TextBox();
22	If a user wants to create controls at runtime which event should be used to write code?	PreLoad	Load	Init	PreInit	PreInit
23	Which attribute is necessary for HTML control to work as a HTML server control?	runat=”server”	runat=”web-server”	ID=”server”	ID=”web-server”	runat=”server”
24	Which is the first event of ASP.NET page, when user requests a web page ?	PreLoad	Load	Preinit	Init	Preinit
25	What is the fully qualified name of the base class of all server controls?	System.Web.UI.Cont	System.Web.UI	System.Control	Web.UI.Control	System.Web.UI.Control
26	Using which type of stylesheet we can change the style of an element in the entire website?	Internal Stylesheet	External Stylesh	Inline stylesheet	cascade stylesheet	External Stylesheet
27	If we want to add graphics using asp.net which of the following web control will you use?	Link Button	AdRotator	Grid View	view link	AdRotator
28	ASP.NET Validation Control works at _____	Client side only.	Server side only.	Both Client Side and Server Side	web server side	Both Client Side and Server Side

29	What object is used to manipulate files?	System.IO.Files	System.IO.Streams	Streams.IO.Files	System.IO.Directories	Streams.IO.Files
30	What object is used to manipulate directories?	System.IO.Files	System.IO.Streams	Streams.IO.Files	System.IO.Directories	System.IO.Directories
31	--------- is used to rename a file	System.IO.Files.Copy()	System.IO.Files.Move()	System.IO.Files.Rename()	System.IO.Files.CopyTo()	System.IO.Files.Move()
32	---------- is a programming structure that encapsulates data and functionality as a single unit.	Class	Object	Collection	methods	Object
33	This is the way we refer to properties of an object in code	{Object name}.{Property}	{Class}.{Property}	{Class}.{Method}	{Object Name}.{Method name}	{Object name}.{Property}
34	A property that returns an object is called ----------	Collection	subroutine	Object Oriented	Object Property	Object Property
35	The process of creating an object is called ----------	integration	instantiation	interfacing	inheritance	instantiation
36	A _____ is a place to store the code we write	class	module	method	subroutine	module
37	Codes written in _____ modules are always available and need not instantiate an object for it	class module	standard module	library modules	None of the above	standard module
38	We cannot create _____ based on standard modules	classes	methods	objects	properties	objects
39	Interfaces are similar to ---------- classes	abstract	collection	inheritance	polymorphism	abstract

40	How many types of constructors are there?	3	4	5	2	2
41	-------- are the special methods that are used to release the instance of a class from memory	constructors	destructors	inheritance	abstract	destructors
42	The ------ method is called to release a resource such as a database connection	compose()	release()	destroy()	dispose()	dispose()
43	File handling in Visual Basic is based on ---------- namespace	System.Input	System.Output	System.IO	System.Files	System.IO
44	How many access methods are there in file?	2	4	5	3	3
45	The ------ class provides access to files and file-related information	Stream	FileStream	StreamFile	FileMode	StreamFile
46	The FileStream class opens a file either in --------- mode	synchronous	asynchronous	synchronous or asynchronous	sequential	synchronous or asynchronous
47	The ------- method is used to open a file in synchronous mode	BeginRead()	Read()	BeginWrite()	Write()	Read(), Write()
48	The ------- method is used to open a file in asynchronous mode	Read()	BeginRead()	Write()	BeginWrite()	BeginRead(), BeginWrite()
49	By default FileStream class opens file in ------ mode	synchronous	asynchronous	sequential	random	synchronous
50	The ----- class is used to read from binary file	StreamReader	StreamWriter	BinaryReader	BinaryWriter	BinaryReader

51	The ----- class is used to write to binary file	StreamReader	BinaryReader	BinaryWriter	StreamWriter	BinaryWriter
52	The -------- method is used to set the file pointer to the beginning of the file	Offset	Peek()	Seek()	SeekOrigin()	Seek()
53	The ------- method is used to read characters from the file	Read()	ReadChars()	ReadCharacters()	BinaryRead()	ReadChars()
54	The System.IO model also enables to work with drives and folders by using the ------- class	File	Directory	Reader	Stream	Directory
55	How many methods are most frequently used in Directory class	5	3	7	6	6
56	The -------- method is used to delete a directory and all its contents	remove	destroy	delete	dispose	delete
57	VB.Net run-time functions allow ----- types of file access	4	2	5	3	3
58	The functions that allow the types of file access are deifned in the -------- namespace	System.IO	System.Assemblies	System.IO.Files.Rename()	System.IO.File	System.IO.File
59	The function that is used to retrieve the date and time when a file was created or modified is ----------	Dir	FileCopy	FileDateTime	FreeFile	FileDateTime
60	The ------ function is used to retrieve a vlaue specifying the current read/write position within an open file	GetAttr	FreeFile	FileDateTime	Loc	Loc
61	The ------ function is used to write data from a variable to a disk file	FreeFile	GetAttr	FilePut	Print	FilePut

62	The file I/O operations can be done in -------- ways	3	4	2	5	2
63	The ------ function is used to retrieve the next file number available for use by FileOpen() function	FreeFile	FileCopy	FileDateTime	FilePut	FreeFile
64	The ------ function is used to retrieve String value containing characters from a file opened in Input or Binary mode	PrintLine	InputString	FilePut	Loc	InputString
65	The function that allows to open a file in any access methods is ----------	Open()	Read()	Create()	FileOpen()	FileOpen()

UNIT-V

SYLLABUS

Data Binding in ASP.Net: Data source Controls – Configuring data source control caching – storing connection information-Using Bound list controls with Data Source Controls – Other Databound Controls-Data Management with ADO.Net

Data Binding in ASP.NET 2.0

Data Source Controls:

ASP.NET provides another set of controls for data bindings. Using the "*DataSource*" property of a data-bind control you can bind the data object to the control directly. Another way is to use *DataSource* controls. Any control that implements the "*IDataSource*" interface becomes a data source control and you can bind the data-source control to a data-bind control using the "*DataSourceID*" property.

DataSource controls are very easy to use and you can create a page with data without writing any code. But what you can does not always means what you should do. In professional applications, the DataSource controls are not generally used. Data access code should be separated from UI and be optimized for your business needs.

DataSource controls can still be very useful when you create a small personal application or the test pages to look at the result without relying on data access code.

1. DataSource Controls

DataSource controls are server controls that allow you to access the data source directly from a web page. They are configurable with parameters.

A parameter values can be obtained from:

- a cookie
- the session
- a form field
- a query string

The *DataSource* controls can be bound to data-bind controls very easily. You just need to add a the DataSource control to a page, set the properties of it, and set the "*DataSourceControlID*" property of a data-bind control. That's all. The magic will happen.

The DataSource controls pefrom data binding after the "*PreRender*" event, which is the last page creation stage before HTML output is rendered. When you perform "*Update*" or "*Insert*" operations using the controls, the events are handled just before the "*PreRender*" event. This life cycle is very important. When you peform "update" or "insert", you cannot catch the change in the "Page_Load".
ASP.NET provides a couple of built-in controls for you.

- *SqlDataSource*: ADO.NET provider (SQL Server, Oracle, OLE DB, ODBC, and a new access file (.accdb))
- *AccessDataSource*: Access database file (.mdb)
- *XmlDataSource*: XML files
- *SiteMapDataSource*: .sitemap
- *LinqDataSource*: LINQ to SQL
- *ObjectDataSource*: a custom data access class
- *EntityDataSource*: Entity Framework

Some controls can cache data: *ObjectDataSource*, *SqlDataSource*, and *AccessDataSource* controls.

2. IDataSource interface

Any controls that implements "*System.Web.UI.IDataSource*" interface can be a*DataSource* control.
The interface has 2 methods to be implemented:

- DataSourceView *GetView*(string viewName)
- ICollection *GetViewNames*()

3. SqlDataSource
The "*SqlDataSource*" control represents the data access to an ADO.NET provider.
- Attributes -

- *ID*: used when referring to the data source during data binding
- *ProviderName*: System.Data.SqlClient, System.Data.OracleClient, System.Data.OleDb, or System.Data.Odbc

- *ConnectionString*: connection string or a page script to read the connection string (<%$ ConnectionStrings:NorthwindConnectionString %>)
- *SelectCommand, UpdateCommand, InsertCommand, DeleteCommand*:
- *SelectCommandType, UpdateCommandType, InsertCommandType, DeleteCommandType*: SqlDataSourceCommandType.Text / SqlDataSourceCommandType.StoredProcedure
- *DataSourceMode*: SqlDataSourceMode.DataSet / SqlDataSourceMode.DataReader

- Filtering -
- The "*DataSourceMode*" attribute must be set to "*DataSet*"
- Use "*FilterExpression*" and "*FilterParameters*" attributes

- Caching -
- The "*DataSourceMode*" attribute must be set to "*DataSet*"
- Use "*EnableCaching*", "*CacheExpirationPolicy*", and "*CacheDuration*" attributes
- Even without writing a single line of code, you can create a web page with 2 data-bind controls (parent-child).

- Exception Handlings -
- Handle *Selected*, *Updated*, *Inserted* and *Deleted* events
- Check the *Exception* property of the *EventArgs* object
-

4. AccessDataSource

The "*AccessDataSource*" control works with Access database (.mdb). The "*DataFile*" attribute is used instead of the "*ConnectionString*" attribute.
You need to use *SqlDataSource* to connect to the newer version of Access files (.accdb).

Provider=Microsoft.ACE.OLEDB.12.0;

Data Source=C:\myFolder\myAccess2007file.accdb;

Persist Security Info=False;

5. XmlDataSource

The "*XmlDataSource*" control is for XML files, which are typically stored in *App_Data*folder.

- Attributes -

- *DataFile*: the path of the xml file
- *TransformFile*: the path of the xsl (XML Stylesheet Language) file
- *XPath*: filtering using XPath expressions

```
<asp:XmlDataSource ID = "XmlDataSource1"
runat="server"
 DataFile="~/App_Data/product.xml"
 TransformFile="~/App_Data/producttransform.xsl"
 XPath="/Products/Product[Category='Food']" >
</ asp:XmlDataSource>
```

6. SiteMapDataSource

The "*SiteMapDataSource*" control automatically picks up the "*Web.sitemap*" file in the root directory of the web application.

- Attributes -

- StartingNodeUrl:
- ShowStartingNode:
- StartingFromCurrentNode

7. LinqDataSource

The "*LinqDataSource*" control is used with "*LINQ to SQL*".

- Attributes -

- *ContextTypeName*: the name of the class that represents the database context
- *EnableDelete*, *EnableInsert*, *EnableUpdate*: Boolean
- *TableName*

- Sorting -

- OrderBy

```
<asp:LinqDataSource ID = "LinqDataSource1" runat="server"
ContextTypeName="NorthwindDataContext"
EnableDelete="True" EnableInsert="True" EnableUpdate="True"
TableName="Suppliers" OrderBy="CompanyName" Where="Country==@Country" >
<WhereParameters>
  <asp:QueryStringParameter DefaultValue="us" Name="Country"
          QueryStringField="country" Type="String" />
</WhereParameters>
</ asp: LinqDataSource >
```

8. EntityDataSource

The "*EntityDataSource*" control is used with "*Entity Framework*" which becomes replacing LINQ to SQL.

9. ObjectDataSource

The "*ObjectDataSource*" control is used with a custom data access object.

- Attributes -

- *TypeName*: the type name of the object for DB access
- *DataObjectTypeName*: the type name of the object for Data Model
- *SelectMethod*: works with IEnumerable, IListSource, IDataSource, IHierarchicalDataSource (return classes as a DataTable, DataSet, or some form of a collection)
- *InsertMethod*, *UpdateMethod*, and *DeleteMethod*:

- Filtering -

- Use *FilterExpression* and *FilterParameters* attributes

- Sorting-

- Use *SortParameterName* attribute

- Paging -

- *EnablePaging*:

- *StartRowIndexParameterName*:
- *MaximumRowsParameterName*:
- *SelectCountMethod*:

- Caching -

- *EnableCaching*:
- *CacheDuration*: in seconds

When you create a "Data Object" class, you need to decorate a class properly.

- Set the "*DataObject*" attribute to the class
- Add "*DataObjectMethod*" attribute to methods – Pass a "*DataObjectMethodType*" enum (*Select*, *Insert*, *Update*, and *Delete*)

```
[System.ComponentModel.DataObject()]
public class MyData
{
    [System.ComponentModel.DataObjectMethod(Syste.ComponentModel.DataObjectMethod.Select)]
    public static DataTable GetAllData()
    {
    }
}
```

Data Binding in ASP.NET

The following controls are list controls which support data binding:

- asp:RadioButtonList
- asp:CheckBoxList
- asp:DropDownList

- asp:Listbox

The selectable items in each of the above controls are usually defined by one or more asp:ListItem controls, like this:

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="countrylist" runat="server">
<asp:ListItem value="N" text="Norway" />
<asp:ListItem value="S" text="Sweden" />
<asp:ListItem value="F" text="France" />
<asp:ListItem value="I" text="Italy" />
</asp:RadioButtonList>
</form>
</body>
</html>
```

- However, with data binding we may use a separate source, like a database, an XML file, or a script to fill the list with selectable items.
- By using an imported source, the data is separated from the HTML, and any changes to the items are made in the separate data source.

Create an ArrayList

- The ArrayList object is a collection of items containing a single data value.
- Items are added to the ArrayList with the Add() method.

The following code creates a new ArrayList object named mycountries and four items are added:

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New ArrayList
  mycountries.Add("Norway")
  mycountries.Add("Sweden")
  mycountries.Add("France")
  mycountries.Add("Italy")
```

```
end if
end sub
</script>
```

- By default, an ArrayList object contains 16 entries.

An ArrayList can be sized to its final size with the TrimToSize() method:

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New ArrayList
  mycountries.Add("Norway")
  mycountries.Add("Sweden")
  mycountries.Add("France")
  mycountries.Add("Italy")
  mycountries.TrimToSize()
end if
end sub
</script>
```

An ArrayList can also be sorted alphabetically or numerically with the Sort() method:

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New ArrayList
  mycountries.Add("Norway")
  mycountries.Add("Sweden")
  mycountries.Add("France")
  mycountries.Add("Italy")
  mycountries.TrimToSize()
  mycountries.Sort()
end if
end sub
</script>
```

To sort in reverse order, apply the Reverse() method after the Sort() method:

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New ArrayList
  mycountries.Add("Norway")
  mycountries.Add("Sweden")
  mycountries.Add("France")
  mycountries.Add("Italy")
  mycountries.TrimToSize()
  mycountries.Sort()
  mycountries.Reverse()
end if
end sub
</script>
```

Data Binding to an ArrayList

An ArrayList object may automatically generate the text and values to the following controls:

- asp:RadioButtonList
- asp:CheckBoxList
- asp:DropDownList
- asp:Listbox

To bind data to a RadioButtonList control, first create a RadioButtonList control (without any asp:ListItem elements) in an .aspx page:

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" />
</form>
</body>
</html>
```

Then add the script that builds the list and binds the values in the list to the RadioButtonList control:

Example

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New ArrayList
  mycountries.Add("Norway")
  mycountries.Add("Sweden")
  mycountries.Add("France")
  mycountries.Add("Italy")
  mycountries.TrimToSize()
  mycountries.Sort()
  rb.DataSource=mycountries
  rb.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" />
</form>
</body>
</html>
```

- The DataSource property of the RadioButtonList control is set to the ArrayList and it defines the data source of the RadioButtonList control.
- The DataBind() method of the RadioButtonList control binds the data source with the RadioButtonList control.

Note: The data values are used as both the Text and Value properties for the control. To add Values that are different from the Text, use either the Hashtable object or the SortedList object.

Create a Hashtable

- The Hashtable object contains items in key/value pairs.
- The keys are used as indexes, and very quick searches can be made for values by searching through their keys.

- Items are added to the Hashtable with the Add() method.

The following code creates a Hashtable named mycountries and four elements are added:

```
<script runat="server">
Sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New Hashtable
  mycountries.Add("N","Norway")
  mycountries.Add("S","Sweden")
  mycountries.Add("F","France")
  mycountries.Add("I","Italy")
end if
end sub
</script>
```

Data Binding

A Hashtable object may automatically generate the text and values to the following controls:

- asp:RadioButtonList
- asp:CheckBoxList
- asp:DropDownList
- asp:Listbox

To bind data to a RadioButtonList control, first create a RadioButtonList control (without any asp:ListItem elements) in an .aspx page:

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>
```

```
</body>
</html>
```

Then add the script that builds the list:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New Hashtable
  mycountries.Add("N","Norway")
  mycountries.Add("S","Sweden")
  mycountries.Add("F","France")
  mycountries.Add("I","Italy")
  rb.DataSource=mycountries
  rb.DataValueField="Key"
  rb.DataTextField="Value"
  rb.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>
</body>
</html>
```

Then we add a sub routine to be executed when the user clicks on an item in the RadioButtonList control.

When a radio button is clicked, a text will appear in a label:

Example

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
```

```
  dim mycountries=New Hashtable
  mycountries.Add("N","Norway")
  mycountries.Add("S","Sweden")
  mycountries.Add("F","France")
  mycountries.Add("I","Italy")
  rb.DataSource=mycountries
  rb.DataValueField="Key"
  rb.DataTextField="Value"
  rb.DataBind()
end if
end sub
sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
</script>

<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lbl1" runat="server" /></p>
</form>
</body>
</html>
```

The SortedList Object

- The SortedList object contains items in key/value pairs. A SortedList object automatically sort the items in alphabetic or numeric order.
- Items are added to the SortedList with the Add() method. A SortedList can be sized to its final size with the TrimToSize() method.

The following code creates a SortedList named mycountries and four elements are added:

```
<script runat="server">
sub Page_Load
```

```
if Not Page.IsPostBack then
  dim mycountries=New SortedList
  mycountries.Add("N","Norway")
  mycountries.Add("S","Sweden")
  mycountries.Add("F","France")
  mycountries.Add("I","Italy")
end if
end sub
</script>
```

Data Binding

A SortedList object may automatically generate the text and values to the following controls:

- asp:RadioButtonList
- asp:CheckBoxList
- asp:DropDownList
- asp:Listbox

To bind data to a RadioButtonList control, first create a RadioButtonList control (without any asp:ListItem elements) in an .aspx page:

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>
</body>
</html>
```

Then add the script that builds the list:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New SortedList
  mycountries.Add("N","Norway")
```

```
  mycountries.Add("S","Sweden")
  mycountries.Add("F","France")
  mycountries.Add("I","Italy")
  rb.DataSource=mycountries
  rb.DataValueField="Key"
  rb.DataTextField="Value"
  rb.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>
</body>
</html>
```

- Then we add a sub routine to be executed when the user clicks on an item in the RadioButtonList control.

When a radio button is clicked, a text will appear in a label:

Example

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New SortedList
  mycountries.Add("N","Norway")
  mycountries.Add("S","Sweden")
  mycountries.Add("F","France")
  mycountries.Add("I","Italy")
  rb.DataSource=mycountries
  rb.DataValueField="Key"
  rb.DataTextField="Value"
  rb.DataBind()
```

```
end if
end sub
sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lbl1" runat="server" /></p>
</form>
</body>
</html>
```

Bind a DataSet to a List Control

First, import the "System.Data" namespace. We need this namespace to work with DataSet objects.

Include the following directive at the top of an .aspx page:

```
<%@ Import Namespace="System.Data" %>
```

Next, create a DataSet for the XML file and load the XML file into the DataSet when the page is first loaded:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New DataSet
  mycountries.ReadXml(MapPath("countries.xml"))
end if
end sub
```

To bind the DataSet to a RadioButtonList control, first create a RadioButtonList control (without any asp:ListItem elements) in an .aspx page:

```
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server" AutoPostBack="True" />
</form>
</body>
</html>
```

Then add the script that builds the XML DataSet:

```
<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New DataSet
  mycountries.ReadXml(MapPath("countries.xml"))
  rb.DataSource=mycountries
  rb.DataValueField="value"
  rb.DataTextField="text"
  rb.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
</form>
</body>
</html>
```

Then we add a sub routine to be executed when the user clicks on an item in the RadioButtonList control. When a radio button is clicked, a text will appear in a label:

Example

```
<%@ Import Namespace="System.Data" %>

<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycountries=New DataSet
  mycountries.ReadXml(MapPath("countries.xml"))
  rb.DataSource=mycountries
  rb.DataValueField="value"
  rb.DataTextField="text"
  rb.DataBind()
end if
end sub
sub displayMessage(s as Object,e As EventArgs)
lbl1.text="Your favorite country is: " & rb.SelectedItem.Text
end sub
</script>
<html>
<body>
<form runat="server">
<asp:RadioButtonList id="rb" runat="server"
AutoPostBack="True" onSelectedIndexChanged="displayMessage" />
<p><asp:label id="lbl1" runat="server" /></p>
</form>
</body>
</html>
```

Bind a DataSet to a Repeater Control

- The Repeater control is used to display a repeated list of items that are bound to the control.
- The Repeater control may be bound to a database table, an XML file, or another list of items.

- Here we will show how to bind an XML file to a Repeater control.

We will use the following XML file in our examples ("cdcatalog.xml"):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<catalog>
<cd>
  <title>Empire Burlesque</title>
  <artist>Bob Dylan</artist>
  <country>USA</country>
  <company>Columbia</company>
  <price>10.90</price>
  <year>1985</year>
</cd>
<cd>
  <title>Hide your heart</title>
  <artist>Bonnie Tyler</artist>
  <country>UK</country>
  <company>CBS Records</company>
  <price>9.90</price>
  <year>1988</year>
</cd>
<cd>
  <title>Greatest Hits</title>
  <artist>Dolly Parton</artist>
  <country>USA</country>
  <company>RCA</company>
  <price>9.90</price>
  <year>1982</year>
</cd>
<cd>
  <title>Still got the blues</title>
  <artist>Gary Moore</artist>
  <country>UK</country>
  <company>Virgin records</company>
  <price>10.20</price>
  <year>1990</year>
</cd>
<cd>
  <title>Eros</title>
  <artist>Eros Ramazzotti</artist>
```

```
  <country>EU</country>
  <company>BMG</company>
  <price>9.90</price>
  <year>1997</year>
</cd>
</catalog>
```

Take a look at the XML file: cdcatalog.xml

First, import the "System.Data" namespace. We need this namespace to work with DataSet objects. Include the following directive at the top of an .aspx page:

```
<%@ Import Namespace="System.Data" %>
```

Next, create a DataSet for the XML file and load the XML file into the DataSet when the page is first loaded:

```
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycdcatalog=New DataSet
  mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
end if
end sub
```

Then we create a Repeater control in an .aspx page.

The contents of the <HeaderTemplate> element are rendered first and only once within the output, then the contents of the <ItemTemplate> element are repeated for each "record" in the DataSet, and last, the contents of the <FooterTemplate> element are rendered once within the output:

```
<html>
<body>
<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">
<HeaderTemplate>
...
```

```
</HeaderTemplate>

<ItemTemplate>
...
</ItemTemplate>

<FooterTemplate>
...
</FooterTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

- Then we add the script that creates the DataSet and binds the mycdcatalog DataSet to the Repeater control.

We also fill the Repeater control with HTML tags and bind the data items to the cells in the<ItemTemplate> section with the <%#Container.DataItem("fieldname")%> method:

Example

```
<%@ Import Namespace="System.Data" %>
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycdcatalog=New DataSet
  mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
  cdcatalog.DataSource=mycdcatalog
  cdcatalog.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">
<HeaderTemplate>
```

```
<table border="1" width="100%">
<tr>
<th>Title</th>
<th>Artist</th>
<th>Country</th>
<th>Company</th>
<th>Price</th>
<th>Year</th>
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</ItemTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

Using the <AlternatingItemTemplate>

You can add an <AlternatingItemTemplate> element after the <ItemTemplate> element to describe the appearance of alternating rows of output.

In the following example each other row in the table will be displayed in a light grey color:

Example

```
<%@ Import Namespace="System.Data" %>
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycdcatalog=New DataSet
  mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
  cdcatalog.DataSource=mycdcatalog
  cdcatalog.DataBind()
end if
end sub
</script>
<html>
<body>
<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">
<HeaderTemplate>
<table border="1" width="100%">
<tr>
<th>Title</th>
<th>Artist</th>
<th>Country</th>
<th>Company</th>
<th>Price</th>
<th>Year</th>
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</ItemTemplate>
<AlternatingItemTemplate>
```

```
<tr bgcolor="#e8e8e8">
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</AlternatingItemTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

Using the <SeparatorTemplate>

The <SeparatorTemplate> element can be used to describe a separator between each record.

The following example inserts a horizontal line between each table row:

Example

```
<%@ Import Namespace="System.Data" %>
<script runat="server">
sub Page_Load
if Not Page.IsPostBack then
  dim mycdcatalog=New DataSet
  mycdcatalog.ReadXml(MapPath("cdcatalog.xml"))
  cdcatalog.DataSource=mycdcatalog
  cdcatalog.DataBind()
end if
end sub
</script>
<html>
<body>
```

```
<form runat="server">
<asp:Repeater id="cdcatalog" runat="server">
<HeaderTemplate>
<table border="0" width="100%">
<tr>
<th>Title</th>
<th>Artist</th>
<th>Country</th>
<th>Company</th>
<th>Price</th>
<th>Year</th>
</tr>
</HeaderTemplate>
<ItemTemplate>
<tr>
<td><%#Container.DataItem("title")%></td>
<td><%#Container.DataItem("artist")%></td>
<td><%#Container.DataItem("country")%></td>
<td><%#Container.DataItem("company")%></td>
<td><%#Container.DataItem("price")%></td>
<td><%#Container.DataItem("year")%></td>
</tr>
</ItemTemplate>
<SeparatorTemplate>
<tr>
<td colspan="6"><hr /></td>
</tr>
</SeparatorTemplate>
<FooterTemplate>
</table>
</FooterTemplate>
</asp:Repeater>
</form>
</body>
</html>
```

Caching Data Using Data Source Controls

- Data source controls provide data services to data-bound controls such as the GridView, FormView, and DetailsView controls.
- These services include caching data to help improve the performance of applications in which the data does not change frequently.

- To cache data using the SqlDataSource or AccessDataSource controls, you must set the DataSourceMode property of those controls to DataSet.

- The ObjectDataSource control can cache objects returned by the underlying business object.

- However, you should not cache objects that hold resources or that maintain state that cannot be shared among multiple requests, such as an open DataReader object.

- Caching is not enabled by default for data source controls, but you can enable it by setting the control's EnableCaching property to true.

- Cached data is refreshed based on the number of seconds you specify using the CacheDuration property.

- You can further refine the behavior of a data source control's caching behavior by setting its CacheExpirationPolicy property.

- Setting the property's value to Absolute forces the cache to be refreshed when the CacheDuration value is exceeded.

- Setting the CacheExpirationPolicy property to Sliding refreshes the cache only if the CacheDuration value has been exceeded since the last time the cached item was accessed.

```
<%@ Page language="VB" %>

<!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>ASP.NET Example</title>
</head>
```

```
<body>
    <form id="form1" runat="server">

        <asp:SqlDataSource
          id="SqlDataSource1"
          runat="server"
          DataSourceMode="DataSet"
          ConnectionString="<%$ ConnectionStrings:MyNorthwind%>"
          EnableCaching="True"
          CacheDuration="20"
                SelectCommand="SELECT  EmployeeID,FirstName,LastName,Title  FROM
Employees">
        </asp:SqlDataSource>

        <asp:GridView
          id="GridView1"
          runat="server"
          AutoGenerateColumns="False"
          DataSourceID="SqlDataSource1">
          <columns>
            <asp:BoundField HeaderText="First Name" DataField="FirstName" />
            <asp:BoundField HeaderText="Last Name" DataField="LastName" />
            <asp:BoundField HeaderText="Title" DataField="Title" />
          </columns>
        </asp:GridView>

    </form>
  </body>
</html>
```

Storing Connection Information

- One of the best aspects of the .NET Frameworkis ADO.NET and data providers that negate the need for ODBCconnections.
- Data providers offer a liaison between your code andthe database.
- They also provide better performance, as well aseasier setup (no one misses the process of installing andconfiguring an ODBC driver).
- .NET database providers make it necessary tospecify database connection information, but it's the developer'sdiscretion to decide where to store this information.

- Connecting to a database requires knowledge ofthe database server, as well as the username and password to usewhen accessing it.
- Also, you can specify parameters such as whichdatabase to use based upon your database platform.

The following VB.NET code demonstrates a sampleconnection string for working with SQL Server:

```
Dim sConn As String
Dim conn As SqlConnection
sConn = "server=(local);InitialCatalog=Northwind;UID=tester;PWD=123456"
conn = New SqlConnection(sConn)
conn.Open()
```

Here's the C# equivalent:

```
string sConn;
SqlConnection conn;
sConn = "server=(local);InitialCatalog=Northwind;UID=tester;PWD=123456";
conn = new SqlConnection(sConn);
conn.Open();
```

- This code works but there are potentialproblems.
- First, including database connection information in yourcode is a security risk; a malicious user could use thisinformation to gain access.
- Also, problems may arise if theconnection string changes—you must recompile the code.
- For these reasons, developers often choose tostore database connection information outside of the code.
- Thereare many options for accomplishing this task, but two popularchoices are using an XML configuration file or utilizing theWindows registry.

Choosing XML

- Storing database connection properties in anXML file allows you to change the settings without recompiling thecode.
- It's easy to edit the XML file with any standard text editor.A good example of this approach is utilizing the XML-basedweb.config file in an ASP.NET application.

- This file allows you to add custom elements(which are accessible from the code) to the application.
- Inaddition, this file isn't viewable from a Web browser so itscontents remain hidden.

You can add new items using the add elementwith key and value attributes. The following sample web.configshows a possible way to add the connection:

```
<configuration>
<add key="dbconnection" value=" server=(local);
Initial Catalog=Northwind;UID=tester;PWD=123456"/>
</configuration>
```

- With this data stored in the web.config file,you can use it in your code with the ConfigurationSettings class.

The following VB.NET reads the connection string from theapplication's web.config file.

It uses the AppSettings property andpasses the key to be read to it.

```
Dim sConn As String
Dim conn As SqlConnection
sConn = ConfigurationSettings.AppSettings("dbconnection")
conn = New SqlConnection(sConn)
conn.Open()
```

Here is the code in C#:

```
String sConn;
SqlConnection conn;
sConn = ConfigurationSettings.AppSettings("dbconnection");
conn = new SqlConnection(sConn);
conn.Open();
```

- You may also use this straightforward approachwith a simple XML file (for a non-ASP.NET application).

- The problemis that it's still open to security threats. If a malicious usergains access to the server, the web.config file is a simple textfile that they can easily read to gain database access.
- To provide tighter access control, manydevelopers utilize the Windows registry to store connectioninformation.
- With the information in the registry, it's easy toread it in the code.

Choosing the registry

- The Microsoft.Win32 namespace provideseverything necessary to interact with a Windows registry.
- You canuse its CreateSubKey method to create registry entries.
- The following C# snippet will do the trick.
- Ituses the Registry class to access a machine's Windows registry, andit uses the RegistryKey class to work with individual elementswithin the registry.

```
Using Microsoft.Win32;
class Class1 {
static void Main(string[] args) {
Registry.LocalMachine.CreateSubKey("SOFTWARE\\MyApp\DBConn"];
RegistryKey regKey;
regKey =Registry.OpenSubKey("SOFTWARE\\MyApplication\\DBConn");
if (regKey != null) {
regKey.SetValue("Server","(local)");
regKey.SetValue("Database","Northwind");
regKey.SetValue("UserID","tester");
regKey.SetValue("Password","123456");
} } }
```

This is the VB.NET equivalent:

```
Imports Microsoft.Win32;
Module Module1
Sub Main()
Registry.LocalMachine.CreateSubKey("SOFTWARE\\MyApp\\DBConn")
Dim regKey As RegistryKey
If Not (regKey Is Nothing) Then
regKey.SetValue("Server","(local)")
regKey.SetValue("Database","Northwind")
```

```
regKey.SetValue("UserID,"tester")
regKey.SetValue("Password","123456")
End If
End Sub
End Module
```

- This approach is inherently more secure since aperson will need full server access to work with its registry.
- Withthe necessary values stored in the registry, you may utilize themto connect to the database.
- You can perform this with the GetValuemethod of the ResourceKey class.

```
using Microsoft.Win32;
using System.Data.SqlClient;
class Class1 {
static void Main(string[] args) {
RegistryKey regKey;
string server, db, uid, pwd
string sConn
SqlConnection conn;
regKey =Registry.LocalMachine.OpenSubKey("SOFTWARE\\MyApp\\DBConn");
if (regKey != null) {
server = regKey.GetValue("Server").ToString();
db = regKey.GetValue("Database").ToString();
uid = regKey.GetValue("UserID").ToString();
pwd = regKey.GetValue("Password).ToString();
sConn = "server=" + server + ";Initial Catalog=" + db + ";UID=" +uid + ";PWD=" +
pwd;
conn = new SqlConnection(sConn);
conn.Open();
} } }
```

This is the VB.NET equivalent:

```
Imports Microsoft.Win32;
Imports System.Data.SqlClient
Module Module1
Sub Main()
```

```
Dim regKey As RegistryKey
Dim server, db, uid, pwd As String
Dim sConn As String
regKey =Registry.LocalMachine.OpenSubKey("SOFTWARE\\MyApp\\DBConn")
If Not (regKey Is Nothing) Then
server = regKey.GetValue("Server").ToString()
db = regKey.GetValue("Database").ToString()
uid = regKey.GetValue("UserID").ToString()
pwd = regKey.GetValue("Password").ToString()
sConn = "server=" + server + ";Initial Catalog=" + db + ";UID=" +uid + ";PWD=" +
pwd
conn = New SqlConnection(sConn)
conn.Open()
End If
End Sub
End Module
```

Peppering the registry calls throughout anapplication is tedious--especially if you need to change anything.Many developers place these calls in the Global.asax (for ASP.NETapplications) or create a special class for it.

Note: Editing the registryis risky, so make sure you have a verified backup before making anychanges.

- Using Boundlist controls with data source controls:
- 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.
- Selection of data

- Managing presentation aspects like paging, caching etc.

- Manipulation of 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 and table-based data source controls.

The data source controls used for hierarchical data are:

- **XMLDataSource**-allows binding to XML files and strings with or without schema information
- **SiteMapDataSource**-allows binding to a provider that supplies site map information

The data source controls used for tabular data are:

Data source controls	Description
SqlDataSource	represents a connection to an ADO.Net data provider that returns SQL data, including data sources accessible via OLEDB and QDBC
ObjectDataSource	allows binding to a custom .Net business object that returns data
LinkdataSource	allows binding to the results of a Link-to-SQL query (supported by ASP.Net 3.5 only)
AccessDataSource	represents connection to a Microsoft Access database

The Data Source Views

- Data source views are objects of the DataSourceView class and represent a customized view of data for different data operations like 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.

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.

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 for 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, 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 our 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 to be used for retrieving data.

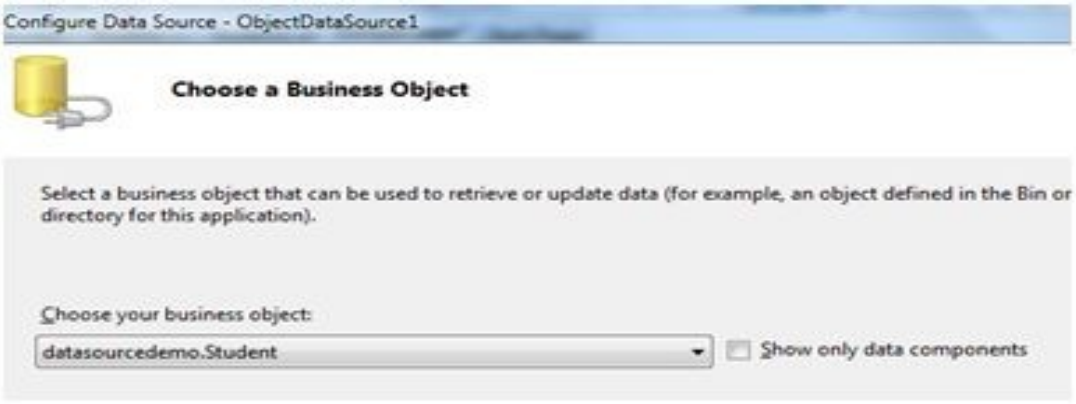
The student class:

```
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;
  }
}
```

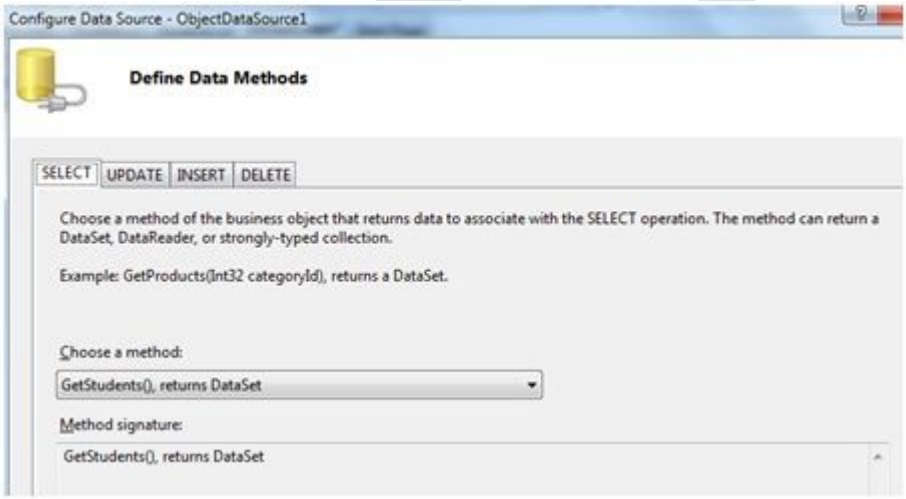
Take the following steps to bind the object with an object data source and retrieve data:

- 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.

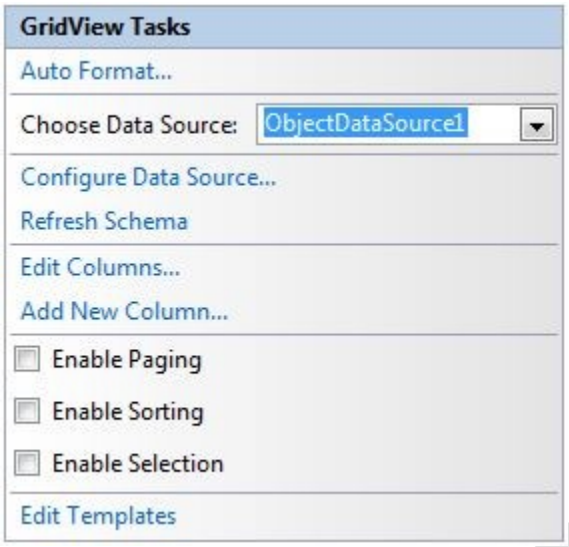
- Place a object data source control in the web form.
- Configure the data source by selecting the object.



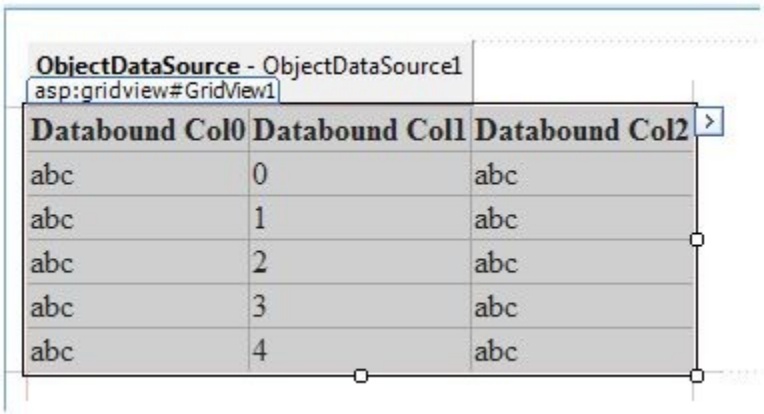
- Select a data method(s) for different operations on data. In this example, there is only one method.



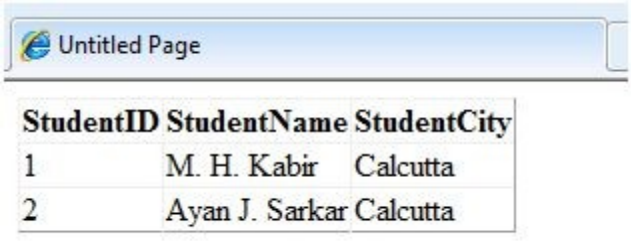
- Place a data bound control like grid view on the page and select the object data source as its underlying data source



- At this stage, the design view should look like the following:



- Run the project, it retrieves the hard coded tuples from the students class.



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 Bound Controls

In this tutorial you will learn about Data Bound Controls - The Hierarchy of Data Bound Controls, Simple Data Bound Controls, Composite DataBound Controls and Hierarchical Data Bound Controls.

The Hierarchy of Data Bound Controls

- Data Bound controls are controls that are bound to data sources.
- Traditionally the DataGrid is the principal data bound control in ASP.NET 1.x. Though DataGrid is still supported, ASP.NET 2.0 introduces three new controls—GridView, FormView and DetailsView.
- Unlike in ASP.NET 1.x, all controls descend from the BaseDataBoundControl class. It has two basic child classes DataBoundControl and HierarchicalDataBoundControl.
- While TreeView and Menu are examples of the latter, AdRotator, ListControls such as BulletedList, CheckboxList, DropDownList, ListBox and RadioButtonList, and CompositeDataBoundControls such as DetailsView, FormsView and GridView are examples of the former.
- All Data bound controls can be classified into Simple, composite and hierarchical controls.
- The DataBoundControl Base class defines the common characteristics of non-hierarchical controls which share the same base class.
- It is inherited from the WebControl and has all the visual and style properties of the base class.
- Additionally it has infrastructural properties such as Context, Page and AccessKey.
- DataMember property of the control selects the list of data that the control has to bind to when the DataSource contains more than one list.
- DataSource indicates the source of the data it has to bind to. It is imperative that the DataSource must be an object (unlike ASP.NET 1.x) that implements the IEnumerable or the IListSource interface.
- DataSourceId is the ID of the data source object used to retrieve data.
- The DataBoundControl class has only one method—the DataBind method.
- This method is called when data has to be pumped out of the data source.

- The internal implementation of this method has been tweaked and modified to enhance performance.

- It takes into account the implementation of the IEnumerable based data sources but is more sophisticated in execution as it implements the new interfaces and new data source controls also.

Simple Data Bound Controls

- List based user interface controls have been classified as Simple Data bound controls. ASP.NET 2.0 has two simple data bound controls—the AdRotator and the list control.
- The AdRotator control retrieves information from a XML file which contains the path to the distinctive image, the URL to go to when the control is clicked and the frequency of the Ad. Arbitrary data sources are supported by the control in ASP.NET 2.0 and it is fully bound to the data source.

- This control has additional capability of creating popup and pop under ads apart from standard banners.

- It supports counters for tracking purposes and updates the ad when the counter is clicked.

- Each ad can be associated with a separate counter.

- The BulletedList control is a new addition to list controls in ASP.NET.

- It can be used to create a list of formatted list items. Individual items can be specified by defining a ListItem for each object.

- The bulleted lists are filled in from the data source and receives the DataTable returned by a query.

- The DataTextField property of the control selects the column to show and the DisplayMode sets the display mode.

- The Hyperlink mode links the page directly to an external URL and click is handled by the browser.

- The LinkButton mode click is handled by the ASP.NET runtime and fires a server side event.

- The OnClick handler will have to be defined in this instance.
- The Index property of the BulletedListEventArgs class contains the base index of the clicked item.
- The BulletStyle property controls the customization of the bullet styles.
- There is no change in the other list controls in ASP.NET 2.0.

Composite DataBound Controls

- Two new base classes have been added to enhance the Composite Data bound controls.
- The new CompositeControl and CompositeDataBoundControl are separate classes with a similar blueprint.
- The CompositeControl class addresses the UI-based needs and CompositeDataBoundControl defines the common foundation for all composite data bound controls.
- The CompositeDataBoundControl is an abstract class that declares and implements the Controls property.
- In ASP.NET 1.x the Controls property stores the references to child controls. The CompositeDataBoundControl additionally exposes the CreateChildControls property.
- It takes a Boolean argument and behaves in accordance with the argument taken to ignore or initialize the view state.
- In ASP.NET 2.0 the developer has to only call the PerformDataBinding method and all boilerplate tasks are performed and the implementation of the CreateChildControls is called to implement the building of the control tree.
- We shall see examples of the GridView, the DetailsView and the FormView controls a little later in this tutorial.

Hierarchical Data Bound Controls

- The HierarchicalDataBoundControl class is the base class for hierarchical data bound controls such as TreeView and Menu.

- It is an abstract class but does not have any predefined services.
- It behaves like a logical container for controls that consume the hierarchical data.
- The TreeView control displays a hierarchy of nodes. Each node may contain child nodes.
- Parent nodes can be expanded to display child nodes or collapsed. Checkboxes can be displayed next to nodes or images from an imagelist control can be displayed.
- Nodes can be programmatically selected or cleared.
- The key properties of the TreeView control are Nodes and SelectedNodes.
- The Nodes property defines the top level nodes in the TreeView.
- The SelectedNode property sets the currently selected node.
- The TreeView control binds to any data source object that implements the IHierarchicalDataSource interface.
- It also exposes a DataSource property of the type object which can be assigned to an XmlDataDocument or to plain XML.
- By default the nodes are bound to its own nodes to reflect the name of the node rather than the attribute or the inner text.
- The node to node association can be controlled by binding parameters.
- The nodes can be bound to a data source field by specifying tree node bindings.
- The TreeNodeBinding object defines the relationship between each data item and the node it is binding.
- The Menu control is an end to end site navigation tool. It can be bound to any data source and also supports explicit list of items for simple cases.
- The menu items are stored in a collection and the MenuItems collection property returns all the child items of a given menu.

- A few static and dynamic methods are supported by this class such as selected item, submenus and mouse over items.

- Dynamic menus are implemented using the Dynamic HTML object model.

Managing Data with ADO.NET

The Role of DataSets in ADO.NET

- The important of datasets can never be denied nor it can be questioned.
- DataSet is a mirror image of the table or the query which you run.

- DataSets makes it easier to edit and update the information.

- Another good thing is that datasets are disconnected in nature so, if you make any changes in the dataset it will not reflect in the database unless use special methods to perform the change and confirm it.

Using DataSets in ASP.NET

- Let's see how we can use datasets in Asp.net.
- The most common way of using the DataSet control is with the dataadapter.

- The DataAdapter fills the dataset control with data coming from the database.

Let's see how we can perform this simple action:

```
SqlDataAdapter ad = new SqlDataAdapter("SELECT * FROm Categories",myConnection);

DataSet ds = new DataSet();

ad.Fill(ds,"Categories");

DataGrid1.DataSource = ds;

DataGrid1.DataBind();
```

Explanation of the code:

1. First we declare a simple dataadapter instance which fetches all the rows from the Categories table. (Categories table is present in the Northwind database which is shipped with Sql Server 2000).

2. New we created an instance of the DataSet control.

3. Next we fill the dataset with the data coming from the dataadapter control. As you can see that I have written ad.Fill(ds,"Categories"); you can also write ad.Fill(ds); and this will work fine too. In the former case I was telling the dataadapter which table I am using in my query and later the dataadapter was finding the table itself.

4. Finally I used DataGrid to bind the data on the screen.

Iterating through the DataSet:

Here is a simple code to iterate through the dataset and select individual items instead of selecting all the data in the DataSet.

```
SqlDataAdapter ad = new SqlDataAdapter("SELECT * FROm Categories",myConnection);

DataSet ds = new DataSet();

ad.Fill(ds,"Categories");

DataTable dt = ds.Tables["Categories"];

Response.Write(dt.Rows[0][1].ToString()) ;
```

1) First few lines are identical which we have done before.

2) In the forth line we declared the DataTable object. DataTable object is used to hold a single table. As I already told you that dataset can hold multiple tables depending upon the query so we can assign a single table from the dataset to the datatable object. I have done that using the line.

```
DataTable dt = ds.Tables["Categories"];
```

3) Next we are printing the first row of the second column. Always remember that the number of

the rows and columns in the database does not start with 1 but it starts at 0. So in this line of code:

Response.Write(dt.Rows[0][1].ToString()) ;

This line of code means that get Row '0' and Column 1 of the from the datatable. And so we get the value and it prints out "Beverages".

- **Saving dataSets in Session State:**

  DataSets can also be saved in the Session State so that it can be available in other areas of the application and in new pages.
- The way of storing the DataSet in Session State is very easy.

  Session["MyDataSet"] = DataSet;

- Later if you want to retrieve the DataSet from the Session State you can easily do this by using this line of code:

  DataSet ds = (DataSet) Session["MyDataSet"]

- As you can see that when I am retrieving the values from the Session object I am casting it into the dataset objec.

- This is because this is explicit conversion and requires casting. I am unboxing in this case.

Using DataTable:

- DataTable is also a collection which you can use like dataset.
- The difference is that DataTable represents only one table.

- Usually its used when you don't have a database and want to save something in the collection.

- // Create a new DataTable.
  System.Data.DataTable myDataTable = new DataTable("ParentTable");
  // Declare variables for DataColumn and DataRow objects.

```
DataColumn myDataColumn;
DataRow myDataRow;
// Create new DataColumn, set DataType, ColumnName and add to DataTable.
myDataColumn = new DataColumn();
myDataColumn.DataType = System.Type.GetType("System.Int32");
myDataColumn.ColumnName = "id";
myDataColumn.ReadOnly = true;
myDataColumn.Unique = true;
// Add the Column to the DataColumnCollection.
myDataTable.Columns.Add(myDataColumn);
```

DataSet Relations:

- In a relational representation of data, individual tables contain rows that are related to one another using a column or set of columns.
- In the ADO.NET DataSet, the relationship between tables is implemented using a DataRelation.
- When you create a DataRelation, the parent-child relationships of the columns are managed only through the relation.
- The tables and columns are separate entities.
- In the hierarchical representation of data that XML provides, the parent-child relationships are represented by parent elements that contain nested child elements.
- To facilitate the nesting of child objects when a DataSet is synchronized with an XmlDataDocument or written as XML data using WriteXml, the DataRelation exposes a Nested property.
- Setting the Nested property of a DataRelation to true causes the child rows of the relation to be nested within the parent column when written as XML data or synchronized with an XmlDataDocument.

The Nested property of the DataRelation is false, by default. For example, consider the following DataSet:

```
SqlDataAdapter custDA = new SqlDataAdapter("SELECT CustomerID, CompanyName FROM
Customers", nwindConn);
SqlDataAdapter orderDA = new SqlDataAdapter("SELECT OrderID, CustomerID, OrderDate
FROM Orders", nwindConn);
nwindConn.Open();
DataSet custDS = new DataSet("CustomerOrders");
custDA.Fill(custDS, "Customers");
orderDA.Fill(custDS, "Orders");
nwindConn.Close();
DataRelation custOrderRel = custDS.Relations.Add("CustOrders",
custDS.Tables["Customers"].Columns["CustomerID"],
custDS.Tables["Orders"].Columns["CustomerID"]);
```

Part B

1. How can you convert any static HTML control into an HTML-server control?
2. Give name of two techniques which provide front-end error handling.
3. Give the name of two configuration file.
4. What is use of id and name properties of HTML server control?
5. How many types of cookies are available in ASP.NET? Explain each in brief.
6. Write use of <% %> symbols.
7. Write name of base class for Web server controls.
8. Suppose web page contain one textbox control. By looking in .aspx file how will you identify whether it is Html-sever or Web-server control?
9. What data types does the RangeValidator control support?
10. How can you bind an event to method?
11. What are ASP.NET Web Forms?
12. What type of code (server or client) is found in a Code-Behind class?
13. Give the name of control for which post back property by default is enable (true).
14. What is viewstate. How Asp.Net manages?
15. What is master page?

16. What is theme?

17. What is the extension of user control page.?
18. What SiteMapPath in Asp.NET?
19. What is web.config file? Compare with machine.config.
20. What FileUpload Control in ASP.NET?

Part C

1. Explain ASP.NET Page Life Cycle.
2. Explain Delegates with examples.
3. State Advantages & disadvantages of web server controls & html server controls.
4. Difference between asp and asp.net.
5. List all validation controls. Explain Regular Expression & Compare validation controls in details.
6. What is Data Binding? Which are 2 types of data binding? Explain any one in details with examples.
7. Difference between ADO and ADO.Net.
8. Which are the 3 Command class methods? Explain them briefly.
9. Write a code to insert, delete, update, select record from the database using Ado.net Class method (Connection class, command class, etc.).
10. Explain Query String Collection. State its limitation.
11. Explain cookies in detail with Examples.

KARPAGAM ACADEMY OF HIGHER EDUCATION
(Deemed to be University)
(Established Under Section 3 of UGC Act, 1956)
Coimbatore-21

DEPARTMENT OF CS, CA & IT

UNIT- V (Objective Type/ Multiple Choice Questions Each Question Carries One Mark)

.NET PROGRAMMING (16CAP502)

S.No	Question	Option1	Option2	Option3	Option4	Answer
1	DataSourceControl provides _____ properties for configuring caching	Two	Five	Three	Four	Four
2	Which method of command object doesn't return any row?	ExecuteReader	ExecuteNonQuery	ExecuteQuery	ExecuteScalar	ExecuteNonQuery
3	Select the Interface which provides Fast, connected forward-only access to data	IdataRecord	Idatabase	IdataReader	Irecorder	IdataReader
4	Which architecture does Datasets follow?	Parallel	disconnected	Connection-oriented	Distributed	disconnected
5	Data source controls do not render any _____markup to the client.	XML	javascript	vbscript	HTML	HTML

6	Where do we store connection string in ASP.NET?	Web.config	App.config	Global.asax	Console.config	Web.config
7	Which of the following is not a member of ConnectionObject?	EndTransaction	BeginTransaction	Open	Execute	Execute
8	The Command object in ADO.NET executes a _____against the database and retrieves a DataReader or DataSet Object	Function	Command	Subroutine	Object	Command
9	Which of these data source controls do not implement Caching?	LinqDataSource	ObjectDataSource	SqlDataSource	XmlDataSource	LinqDataSource
10	Which method do you invoke on the DataAdapter control to load your generated dataset with data?	Load ( )	Fill( )	DataList	DataBind	Fill( )
11	_____ method copies the structure of the DataSet including all DataTable Schemas, relations, constraints and does not copy any data.	Copy	CopyAll	Clone	Finalize	Clone
12	DataReaderObject is suitable for_____ access such populating a list and then breaking the connection.	Write-only	Read-Write	Read-Write-Execute	Read-only	Read-only
13	The_____ property sets the number of seconds that the cache remains valid.	CacheExpirationPolicy	CacheDuration	CacheKeyDependency	EnableCaching	CacheDuration
14	Which objects is used to create foreign key between tables?	DataRelation	DataRelationship	DataConstraint	Datakey	DataRelation
15	ADO.Net provides the ability to create and process in-memory databases called	views	relations	datasets	tables	datasets
16	_____object can hold more than one rowset from the same data source and the relationships between them	DataReader object	Dataset object	OleDB connection object	Data Adapter	Dataset object
17	EnableCaching is a _____property that determines whether or not caching is enabled for the data source control	Character	Double	String	Boolean	Boolean
18	The _____ method of the RadioButtonList control binds the data source with the RadioButtonList control.	DataSource	DataBind( )	DataMember	DataView	DataBind( )
19	DropDownList control is used to give a _____ select option to the user from multiple listed items	multiple	Two	single	four	single

20	Method name that fires when user changes the selection of the dropdown box	AppendDataBoundItems	AutoPostBack	SelectedItem	OnSelectedIndex Changed	OnSelectedIndexChan ged
21	Choose the form in which Postback occur	HTMLForms	Webforms	Winforms	Double forms	Webforms
22	Which of the following object is not an ASP component?	LinkCounter	Counter	AdRotator	File Access	LinkCounter
23	The first event triggers in an aspx page is.	Page_Init()	Page_Load()	Page_click()	page_char()	Page_Init()
24	Which of the following method must be overridden in a custom control?	The Paint() method	The Control_Build() method	The default constructor	The Render() method	The Render() method
25	Which of the following tool is used to manage the GAC?	RegSvr.exe	GacUtil.exe	GacSvr32.exe	GacMgr.exe	GacUtil.exe
26	We can manage states in asp.net application using	Session Objects	Objects	Viewobject	.ASP	Session Objects
27	Attribute must be set on a validator control for the validation to work.	ControlToValidate	ControlToBind	ValidateControl	Validate	ControlToValidate
28	Caching type supported by ASP.Net	Output Caching	DataCaching	a and b	input caching	a and b
29	File extension used for ASP.NET files.	Web	.ASP	ASPX	ASP.X	ASP
30	An alternative way of displaying text on web page using	asp:label	asp:listitem	asp:button	asp:list	asp:label
31	Which of the following is not a member of ADODBCommand object	ExecuteScalar	ExecuteStream	Open	ExecuteReader	Open
32	Which DLL translate XML to SQL in IIS?	SQLISAPI.dll	SQLXML.dll	LISXML.dll	SQLIIS.dll	SQLISAPI.dll
33	Default Session data is stored in ASP.Net.	StateServer	Session Object	InProcess	outProcess	InProcess

34	Default scripting language in ASP.	EcmaScript	VBScript	PERL	JavaScript	VBScript
35	How do you get information from a form that is submitted using the "post" method?	Request.QueryString	Request.Form	Response.write	Response.writeln	Request.Form
36	Which object can help you maintain data across users?	Application object	Session object	Response object	Server object	Application object
37	Which of the following ASP.NET object encapsulates the state of the client?	Session object	Application object	Response object	Server object	Session object
38	Mode of storing ASP.NET session	Proc	Server	SQL Server	oracle Server	SQL Server
39	Which of the following is not the way to maintain state?	View state	Cookies	Hidden fields	Request object	Request object
40	Which of the following is the performance attributes of processModel?	request	maxWorkerThreads	maxIdThreads	QLDataReader	maxWorkerThreads
41	Where do we include the user lists for Form authentication?	< credential>	< authorization>	< Identity>	< authentication>	< credential>
42	_____ tests make sure that new code does not break existing code.	Regression tests	Integration tests	Unit tests	Load test	Integration tests
43	The .NET Framework provides a runtime environment called..... ?	RMT	CLR	RCT	RC	CLR
44	In ASP.NET in form page the object which contains the user name is _____ ?	Page.User.Identity	Page.User.IsInRole	Page.User.Name	Page.Name	Page.User.Identity
45	WSDL stands for _____ ?	Web Server Description Language	Web Server Descriptor Language	. Web Services Description Language	Web Services Descriptor Language	Web Services Description Language
46	In .NET the operation of reading metadata and using its contents is known as _____?	Reflection	Enumeration	Binding	Serialization	Reflection
47	How many classes can a single .NET DLL contain?	. One	Two	None	Many	Many

48	. Suppose a .NET programmer wants to convert an object into a stream of bytes then the process is called _____ ?	Serialization	Threading	RCW	AppDomain	Serialization
49	Which method do you invoke on the DataAdapter control to load your generated dataset with data?	Load ( )	Fill( )	DataList	DataBind	Fill( )
50	Which of the following languages can be used to write server side scripting in ASP.NET?	. C-sharp	VB	C++	A and B	A and B