

# Windows Forms Programming with C#

Are you registered with  
**Onlinevarsity.com?**

Yes



No



Did you download this book  
from **Onlinevarsity.com?**

Yes



No



## Scores

For each **YES** you score **50**

For each **NO** you score **0**

If you score less than 100 this book is illegal.

Register on **[www.onlinevarsity.com](http://www.onlinevarsity.com)**

# **Windows Forms Programming with C#**

© 2014 Aptech Limited

All rights reserved.

No part of this book may be reproduced or copied in any form or by any means – graphic, electronic or mechanical, including photocopying, recording, taping, or storing in information retrieval system or sent or transferred without the prior written permission of copyright owner Aptech Limited.

All trademarks acknowledged.

## **APTECH LIMITED**

Contact E-mail: [ov-support@onlinevarsity.com](mailto:ov-support@onlinevarsity.com)

Edition 1 - 2014



## Dear Learner,

We congratulate you on your decision to pursue an Aptech Worldwide course.

Aptech Ltd. designs its courses using a sound instructional design model – from conceptualization to execution, incorporating the following key aspects:

- Scanning the user system and needs assessment

Needs assessment is carried out to find the educational and training needs of the learner

Technology trends are regularly scanned and tracked by core teams at Aptech Ltd. TAG\* analyzes these on a monthly basis to understand the emerging technology training needs for the Industry.

An annual Industry Recruitment Profile Survey# is conducted during August - October to understand the technologies that Industries would be adapting in the next 2 to 3 years. An analysis of these trends & recruitment needs is then carried out to understand the skill requirements for different roles & career opportunities.

The skill requirements are then mapped with the learner profile (user system) to derive the Learning objectives for the different roles.

- Needs analysis and design of curriculum

The Learning objectives are then analyzed and translated into learning tasks. Each learning task or activity is analyzed in terms of knowledge, skills and attitudes that are required to perform that task. Teachers and domain experts do this jointly. These are then grouped in clusters to form the subjects to be covered by the curriculum.

In addition, the society, the teachers, and the industry expect certain knowledge and skills that are related to abilities such as *learning-to-learn, thinking, adaptability, problem solving, positive attitude etc.* These competencies would cover both cognitive and affective domains.

**A precedence diagram for the subjects is drawn where the prerequisites for each subject are graphically illustrated. The number of levels in this diagram is determined by the duration of the course in terms of number of semesters etc. Using the precedence diagram and the time duration for each subject, the curriculum is organized.**

- Design & development of instructional materials

The content outlines are developed by including additional topics that are required for the completion of the domain and for the logical development of the competencies identified. Evaluation strategy and scheme is developed for the subject. The topics are arranged/organized in a meaningful sequence.

The detailed instructional material – Training aids, Learner material, reference material, project guidelines, etc.- are then developed. Rigorous quality checks are conducted at every stage.

➤ Strategies for delivery of instruction

Careful consideration is given for the integral development of abilities like thinking, problem solving, learning-to-learn etc. by selecting appropriate instructional strategies (training methodology), instructional activities and instructional materials.

The area of IT is fast changing and nebulous. Hence considerable flexibility is provided in the instructional process by specially including creative activities with group interaction between the students and the trainer. The positive aspects of web based learning –acquiring information, organizing information and acting on the basis of insufficient information are some of the aspects, which are incorporated, in the instructional process.

➤ Assessment of learning

The learning is assessed through different modes – tests, assignments & projects. The assessment system is designed to evaluate the level of knowledge & skills as defined by the learning objectives.

➤ Evaluation of instructional process and instructional materials

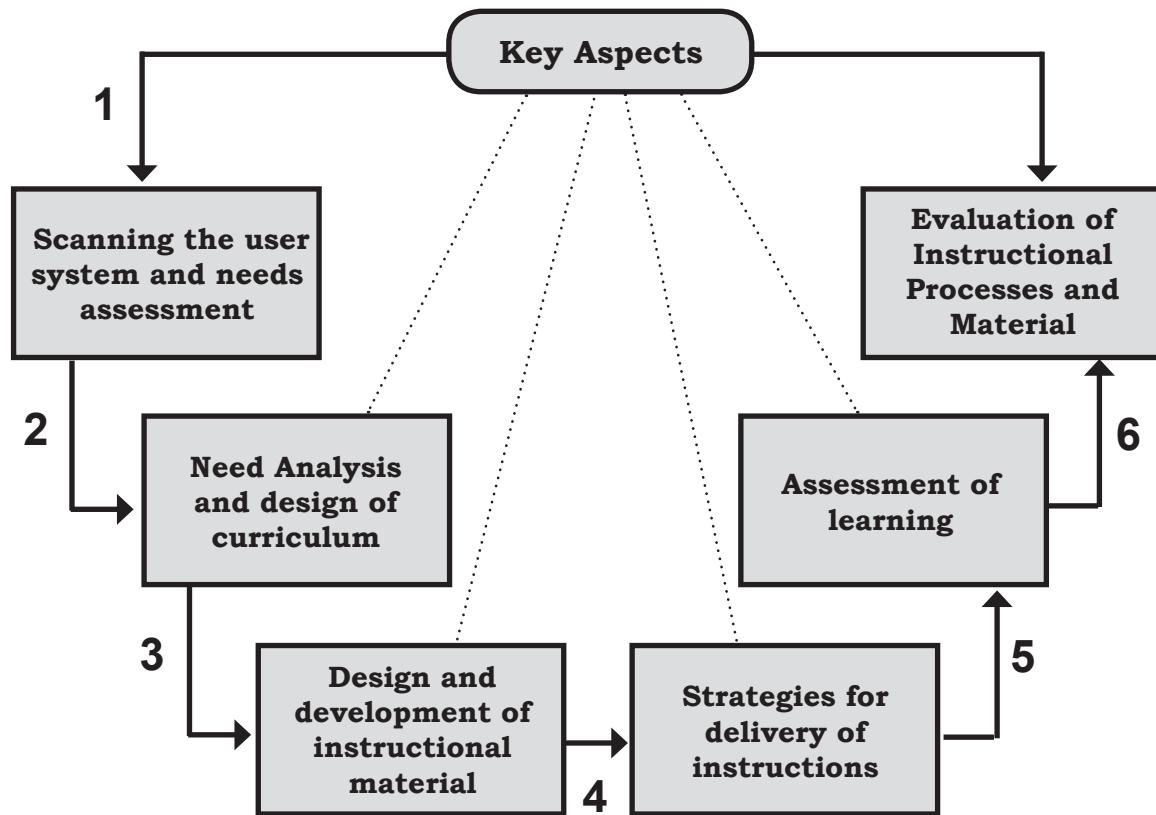
The instructional process is backed by an elaborate monitoring system to evaluate - on-time delivery, understanding of a subject module, ability of the instructor to impart learning. As an integral part of this process, we request you to kindly send us your feedback in the reply pre-paid form appended at the end of each module.

\*TAG – Technology & Academics Group comprises of members from Aptech Ltd., professors from reputed Academic Institutions, Senior Managers from Industry, Technical gurus from Software Majors & representatives from regulatory organizations/forums.

Technology heads of Aptech Ltd. meet on a monthly basis to share and evaluate the technology trends. The group interfaces with the representatives of the TAG thrice a year to review and validate the technology and academic directions and endeavors of Aptech Ltd.

Industry Recruitment Profile Survey - The Industry Recruitment Profile Survey was conducted across 1581 companies in August/September 2000, representing the Software, Manufacturing, Process Industry, Insurance, Finance & Service Sectors.

### Aptech New Products Design Model





are just a  
*click away*

To chat with a

**tutor**

*Login to*  **www.onlinevarsity.com**

---

## Preface

---

Windows Forms is an extensive set of class libraries that can be used to develop desktop and client server based Windows applications. Windows Forms, provided by the .NET Framework, is used to design and develop Windows applications using any of the languages supported by the .NET Framework. This book aims at making the students comfortable with the IDE. In this module, the students will build applications using C# and Windows Forms. It is to be noted that this module is based on Visual Studio 2005, 2008, and .NET Framework 2.0, 3.5.

Windows Forms in Visual Studio 2008 helps you design and develop powerful applications faster and in a more efficient manner. Features and controls of Windows Forms such as nonrectangular Windows Forms, TabControl and TrackBar controls, NotifyIcon component, ToolStripItems, and so forth are described. Further, the WPF technology, its structure and various controls are explained. Two of the most important and beneficial new features in Visual Studio 2008 are LINQ and the ADO.NET Entity Framework. These two features are explained in detail. Microsoft facilitated deployment of Windows-based applications through a new technology called ClickOnce introduced in Visual Studio 2005. In Visual Studio 2008, both ClickOnce and Windows Installer deployment have been enhanced and these enhancements are also explored in this book.

This book is the result of a concentrated effort of the Design Team, which is continuously striving to bring you the best and the latest in Information Technology. The process of design has been a part of the ISO 9001 certification for Aptech-IT Division, Education Support Services. As part of Aptech's quality drive, this team does intensive research and curriculum enrichment to keep it in line with industry trends.

We will be glad to receive your suggestions. Please send us your feedback, addressed to the Design Centre at Aptech's corporate office, Mumbai.

Design Team

WRITE-UPS BY

**EXPERTS AND LEARNERS**

TO PROMOTE NEW AVENUES AND  
ENHANCE THE LEARNING EXPERIENCE



FOR FURTHER READING, LOGIN TO

**[www.onlinevarsity.com](http://www.onlinevarsity.com)**

## Table of Contents

### Sessions

1. Windows Forms and Basic Controls
2. Grouping and Graphic Controls
3. Advanced Controls
4. Date Controls and Timer Component
5. Dialog Boxes
6. MDI Applications and Menus
7. Introduction to ADO.NET
8. Data Classes
9. DataView and DataGridView Controls
10. Data Binding
11. Working with GDI+
12. Custom Controls
13. Crystal Reports
14. Printing and Adding Help
15. Packaging and Deployment
16. Exploring Windows Forms
17. Exploring Windows Forms (Lab)
18. Windows Presentation Foundation
19. Windows Presentation Foundation (Lab)
20. LINQ and ADO.NET Entity Framework
21. LINQ and ADO.NET Entity Framework (Lab)
22. Configuring and Deploying Applications
23. Configuring and Deploying Applications (Lab)

Answers to Knowledge Check



# Balanced Learner-Oriented Guide

**for enriched learning available**



[www.onlinevarsity.com](http://www.onlinevarsity.com)

# Module - 1

## Windows Forms and Basic Controls

Welcome to the Module, **Windows Forms and Basic Controls**.

This module provides an overview of Windows Forms and the basic controls used to create graphical user interfaces. A Windows Form is a window used to hold various controls. These controls are graphical objects that facilitate user interaction to accept data or to display information. Controls are classified into different categories based on the functionality they provide. This module covers the basic controls category, which include controls such as Labels, TextBoxes, and Buttons.

In this Module, you will learn about:

- Windows Forms
- Form Class
- Types of Controls
- Basic Controls

## 1.1 Windows Forms

In this first lesson, **Windows Forms**, you will learn to:

- Describe Windows Forms and its role in .NET development.
- State the features of Windows Forms 2.0.

### 1.1.1 Need for Windows Forms

Consider a scenario of an IT firm where the accountant is maintaining the salary details of employees in an excel sheet. The accountant finds it difficult and tedious to maintain such huge data. In addition, frequent manual errors were found such as typing incorrect data in the fields, making wrong mathematical calculations by missing the cell address in the formula, and so on. Therefore, the firm has decided to automate the process by creating a secured application that can be used only by the authorized employees.

To overcome such problems and to meet the requirements of the firm, the .NET Framework provides you with the option of creating user-friendly applications using Windows Forms.

### 1.1.2 Windows Forms

Windows Forms allows the developer to create user interfaces using various built-in components. These components are controls that allow you to take information and view the requested information through the form. This makes the application user-friendly. After designing the interface using the components, the developers can use any language supported by the .NET Framework to provide the required functionalities. Therefore, you can create powerful Windows-based applications with Windows Forms.

The applications created using Windows Forms are executed on the local machine. The form can access the resources of the local machine such as memory, printer, folders, and files. Therefore, they are best suited for creating desktop applications and managing them privately. The role of Windows Forms include:

- Holding controls
- Handling user input
- Displaying information
- Connecting to database, which might exists on another machine

### 1.1.3 Windows Forms in .NET Framework

Windows Forms are a part of the .NET Framework that is a multi-lingual and multi-platform environment to build, deploy, and run applications. Windows Forms provide with a standard programming model that allows you to create applications with least bugs and inconsistencies.

In the .NET Framework, the forms are executed in the .NET CLR (Common Language Runtime) environment. The .NET Framework provides CLR environment, which manages all the programs in .NET and also provides services for development process.

Figure 1.1 displays the Windows Forms in .NET Framework.

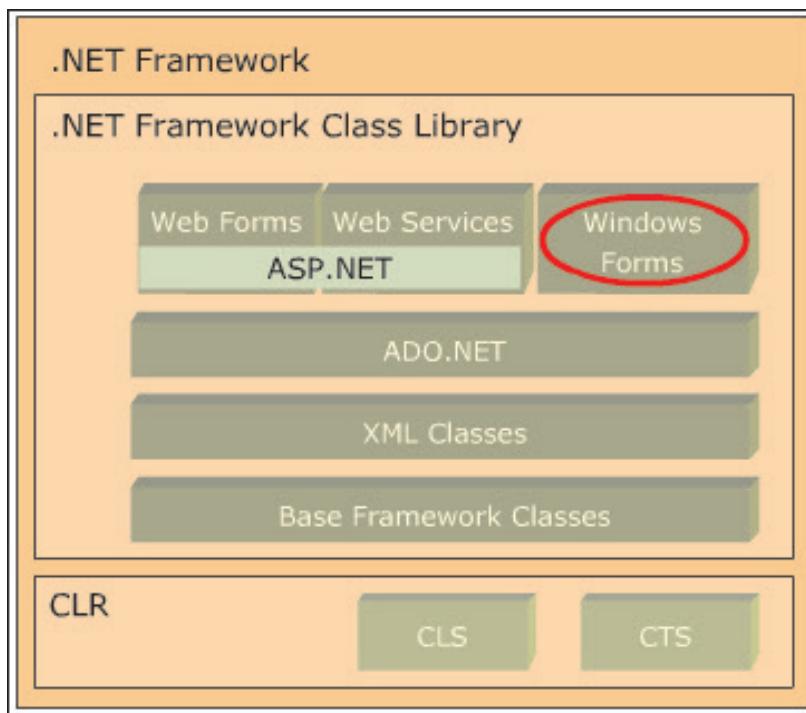


Figure 1.1: Windows Forms in .NET Framework

### 1.1.4 Features of Windows Forms

Windows Forms include many powerful features for creating, designing, and implementing Windows-based applications. Some of the features are as follows:

#### → ClickOnce Deployment

The ClickOnce technology is used to create applications that can be installed with least user interaction. In case of re-installing the application, only those parts of the application are updated that are changed, saving time and effort.

#### → Application Settings

The application settings feature allows you to store global data and some user-related settings

that can be reused in the application. For example, you can store database connection details such as database name and login details and user-specific settings such as color, font, and the access permissions of the form.

#### → New Windows Forms Controls

Windows Forms contains a variety of new controls, which you can use to create different applications. These new controls include ToolStrip, MaskedTextBox, SplitContainer, ListView, WebBrowser, ContextMenuStrip, MenuStrip, StatusStrip, ToolStripContainer and so on.

#### → New Data Binding Model

The data binding feature of Windows Forms provides a simple, convenient, and powerful way to link controls on a form and a target data source to store and fetch data. The new data binding model provides with the BindingSource component that simplifies the task of saving and fetching data.

#### → Rich Graphics

Windows Forms supports an advanced version of the Windows Graphics Device Interface (GDI) called GDI+. GDI is used for drawing and painting images on forms. GDI+ provides rich graphic support such as alpha blending to provide color effects and texture brushes using which you can use a bitmap to paint the image.

**Note** - One more feature of Windows Forms is the SnapLine feature. This feature provides horizontal and vertical lines, which appear automatically on the form to position the controls. These lines help in arranging the controls with other controls on forms.

### Knowledge Check 1

- Which of these statements about Windows Forms are true and which statements are false?

(A)	Windows Forms execute on a remote machine.
(B)	Windows Forms does not support GDI+.
(C)	The application settings feature allows storing only user-defined settings.
(D)	The ToolStripContainer control is a new control introduced in Windows Forms.
(E)	Windows Forms can access a database, which exists on another machine.

- Can you match the features of Windows Forms 2.0 against their corresponding descriptions?

	Description		Feature
(A)	Creates application that can be installed with less user interaction.	(1)	Data Binding Model
(B)	Stores user specific data.	(2)	Application Settings

Description		Feature
(C)	Supports alpha blending and texture brushes.	(3) Windows Forms Controls
(D)	Provides the <code>BindingSource</code> component.	(4) ClickOnce Deployment
(E)	Provides the <code>MaskedTextBox</code> and <code>WebBrowser</code> controls.	(5) Rich Graphics

## 1.2 Form Class

In this second lesson, `Form` Class, you will learn to:

- Explain the `Form` class.
- List and describe the properties of `Form` class.
- List and describe the methods of `Form` class.
- Explain the commonly used events of `Form` class.

### 1.2.1 Form Class

The basic unit of an application is a form. A form is a container that holds and manages controls and is used to create GUI based Windows applications. A Windows Forms application will have at least one form (window), which would be used for the interaction between the application and user. Huge applications consist of multiple forms, which can be navigated through certain controls.

You can create a form in an application by creating an object of the `Form` class. The class exists in the `System.Windows.Forms` namespace. The classes defined in this namespace use the features for creating user-friendly interfaces provided by the Windows operating system.

**Note** - Windows Forms are also known as WinForms.

### 1.2.2 Properties

The properties of the `Form` class allow you to modify the appearance of the form such as the size, color, and location. Table 1.1. lists the most commonly used properties of the `Form` class.

Property	Description
<code>AcceptButton</code>	Specifies or retrieves the button on the form that is clicked when the user presses the <b>ENTER</b> key.
<code>ForeColor</code>	Specifies or retrieves the foreground color.
<code>Location</code>	Specifies or retrieves the co-ordinates of the upper left corner of the form.
<code>Modal</code>	Specifies or retrieves a value indicating whether the form is shown modally. This means if the user tries to activate another window or form, the user cannot do so without closing the current modal form.

Property	Description
Name	Specifies or retrieves the name of the form.
Text	Specifies or retrieves the text to be displayed on the title bar of the form.
WindowState	Specifies or retrieves a value indicating whether a form is minimized, maximized, or is in the normal state. The window state can be changed using the <code>FormWindowState</code> enumeration.

Table 1.1: Properties of the Form Class

The following snippet demonstrates how to use the properties of `Form` class.

#### Code Snippet:

```
Form frmStudent = new Form();
frmStudent.Text = "Student Details";
frmStudent.ForeColor = System.Drawing.Color.Brown;
frmStudent.WindowState = System.Windows.Forms.FormWindowState.Maximized;
```

The code creates an object of the `Form` class. The `Text` property is used to display the text `Student Details` on the title bar of the form. The `ForeColor` property is used to set the color of the text on the controls to Brown. Brown is the static property of the `Color` structure defined in the `System.Drawing` namespace. The `WindowState` property is used to specify the state of the form as `Maximized`.

### 1.2.3 Methods

The methods of `Form` class are used to change the behavior of the form. Table 1.2 lists the most commonly used methods of the `Form` class.

Method	Description
Activate	Activates a form and gives it focus.
Close	Closes a form.
Focus	Puts the focus on a control.
Hide	Makes a form invisible to the user.
Show	Displays a form to the user.

Table 1.2: Methods of the Form Class

The following snippet demonstrates how to use the methods of the `Form` class.

#### Code Snippet:

```
Form frmStudent = new Form();
frmStudent.Show();
frmStudent.Hide();
```

The code creates an object of the `Form` class. The `Show()` method is invoked to display the form to the user. The `Hide()` method is used to hide a form from the user.

### 1.2.4 Events

Consider a group of people at a picnic playing Bingo. When a number is read, the participants verify if the number is present on their cards. Here, the calling of the number is referred to as the occurrence of an event from the view point of a developer. Here, the participants are paying attention (subscribing) to what the announcer (the source of the event) has to say.

Similarly, in Windows Forms programming with C#, an event allow an object (source of the event) to inform other objects (subscribers) about the event. The most common example is clicking the `Button`, where click is the event.

The events of the `Form` class allow the form to respond to the actions performed by the user on the form. To handle events, event handlers are defined, which are methods invoked when the event is raised. Table 1.3 lists the most commonly used events of the `Form` class.

Event	Description
Activated	Occurs when the form is activated in code or by the user.
Click	Occurs when a control is clicked.
Deactivate	Occurs when the focus on the form is taken away in code or by the user.
FormClosed	Occurs after the form is closed.
FormClosing	Occurs before the form is closed.
GotFocus	Occurs when the focus is put on a control of the form.
Load	Occurs when the form is displayed for the first time.
Resize	Occurs when a control on the form is resized.
Validated	Occurs when validation is completed by a control. For example, the event is raised after validating the login details in the <code>TextBox</code> control.
Validating	Occurs when the validating process is being carried out by the control. For example, validating the login details in the <code>TextBox</code> control.

Table 1.3: Events of the Form Class

The following snippet demonstrates how to use the events of the `Form` class.

#### Code Snippet:

```
private void frmStudent_Activated(object sender, EventArgs e)
{
    frmStudent.BackColor = System.Drawing.Color.Coral;
}
private void frmStudent_FormClosed(object sender, FormClosedEventArgs e)
{
    MessageBox.Show("Closing the form");
}
```

The code defines two events handlers for the `Activated` and `FormClosed` events. When the form is activated by the user, the `frmStudent_Activated` event handler is invoked and the back color of the form is set to Coral. When the form is closed, the `frmStudent_FormClosed` event handler is invoked

and a message box is displayed stating that the form is closing.

The life of a Windows Form starts when you create the object of the `Form` class. After coming into existence, the form faces various state changes based on the interaction with the user. This leads to a sequence of events taking place, which measures the life of the form. These events are used to manage and control the behavior of the form.

#### → Construction

The constructor of the `Form` class is invoked to construct a form when the form is executed. This constructor initializes a new object of the `Form` class, which invokes the `InitializeComponent()` method to create and initialize all the controls.

#### → Load

The `Load` event takes place after the form is constructed. The `Load` event occurs when the form is displayed for the first time. This event is useful for initializing controls by writing an appropriate code in the `Load` event handler.

#### → Activated

The `Activated` event occurs when the form is loaded for the first time.

#### → Shown

The `Shown` event occurs after the `Activated` event and indicates that the form is displayed to the user.

#### → Deactivated

The `Deactivate` event occurs when the user switches away from the form and concentrates on other form.

#### → Activated

The `Activated` event is invoked to activate the form when the user again switches back to the form, which has been deactivated.

#### → FormClosing

The `FormClosing` event arises when the form is in its closing state.

#### → FormClosed

The `FormClosed` event arises when the form is closed.

→ **Disposed**

The `Disposed` event occurs when the form is closed and it removes the object of the `Form` class from the memory.

**Note** - The `InitializeComponent()` method is used to initialize the controls on the form when the form is executed.

## Knowledge Check 2

1. Can you arrange the sequence of events in the lifecycle of a form?

(A)	Activated
(B)	FormClosing
(C)	Construction
(D)	FormClosed
(E)	Load

2. Can you match the methods, properties, and events of the `Form` class against their corresponding descriptions?

Description		Property, Method, and Event
(A)	Indicates whether a form can be displayed as modal.	(1) Load
(B)	Displays the form after the form is constructed.	(2) WindowState
(C)	Indicates that the user has focused back to the form.	(3) Modal
(D)	Displays the form as an active window.	(4) Activated
(E)	Indicates whether the form is minimized or maximized.	(5) Activate

## 1.3 Types of Controls

In this third lesson, **Types of Controls**, you will learn to:

- Identify the types of controls used in Windows Forms applications.
- Explain the `Control` class.

### 1.3.1 Use of Controls

A control is a visual object that is added on a form to create graphical user interfaces for an application. The purpose of a control is to display information or to take the information from the user. For example, a label is used to display information and a text box is used to take information from the user. Thus, controls are placed on the forms in an organized manner to facilitate user interaction in applications

using Windows Forms.

Microsoft Visual Studio 2005 provides different types of controls for Windows Forms programming.

A control is associated with its respective built-in class defined in the `System.Windows.Forms` namespace. For example, the `Label` control is associated with the `Label` class. Each control class consists of various properties, methods, and events; which allow you to identify the function of the respective control. Based on the functionality provided by the various controls, they are classified into different categories. These categories are as follows:

→ **Basic**

These controls are the commonly used controls such as `Labels` and `Buttons`.

→ **Value Setting**

These controls allow you to select an option or multiple options from the given set of options. For example, `RadioButton` and `Checkboxes` are value setting controls.

→ **Selection List**

These controls allow you to select a value from a given list using the up and down arrows similar to a spin control.

→ **Grouping**

These controls allow you to group different controls such as `Labels`, `TextBoxes`, and `Buttons` in a group to display or take specific information from the user.

→ **Graphics**

These controls allow you to display images on a form.

→ **Menu**

These controls allow you to create menus and shortcut menus on the form, similar to the menus and shortcut menus that appear in any Windows application, such as Microsoft Word and Microsoft Excel.

→ **Date Setting**

These controls allow you to select or display a particular date and time.

### 1.3.2 Control Class

The `Control` class is the base class of all the controls available for Windows Forms programming. It is used to customize the appearance of the controls and handle the user input received through controls. The class is defined in `System.Windows.Forms` namespace and defines various properties, methods, and events.

The properties and methods of a form can be read or set using the `this` keyword. The `this` keyword is used to represent the currently active form.

The `Control` class defines common properties, methods, and events; which are inherited by the respective classes of the controls.

The `Control` class defines common properties, methods, and events; which are inherited by the respective classes of the controls.

#### → Properties

The properties of the `Control` class handle the appearance and layout of the controls. Table 1.4 lists the commonly used properties of the `Control` class.

Property	Description
<code>CanFocus</code>	Specifies or retrieves a value that identifies whether the control can receive focus.
<code>Controls</code>	Retrieves a collection of controls contained within the control.
<code>Enabled</code>	Specifies or retrieves a value that indicates whether the control can react to an action performed by the user.
<code>Name</code>	Specifies or retrieves the name of a control.
<code>Parent</code>	Specifies or retrieves the parent container of the control.
<code>TabIndex</code>	Specifies or retrieves the tab order of the control within the form. For example, if you press the <b>Tab</b> key multiple times, this property will set the order in which you can navigate through each control on the form.
<code>Text</code>	Specifies or retrieves the text associated with a control.
<code>Visible</code>	Specifies or retrieves a value that identifies whether a control is displayed on the form.

Table 1.4: Properties of the Control Class

#### → Methods

The methods of the `Control` class handle the behavior of the controls. Table 1.5 lists the commonly used methods of the `Control` class.

Method	Description
<code>Focus</code>	Sets input focus to the control.
<code>GetNextControl</code>	Retrieves the next control ahead or in the backward direction according to the specified tab order.
<code>Hide</code>	Hides the control from the user.
<code>IsMnemonic</code>	Identifies a mnemonic character in a given string associated with the control, which can be used with the <b>Alt</b> key to access the control from the keyboard.
<code>Select</code>	Activates a control.

Method	Description
Show	Displays the control.
SelectNextControl	Activates the next control.

Table 1.5: Methods of the Control Class

→ **Events**

The events of the `Control` class handle the response of the controls against user interaction. Table 1.6 lists the commonly used events of the `Control` class.

Event	Description
Click	Occurs when a control is clicked.
ControlAdded	Occurs when a new control is added to the <code>Control.ControlCollection</code> class, which is used to keep the track of total controls.
DoubleClick	Occurs when the control is double-clicked.
GotFocus	Occurs when the focus is on the control.
KeyPress	Occurs when a key is pressed while the control has focus.
Leave	Occurs when the input focus leaves the control.
LostFocus	Occurs when the control loses focus.
MouseClick	Occurs when the user clicks the control by the mouse.
Move	Occurs when the user moves the control.
TextChanged	Occurs when the value of the <code>Text</code> property changes.
Validating	Occurs when the validating process is in continuation by the control. For example, the validation could be on the login details, where the data in the <code>TextBox</code> controls is being validated.
Validated	Occurs when the validation is done by the control. For example, when the login details in the <code>TextBox</code> controls are validated, this event is raised.

Table 1.6: Events of the Control Class

`Controls` is a collection of controls present on a form. The `ControlCollection` class defined within the `Control` class and is used to add, remove, and copy controls. The `Add()` method is used to add controls on the form. For example, `this.Controls.Add(lblName)`, will add a `Label` control named, `lblName` to the form.

### Knowledge Check 3

1. Which of these statements about controls and `Control` class are true and which statements are false?

(A)	The basic controls are controls that include the <code>RadioButton</code> and <code>CheckBox</code> controls.
(B)	The selection list controls are controls that allow you to select a value from the list using the down arrows only.
(C)	The <code>Control</code> class is the parent class of all graphic controls.

(D)	The <code>Control</code> class is used to create custom controls.
(E)	The <code>Visible</code> property of the <code>Control</code> class is used to activate the control.

2. Can you match the different types of controls against their corresponding descriptions?

Description		Control Type
(A)	Controls allow you to use the most commonly used controls.	(1) Selection List controls
(B)	Controls allow you to choose the required options from the given set of options.	(2) Grouping controls
(C)	Controls allow you to select values from a list using the up and down arrows.	(3) Value Setting controls
(D)	Controls allow you to organize controls in a group.	(4) Menu controls
(E)	Controls allow you to create menus.	(5) Basic controls

3. Can you match the properties, methods, and events of the `Control` class against their corresponding descriptions?

Description		Property, Method, and Event
(A)	Method used to specify the next control to be navigated according to the tab order.	(1) <code>Focused</code>
(B)	Property to determine for any mnemonic characters associated with the control.	(2) <code>IsMnemonic</code>
(C)	Event that is invoked when the focus is received by the control.	(3) <code>Focus</code>
(D)	Property to retrieve a value indicating whether the control is focused.	(4) <code>SetNextControl</code>
(E)	Method to place the focus on the control.	(5) <code>GotFocus</code>

## 1.4 Basic Controls

In this fourth lesson, **Basic Controls**, you will learn to:

- List the commonly used controls.
- Describe the `Label` control.
- State and describe the properties, methods, and events of `TextBox` and `MaskedTextBox` controls.
- Identify and describe the properties, methods, and events of `Button` control.
- List and describe the properties, methods, and events of `ListBox` and `ComboBox` control.
- List and describe the properties, methods, and events of `LinkLabel` control.

### 1.4.1 Common Controls

A form contains controls such as `TextBox`, `Label`, `Button`, and so on. All these controls are associated with a built-in class defined in the .NET class library. Controls are classified into different categories according to the functions they provide. One such category is of the basic controls, which provide the basic functionality of taking input from the user and displaying the requested information.

You are aware of most of these common controls while working with the Windows operating system. The commonly used controls are as follows:

- ➔ `Label`
- ➔ `TextBox`
- ➔ `Button`
- ➔ `ListBox`
- ➔ `ComboBox`
- ➔ `LinkLabel`

### 1.4.2 Label Control

The `Label` control is a control that is used to show detailed information, which cannot be modified by the user. The most common purpose of the `Label` control is to provide captions to the controls. For example, you can create a `Label` control captioned 'Student Name' to accept the name of the student in corresponding `TextBox` control of the `Label` control. The `Label` control informs the user to enter the name of the student in the corresponding `TextBox` control.

You have come across the `Label` control many times while working with Windows applications. For example, in Microsoft Word application, the Print dialog box consists of various labels that inform you about the various printing options. The image shown highlights the `Label` control in the Print dialog box.

**Note** - Controls used for Windows Forms programming can be created while designing the form. This way of creating controls is referred to as design-time creation of controls. During design-time, you just need to drag the required control from the `ToolBox` on to the form.

An instance of the 'Label' class is created when a `Label` control is added to the form. The following syntax is used to create the `Label` control.

#### Syntax:

```
Label <objectName> = new Label();
```

where,

`Label`: Is the built-in class used to create the `Label` control.

`objectName`: Is the object name of the `Label` class.

`new`: Is the keyword used to allocate memory to the object of the `Label` class and invokes the constructor of the `Label` class.

The following code is used to create the `Label` control, named `lblStudentName`.

#### Code Snippet:

```
Label lblStudentName = new Label();
```

The `Label` class is a built-in class used to create the `Label` control and provides various properties, methods, and events.

#### → Properties

The properties of the `Label` class allow you to specify certain properties such as caption, alignment, and access key. Table 1.7 lists the commonly used properties of the `Label` class.

Property	Description
Name	Specifies or retrieves the name of the control.
Text	Specifies or retrieves the text on the control.
TextAlign	Specifies or retrieves the position of the text on the control.
UseMnemonic	Specifies or retrieves a value indicating whether an ampersand character (&) exists in the <code>Text</code> property. If the value is true, the first character after the ampersand will be used as the label's mnemonic key. This means that the label will be accessed from the keyboard using the <code>Alt + &lt;Mnemonic key&gt;</code> .

Table 1.7: Properties of the Label Class

#### → Methods

The methods of the `Label` class allow you to show and hide the `Label` control. Table 1.8 lists the commonly used methods of the `Label` class.

Method	Description
Contains	Retrieves a value indicating whether the given control is the child of the <code>Label</code> control.
Hide	Makes the control invisible to the user.
Show	Exhibits the control on the form.

Table 1.8: Methods of the Label Class

#### → Events

Table 1.9 lists the commonly used event of the `Label` class.

Event	Description
Click	Takes place when the control is clicked.

Table 1.9: Events of the Label Class

The following code demonstrates the properties and methods of the `Label` class.

**Code Snippet:**

```
Label lblFirstName = new Label();  
lblFirstName.Text = "&First Name:";  
lblFirstName.UseMnemonic = true;  
this.Controls.Add(lblFirstName);  
lblFirstName.Hide();
```

The code creates a `Label` control using the `Label` class. The caption on the `Label` control is set to `&First Name`, making '`F`' the mnemonic key. The `UseMnemonic` property is set to `true`, which means that you can press `Alt + F` keys to access the `Label` control. The control is added to the form using the `Add()` method. The `Hide()` method is used to hide the control from the user.

### 1.4.3 TextBox and MaskedTextBox Controls

The `TextBox` control takes the required information from the user, which can be modified. For example, you can create a `TextBox` control to accept the name of the student.

The `MaskedTextBox` control is a new control used to validate user input. It is an extension of the `TextBox` control and can perform validations for checking:

- ➔ Total input characters
- ➔ Numeric and alphabetical characters
- ➔ Casing
- ➔ Special characters such as '/' and '-'

When a particular validation is applied using the `MaskedTextBox` control, the control shows the mask of required number of characters. For example, if you apply validation for accepting a date, the masked text in the control will be `__ / __ / __`, indicating the month, date, and the year.

Figure 1.2 displays the TextBox and MaskedTextBox controls.

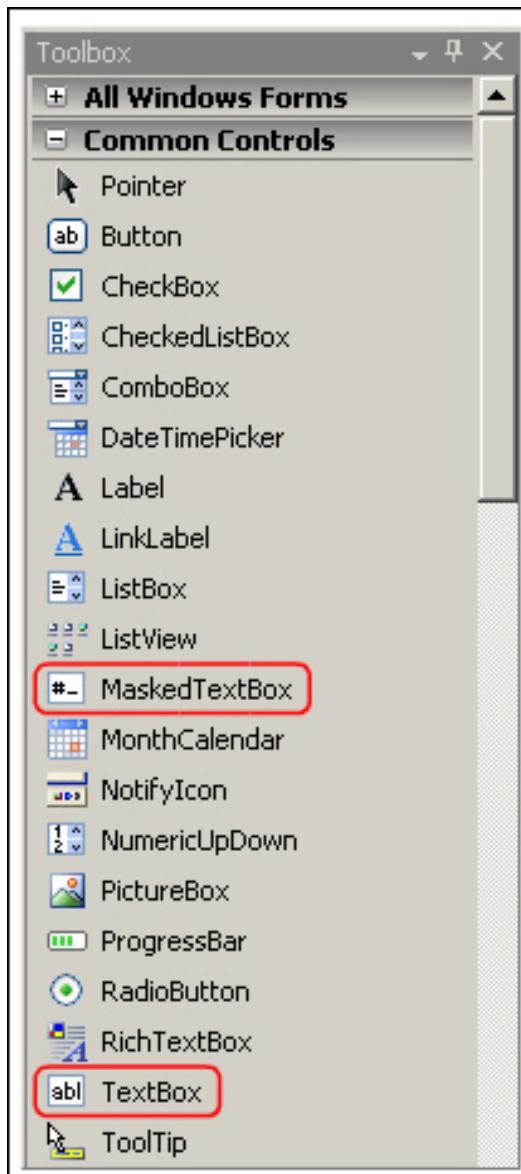


Figure 1.2: TextBox and MaskedTextBox Controls

The following syntax is used to create the TextBox and MaskedTextBox controls.

#### Syntax:

```
TextBox <objectName> = new TextBox();  
  
MaskedTextBox <objectName> = new MaskedTextBox();
```

where,

**TextBox:** Is the built-in class used to create the TextBox control.

**MaskedTextBox:** Is the built-in class used to create the MaskedTextBox control.

The following code is used to create the `TextBox` and `MaskedTextBox` controls.

**Code Snippet:**

```
TextBox txtStudentName = new TextBox();
MaskedTextBox msktxtPhoneNumber = new MaskedTextBox();
```

The `TextBox` class is a built-in class used to create the `TextBox` control and it provides various properties, methods, and events.

→ **Properties**

The properties of the `TextBox` class allow you to specify casing, length, and password character. Table 1.10 lists the commonly used properties of the `TextBox` class.

Property	Description
CharacterCasing	Specifies whether the characters should be converted to uppercase or lowercase.
MaxLength	Specifies or retrieves the maximum number of characters that the user can enter in the control. This property is inherited from the <code>TextBoxBase</code> class.
MultiLine	Specifies or retrieves a value indicating whether the user can enter multi-line text.
Name	Specifies or retrieves the name of the control.
PasswordChar	Specifies or retrieves the special character used to mask characters of a password.
ReadOnly	Specifies or retrieves a value whether the text in the control is editable. This property is inherited from the <code>TextBoxBase</code> class.
Text	Specifies or retrieves the text on the control.

Table 1.10: Properties of the `TextBox` Class

→ **Methods**

The methods of the `TextBox` class allow you to append text, remove, and copy text. Table 1.11 lists the commonly used methods of the `TextBox` class.

Method	Description
AppendText	Adds text to the existing text of the control and this method is inherited from the <code>TextBoxBase</code> class.
Clear	Removes text from the control and this method is inherited from the <code>TextBoxBase</code> class.
Copy	Copies the selected text in the control to the <b>Clipboard</b> . This method is inherited from the <code>TextBoxBase</code> class.
Focus	Sets the input focus on the control to enter text.
Paste	Replaces the selected text in the control with the text copied to the <b>Clipboard</b> .

Table 1.11: Methods of the `TextBox` Class

## → Events

Table 1.12 lists the commonly used events of the `TextBox` class.

Event	Description
KeyPress	Occurs when the user has finished pressing a key. The event handler receives an argument of type <code>KeyPressEventArgs</code> . The <code>KeyChar</code> property of <code>KeyPressEventArgs</code> retrieves the character corresponding to the key pressed. The <code>Handled</code> property of <code>KeyPressEventArgs</code> determines whether the event is handled or not.
Leave	Occurs when the control is no longer the active control of the form.
TextChanged	Occurs when the value of the <code>Text</code> property changes.

Table 1.12: Events of the `TextBox` Class

The following code demonstrates the properties, methods, and events of the `TextBox` class.

### Code Snippet:

```
TextBox txtUserID=new TextBox();
TextBox txtPassword=new TextBox();
txtUserID.AppendText("@school.com");
txtPassword.PasswordChar='*';
txtPassword.Clear();

private void txtUserID_KeyPress(object sender, KeyPressEventArgs e)
{
    txtUserID.CharacterCasing = CharacterCasing.Upper;
}
```

The code creates two `TextBox` controls to accept the user ID and password. The `AppendText` property is used to append the text `@school.com` in `txtPassword` text box. The `PasswordChar` property is used to set the password character as `*`. The `Clear()` method is used to delete the contents in the `txtPassword` text box. When the user presses a key while the focus is on the `txtUserID` text box, the content in the text box is converted to upper case using the `CharacterCasing` property.

The `MaskedTextBox` class is a built-in class used to create the `MaskedTextBox` control and it provides various properties, methods, and events.

## → Properties

The properties of the `MaskedTextBox` class allow you to specify the input mask and prompting character if the user has not given the required input. Table 1.13 lists the commonly used properties of the `MaskedTextBox` class.

Property	Description
Mask	Specifies or retrieves the characters to be used as a mask.

Property	Description
MaskFull	Retrieves a value indicating whether the user has entered all the required characters and the optional characters.
MaskCompleted	Retrieves a value indicating whether the user has entered all the required characters into the input mask.
Name	Specifies or retrieves the name of the control.
PromptChar	Specifies or retrieves the character to be used for prompting the user if the user has missed some input.
Text	Specifies or retrieves the text on the control.

Table 1.13: Properties of the MaskedTextBox Class

#### → Methods

The methods of the MaskedTextBox class allow you to get the location of a character and to select the text in the control. Table 1.14 lists the commonly used methods of the MaskedTextBox class.

Method	Description
GetPositionFromCharIndex	Retrieves the location of a specified character at its given index.
IsKeyLocked	Identifies whether the Caps Lock, Num Lock, or Scroll lock keys are effective on the control.
SelectAll	Selects the entire text. This method is inherited from the TextBoxBase class.

Table 1.14: Methods of the MaskedTextBox Class

#### → Events

The events of the MaskedTextBox class allow you to raise events when the user inputs data. Table 1.15 lists the commonly used events of the MaskedTextBox class.

Event	Description
MaskedChanged	Occurs after the input mask is changed.
MaskedInputRejected	Occurs when the user input does not match with the specified format of the input mask.

Table 1.15: Events of the MaskedTextBox Class

The following code demonstrates the properties, methods, and events of the MaskedTextBox class.

#### Code Snippet:

```
MaskedTextBox msctxtZipCode = new MaskedTextBox();
msctxtZipCode.Mask = "00000-9999";
msctxtZipCode.Text = "095204-7763";
if (!(msctxtZipCode.MaskFull))
{
    MessageBox.Show("Enter the proper zip code");
}
MaskedTextBox msctxtZipCode = new MaskedTextBox();
```

```

msktxtZipCode.Mask = "00000-9999";
msktxtZipCode.Text = "095204-7763";
if (!(msktxtZipCode.MaskFull))
{
    MessageBox.Show("Enter the proper zip code");
}
msktxtZipCode.Focus();
msktxtZipCode.SelectAll();

public void msktxtZipCode_MaskInputRejected(object sender,
MaskInputRejectedEventArgs e)
{
    // You can ask for the correct input
}

```

The code creates a `MaskedTextBox` control using the `MaskedTextBox` class. The text in the control is set to 095204-7763. The input mask is specified as 00000-9999 using the `Mask` property. The `MaskFull` property is used to check whether the user has entered the zipcode. If the user has not entered the zipcode, the user is prompted to enter the zipcode. The focus is maintained on the control using the `Focus()` method and the text in the control is selected using the `SelectAll()` method. The `MaskInputRejected` event is assigned with a handler `msktxtZipCode_MaskInputRejected`, which is invoked when the `MaskInputRejected` event is raised. This event is only raised when the user input does not match the specified format of the input mask.

#### 1.4.4 Button Control

The `Button` control provides the easiest way to allow the user to interact with an application. The user can click the button to perform the required action. For example, the **OK** button in Print dialog box will print the specified document.

The following syntax is used to create the `Button` control.

##### Syntax:

```
Button <objectName> = new Button();
```

where,

`Button`: Is the built-in class used to create the `Button` control.

The following code is used to create a `Button` control named, `btnSubmit`.

##### Code Snippet:

```
Button btnSubmit = new Button();
```

The `Button` class is used to create the `Button` control and provides various properties, methods, and events.

##### → Properties

The properties of the `Button` class handle the appearance and layout of the `Button` control.

Table 1.16 lists the commonly used properties of the `Button` class.

Property	Description
<code> DialogResult</code>	Specifies or retrieves a value which is sent to the parent form when a button is clicked.
<code> Enabled</code>	Specifies or retrieves a value indicating whether the control can be accessed by the user.
<code> FlatStyle</code>	Specifies or retrieves the flat style of the control, which can be <code>Flat</code> , <code>Standard</code> , <code>Popup</code> , or <code>System</code> . These styles are defined in the <code>FlatStyle</code> enumeration.
<code> Image</code>	Specifies or retrieves the image on the control.
<code> Name</code>	Specifies or retrieves the name of the control.
<code> Text</code>	Specifies or retrieves the text on the control.

Table 1.16: Properties of the Button Class

#### → Methods

The methods of the `Button` class handle the behavior of the `Button` control. Table 1.17 lists the commonly used methods of the `Button` class.

Method	Description
<code> Focus</code>	Sets the focus on the control.
<code> PerformClick</code>	Triggers a <code>Click</code> event for a button.

Table 1.17: Methods of the Button Class

#### → Events

Table 1.18 lists the commonly used events of the `Button` class.

Event	Description
<code> Click</code>	Occurs when the control is clicked.
<code> DoubleClick</code>	Occurs when the control is double-clicked.
<code> MouseDoubleClick</code>	Occurs when the control is double-clicked using the mouse.

Table 1.18: Events of the Button Class

The following code demonstrates the creation of a `Button` control.

#### Code Snippet:

```
Button btnSubmit = new Button();
btnSubmit.Text = "Submit";
btnSubmit.FlatStyle = System.Windows.Forms.FlatStyle.System;
void btnSubmit_Click(object sender, EventArgs e)
{
    // User code
}
```

In the code, a Button control is created using the `Button` class. The caption on the control is given as `Submit`. The `FlatStyle` property of the control is set to `System`, which is defined in the `FlatStyle` enumeration. When the user clicks the control, the `Click` event is generated.

### 1.4.5 ListBox and ComboBox Controls

The `ListBox` control is a control used to select a single value or multiple values from the given list of values. It is widely used when the user has to select a value from a huge number of values in the list.

The `ComboBox` control is similar to the `ListBox` control, but it allows you to select only one value at a time. In addition, you can type the value even if the value is not present in the list. You might use the `ComboBox` control to present only the optional list of choices. The `ListBox` control is used when you want the user to compulsorily select from the given choices and prevent the user from entering any of the personal choices.

Figure 1.3 displays the `ListBox` and `ComboBox` controls.

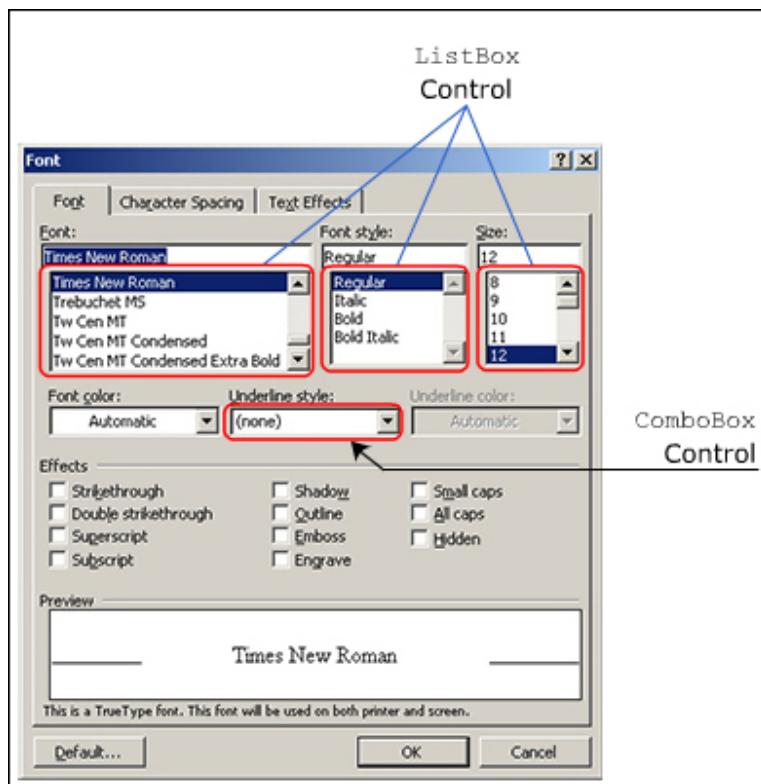


Figure 1.3: ListBox and ComboBox Controls

**Note -** The `ComboBox` control is a combination of the `TextBox` control and the `ListBox` control.

The following syntax is used to create the `ListBox` and `ComboBox` controls.

#### Syntax:

```
ListBox <objectName> = new ListBox();
```

```
ComboBox <objectName> = new ComboBox();
```

where,

`ListBox`: Is the built-in class used to create the `ListBox` control.

`ComboBox`: Is the built-in class used to create the `ComboBox` control.

The following code is used to create the `ListBox` control to list the names of cities and `ComboBox` control to lists the name of countries.

#### Code Snippet:

```
ListBox lstCities = new ListBox();
ComboBox cboCountry = new ComboBox();
```

The `ListBox` class is a built-in class used to create the `ListBox` control and provides various properties, methods, and events.

#### → Properties

The properties of the `ListBox` class allow you to get a list of items and set the way of selecting items. Table 1.19 lists the commonly used properties of the `ListBox` class.

Property	Description
<code>DisplayMember</code>	Specifies or retrieves the property to be displayed for the control. This property is inherited from the <code>Control</code> class.
<code>Items</code>	Retrieves the items of the control.
<code>SelectionMode</code>	Specifies or retrieves the way in which items are selected in the control, which can be: <code>None</code> : For not selecting any item <code>One</code> : For single selection <code>MultiSimple</code> : For multiple selections <code>MultiExtended</code> : For multiple selections using the <b>Shift</b> + Mouse click, <b>Shift</b> + Arrow keys, or <b>Ctrl</b> + Mouse click.
<code>SelectedIndex</code>	Specifies or retrieves the zero-based index number of the selected item in the control. For example, if there are five cities displayed as a list, the third city in the list will have the index number as two.
<code>Text</code>	Specifies or retrieves the text of the selected item in the control.
<code>ValueMember</code>	Specifies or retrieves a user-defined property created in the code, which is to be used as the required value for the items.

Table 1.19: Properties of the `ListBox` Class

#### → Methods

The methods of the `ListBox` class allow you to select an item, deselect an item, and get the text of the item selected.

Table 1.20 lists the commonly used methods of the `ListBox` class.

Method	Description
<code>ClearSelected</code>	Deselects all the selected items in the control.
<code>GetItemText</code>	Retrieves the text of the given item.
<code>GetSelected</code>	Returns true if the specified item is selected and returns false if the specified item is not selected.
<code>SelectSelected</code>	Selects or removes the selection for a specified item.

Table 1.20: Methods of the `ListBox` Class

## → Events

The events of the `ListBox` class allow you to raise events when the user selects an item from the list. Table 1.21 lists the commonly used events of the `ListBox` class.

Event	Description
<code>SelectedIndexChanged</code>	Occurs when the value of the <code>SelectedIndex</code> property is modified. By default, all items are deselected in the list. This means the value of the <code>SelectedIndex</code> property is set to -1. Similarly, when an item is selected, the index value is assigned as the value to the <code>SelectedIndex</code> property.
<code>SelectedValueChanged</code>	Occurs when the value of the <code>SelectedValue</code> property is modified.
<code>ValueMemberChanged</code>	Occurs when the value of the <code>ValueMember</code> property is modified.

Table 1.21: Events of the `ListBox` Class

The following code demonstrates the properties, methods, and events of the `ListBox` class.

### Code Snippet:

```
ListBox lstCountry = new ListBox();
Boolean flag;
lstCountry.Name = "lstCountry";
lstCountry.Items.Add("U.S");
lstCountry.Items.Add("U.A.E.");
lstCountry.Items.Add("Japan");
lstCountry.Items.Add("China");
lstCountry.Items.Add("Russia");
lstCountry.SelectionMode = SelectionMode.MultiExtended;
flag=lstCountry.GetSelected(2);

private void lstCountry_SelectedIndexChanged(object sender, EventArgs e)
{
    MessageBox.Show(lstCountry.SelectedItem.ToString());
}
```

The code creates a `ListBox` control using the `ListBox` class. The `Add()` method is used to add the items to the list. The `SelectionMode` property is set to `MultiExtended` to enable you to select multiple items. The `GetSelected()` method is used to determine whether the item at index two is selected. When the user selects any item from the list, the value of the `SelectedIndex` property is changed and

the `SelectedIndexChanged` event is raised. Thus, a message box is displayed showing the selected item.

The `ComboBox` class is a built-in class used to create the `ComboBox` control and provides various properties, methods, and events.

### → Properties

The properties of the `ComboBox` class allow you to specify the drop-down style, retrieve a list of items, and retrieve the selected item. Table 1.22 lists the commonly used properties of the `ComboBox` class.

Property	Description
<code>DropDownStyle</code>	Specifies or retrieves a value indicating the style of the control. The appearance of the control can be changed to Simple, DropDown, and DropDownList styles. These styles are defined in the <code>ComboBoxStyle</code> enumeration.
<code>Items</code>	Retrieves the object containing the items of the control.
<code>MaxDropDownItems</code>	Specifies or retrieves the maximum number of items to be displayed in the control.
<code>SelectedItem</code>	Specifies or retrieves the selected item in the control.
<code>SelectedIndex</code>	Specifies or retrieves the zero-based index number of the selected item in the control.
<code>Text</code>	Specifies or retrieves the text of the control.
<code>ValueMember</code>	Specifies or retrieves a user-defined property created in the code, which is to be used as the required value for the items.

Table 1.22: Properties of the `ComboBox` Class

### → Methods

The methods of the `ComboBox` class allow you to select the text and activate the control. Table 1.23 lists the commonly used methods of the `ComboBox` class.

Method	Description
<code>GetItemText</code>	Gets the text of the given item.
<code>SelectAll</code>	Selects the text appearing in the control.
<code>Select</code>	Activates a control and is inherited from the <code>Control</code> class.

Table 1.23: Methods of the `ComboBox` Class

### → Events

The events of the `ComboBox` class allow you to raise events when the user selects an item from the list. Table 1.24 lists the commonly used events of the `ComboBox` class.

Event	Description
<code>DropDown</code>	Occurs when the drop-down portion of the control is displayed.

Event	Description
SelectedIndexChanged	Occurs when the value of the SelectedIndex property is modified.
SelectedValueChanged	Occurs when the value of the SelectedValue property is modified.
ValueMemberChanged	Occurs when the value of the ValueMember property is modified.

Table 1.24: Events of the ComboBox Class

The following code demonstrates the properties and events of the ComboBox class.

#### Code Snippet:

```
ComboBox cboCountry = new ComboBox();
cboCountry.Text = "(Select)";

cboCountry.Items.Add("U.S.");
cboCountry.Items.Add("U.A.E.");
cboCountry.Items.Add("Japan");
cboCountry.Items.Add("China");
cboCountry.Items.Add("Russia");
cboCountry.MaxDropDownItems = 3;
cboCountry.DropDownStyle = ComboBoxStyle.Simple;

private void cboCountry_SelectedValueChanged(object sender, EventArgs e)
{
    MessageBox.Show("You have selected " + cboCountry.SelectedIndex.
ToString());
}
```

The code creates a ComboBox control using the ComboBox class. The Text property is used to set the text in the control to (Select). The Add() method is used to add the items to the list. The value of the MaxDropDownItems property is set to 3 to indicate that there will be only three items displayed in the list. The style of the control is set to Simple. When the user selects an item from the list, the SelectedValueChanged event is raised and a message box is displayed showing the item that is selected.

#### 1.4.6 LinkLabel Control

The LinkLabel control is similar to the Label control, but allows you to display its caption as a hyperlink. The control is used to provide a link to a Web page or another Windows Form. For example, you can use the LinkLabel control to open a help file explaining how to use the application.

Figure 1.4 displays the `LinkLabel` control.

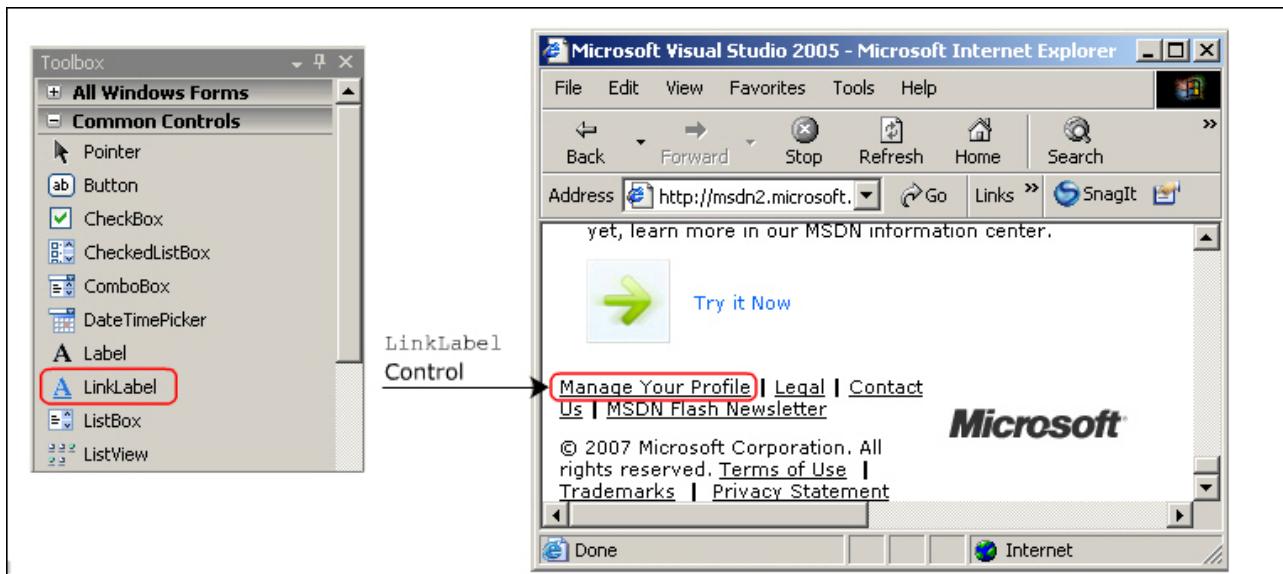


Figure 1.4: LinkLabel Control

The `LinkLabel` control has all the properties, methods, and events of the `Label` class.

The following syntax is used to create the `LinkLabel` control.

#### Syntax:

```
LinkLabel <objectName> = new LinkLabel();
```

where,

`LinkLabel`: Is the built-in class used to create the `LinkLabel` control.

The following code is used to create the `LinkLabel` control.

#### Code Snippet:

```
LinkLabel lnkHelp = new LinkLabel();
```

The `LinkLabel` class is a built-in class used to create the `LinkLabel` control and it provides various properties, methods, and events.

#### → Properties

The properties of the `LinkLabel` class allow you to specify the link color, color of the active link, and the color of the visited link. Table 1.25 lists the commonly used properties of the `LinkLabel` class.

Property	Description
<code>ActiveLinkColor</code>	Specifies or retrieves the color of the hyperlinked text on the control when the user clicks the link.

Property	Description
Links	Retrieves the set of links existing in the control.
LinkColor	Specifies or retrieves the color of the hyperlinked text on the control.
LinkVisited	Specifies or retrieves a value which indicates that the link was visited.
Text	Specifies or retrieves the text on the control.
VisitedLinkColor	Specifies or retrieves the color of the visited hyperlink of the control.

Table 1.25: Properties of the LinkLabel Class

#### → Methods

The methods of the `LinkLabel` class handle the behavior of the `LinkLabel` control. Table 1.26 lists the commonly used method of the `LinkLabel` class.

Method	Description
Select	Activates a control.

Table 1.26: Methods of the LinkLabel Class

#### → Events

The events of the `LinkLabel` class allow you to respond to the user when the user clicks the link. Table 1.27 lists the commonly used event of the `LinkLabel` class.

Event	Description
LinkClicked	Occurs when the hyperlinked text is clicked.

Table 1.27: Events of the LinkLabel Class

The following code demonstrates the creation of `LinkLabel` control.

#### Code Snippet:

```

LinkLabel lnkHelp = new LinkLabel();
lnkHelp.Text = "Help";

void lnkHelp_LinkClicked(object sender, LinkLabelLinkClickedEventArgs e)
{
    lnkHelp.LinkVisited = true;
    lnkHelp.Refresh();
}

```

The code creates a `LinkLabel` control using the `LinkLabel` class. The caption on the `LinkLabel` control is set to `Help`. When the user clicks the control, the `LinkClicked` event is triggered and the `lnkHelp_LinkClicked` method is invoked. This changes the value of the `LinkVisited` property to `true`, and redraws the control on the form using the `Refresh()` method.

## Knowledge Check 4

1. Can you match the properties of the various controls against their corresponding descriptions?

	Description		Property
(A)	Property of TextBox control used to allow the user to enter text on multiple lines.	(1)	PromptChar
(B)	Property of MaskedTextBox control used to denote missed input.	(2)	Image
(C)	Property of ComboBox control used to indicate the number of items in the list.	(3)	Multiline
(D)	Property of the ListBox control used to get or set the selected item.	(4)	MaxDropDownItems
(E)	Property of the Button control used to specify the image.	(5)	SelectedItem

2. You want to create a ListBox control that allows you to select multiple hobbies using the **Ctrl** key from the following list:

Reading, Singing, Writing Poems, Dancing, and Swimming  
Which one of the following codes help you to achieve this?

(A)	<pre>ListBox lstHobbies = new ListBox();  private void frmStudent_Load(object sender, EventArgs e) {     lstHobbies.Items.Add("Reading ");     lstHobbies.Items.Add("Singing");     lstHobbies.Items.Add("Writing Poems");     lstHobbies.Items.Add("Dancing");     lstHobbies.Items.Add("Swimming");     lstHobbies.SelectionMode = SelectionMode.MultiSimple; }</pre>
-----	---

(B)

```
ListBox lstHobbies = new ListBox();

private void frmStudent_Load(object sender, EventArgs e)

{
    lstHobbies.Name = "lstHobbies";

    lstHobbies.Items.Add("Reading ");
    lstHobbies.Items.Add("Singing");
    lstHobbies.Items.Add("Writing Poems");
    lstHobbies.Items.Add("Dancing");
    lstHobbies.Items.Add("Swimming");

    lstHobbies.SelectionMode = SelectionMode.MultiLine;
}
```

(C)

```
ListBox lstHobbies = new ListBox();

private void frmStudent_Load(object sender, EventArgs e)

{
    lstHobbies.Items.Add("Reading ");
    lstHobbies.Items.Add("Singing");
    lstHobbies.Items.Add("Writing Poems");
    lstHobbies.Items.Add("Dancing");
    lstHobbies.Items.Add("Swimming");

    lstHobbies.SelectionMode = SelectionMode.MultiExtended;
}
```

```
ListBox lstHobbies = new ListBox();  
  
private void frmStudent_Load(object sender, EventArgs e)  
{  
  
    lstHobbies.Name = "lstHobbies";  
  
    lstHobbies.Items.Add("Reading ");  
  
(D)    lstHobbies.Items.Add("Singing");  
  
    lstHobbies.Items.Add("Writing Poems");  
  
    lstHobbies.Items.Add("Dancing");  
  
    lstHobbies.Items.Add("Swimming");  
  
    lstHobbies.SelectionMode = SelectionMode.MultiLine;  
  
}
```

## Module Summary

In this module, **Windows Forms and Basic Controls**, you learnt about:

### → Windows Forms

Windows Forms is the Integrated Development Environment (IDE) provided with Visual Studio 2005, which can be used for Windows application development. Windows Forms allow you to install applications with least manual interaction, save global data and user related information, draw graphics having different color effects and textures, and support components for data binding.

### → Form Class

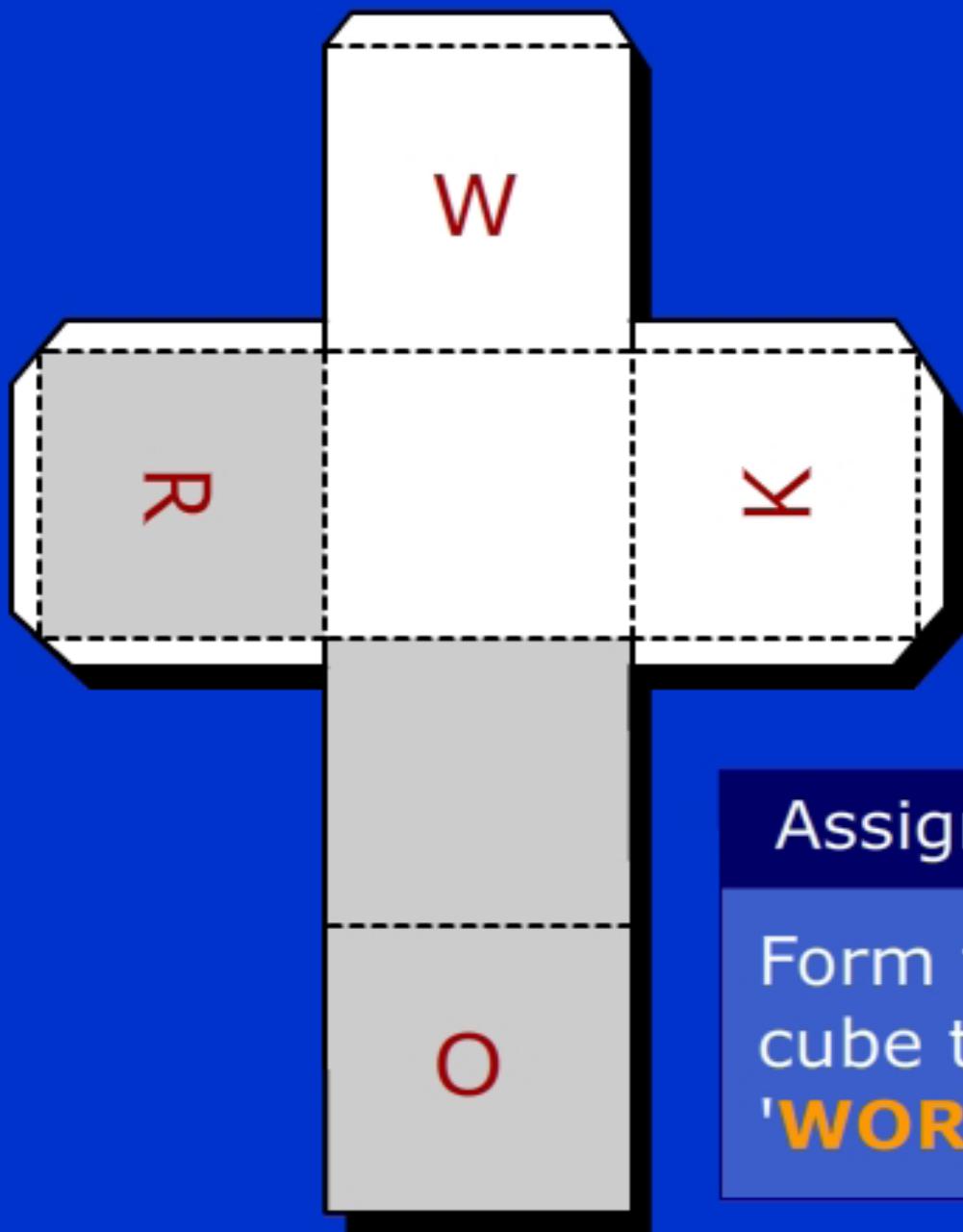
The Form class is used to create a form in an application. It defines various properties used to create modal forms and to set the maximized and minimized icons on the form. It also defines various methods and events used to focus, resize, activate, deactivate, and close the form.

### → Types of Controls

The different types of controls are basic controls, value setting controls, selection list controls, grouping controls, graphics controls, date controls, and menu controls. Each control has a predefined class in the .NET Framework class library, which can be used to customize the controls.

### → Basic Controls

The basic controls include Label, TextBox, MaskedTextBox, Button, ListBox, ComboBox, and LinkLabel. Most of the properties, methods, and events of these controls such as Name, Text properties, Show() method, and the Click event are inherited from the Control class.



Assignment

Form the  
cube to read  
**'WORK'**.

"Practice does not make perfect. Only perfect practice makes perfect."  
- Vince Lombardi

For perfection, solve the assignments @

**[www.onlinevarsity.com](http://www.onlinevarsity.com)**

# Module - 2

## Grouping and Graphic Controls

Welcome to the Module, **Grouping and Graphic Controls**.

Windows Forms contains various controls, which you can use to create a graphical user interface (GUI) of an application. One such category is values setting controls, which allow you to select the required values from a list of values. Another category is of the grouping controls, which enable you to organize the related controls in a group. Lastly, the other category is of the graphic controls, which are used to display images on the Windows Forms.

In this Module, you will learn about:

- Value Setting Controls
- Grouping Controls
- Graphic Controls
- SplitContainer Control

## 2.1 Value Setting Controls

In this first lesson, **Value Setting Controls**, you will learn to:

- List and explain different types of value setting controls.
- State and describe the properties, methods, and events of `RadioButton` control.
- State and describe the properties, methods, and events of `CheckBox` control.
- State and describe the properties, methods, and events of `CheckedListBox` control.

### 2.1.1 Value Setting Controls

Consider a scenario where a developer wants to create a desktop application for tracking the personal details of the employees in an IT firm. The application must allow the HR manager to select the gender and educational degree of employees from two separate lists. Therefore, there is need of such controls, which allow users to select values from a given set of values. This task can be achieved using the value setting controls in Windows Forms.

Value setting controls are controls that allow you to select the desired values from a list of values. There are three types of value setting controls. These are as follows:

#### → `RadioButton`

The `RadioButton` control is used to select a single option from a set of options.

#### → `CheckBox`

The `CheckBox` control is used to select multiple values from a list of values.

#### → `CheckedListBox`

The `CheckedListBox` control is a combination of the `ListBox` and `CheckBox` controls and is used to select multiple values from a list of values.

### 2.1.2 `RadioButton` Control

The `RadioButton` control allows the user to select a single value from the given set of values. This means that only one option can be selected at a time from a given set of options. Therefore, the `RadioButton` control is used to display mutually exclusive options.

The following syntax is used to create the RadioButton control.

#### Syntax:

```
RadioButton <objectName> = new RadioButton();
```

where,

RadioButton: Is the built-in class used to create the RadioButton control.

The following code is used to create a RadioButton control named, radGender.

#### Code Snippet:

```
RadioButton radGender = new RadioButton();
```

The RadioButton class is used to create the RadioButton control. This class defines various properties, methods, and events used to change the appearance and behavior of RadioButton control.

#### → Properties

The properties of the RadioButton class allow you to specify and retrieve the image on the control and the checked state of the control. Table 2.1 lists the most commonly used properties of RadioButton class.

Property	Description
Appearance	Specifies or retrieves a value indicating the appearance of the control. The radio button can appear as a normal button or as a Windows button.
AutoCheck	Specifies or retrieves a value indicating whether the value of the Checked property and the appearance of the control change automatically when the control is clicked.
Checked	Specifies or retrieves a value indicating whether the control is checked or not.
Image	Specifies or retrieves the image appearing on the control.

Table 2.1: Properties of the RadioButton Class

#### → Methods

Table 2.2 lists the most commonly used methods of the RadioButton class.

Method	Description
PerformClick	Generates a Click event for the control, imitating the click action by the user.
Select	Activates the control.
Show	Exhibits the control to the user.

Table 2.2: Methods of the RadioButton Class

#### → Events

The events of the RadioButton class allow the control to respond to the actions performed by the user.

Table 2.3 lists the most commonly used events of the `RadioButton` class.

Event	Description
CheckedChanged	Occurs when there is a change in the value of the <code>Checked</code> property.
Click	Occurs when the radio button control is clicked.

Table 2.3: Events of the `RadioButton` Class

The following code demonstrates how to use the properties, methods, and events of the `RadioButton` class.

**Code Snippet:**

```
RadioButton radMale = new RadioButton();
RadioButton radFemale = new RadioButton();

radMale.Text = "Male";
radMale.Checked = false;
radMale.Show();

radFemale.Text = "Female";
radFemale.Checked = false;
radFemale.Show();

public void radFemale_CheckedChanged(object sender, EventArgs e)
{
    if (radFemale.Checked==true)
    {
        MessageBox.Show("You selected " + radFemale.Text);
        radMale.Checked = false;
    }
}

public void radMale_CheckedChanged(object sender, EventArgs e)
{
    if (radMale.Checked == true)
    {
        MessageBox.Show("You selected " + radMale.Text);
        radFemale.Checked = false;
    }
}
```

The code creates two objects of the `RadioButton` class named, `radMale` and `radFemale`. The `Text` property is set to 'Male' and 'Female' for `radMale` and `radFemale` radio buttons respectively. The `Checked` property is used to unselect the controls by specifying the value as `false`. The `Show()` method is used to display the control on the form to the user. When the user selects any one of the `RadioButton` controls, the `Checked` property is used to check whether the control is checked. If it returns `true`, the text on the control is displayed to the user and the checked status of the other `RadioButton` control is set to `false`.

Figure 2.1 displays the RadioButton control.

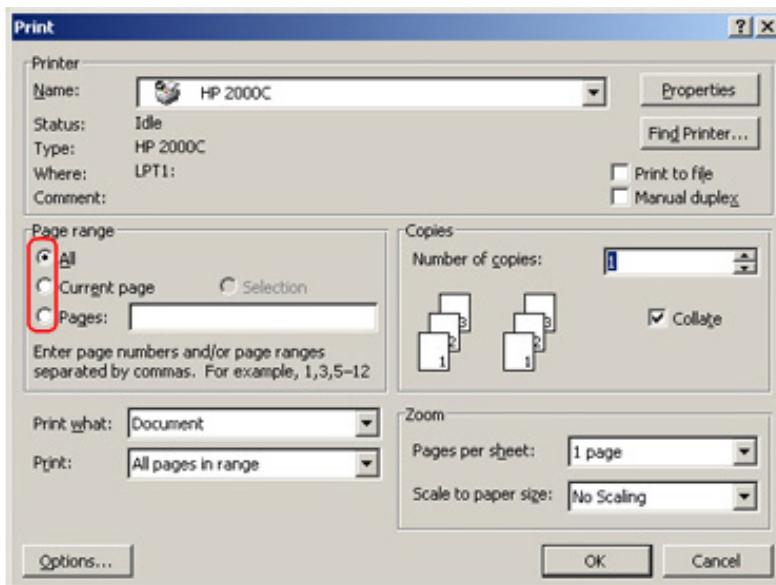


Figure 2.1: RadioButton Control

### 2.1.3 CheckBox Control

The CheckBox control allows you to accept multiple values from a list of values. This control can be used when you want the user to choose one or more options.

The following syntax is used to create the CheckBox control.

#### Syntax:

```
CheckBox <objectName> = new CheckBox ();
```

where,

**CheckBox**: Is the built-in class used to create the CheckBox control.

The following code creates a CheckBox control named, `chkHobbies`.

#### Code Snippet:

```
CheckBox chkHobbies = new CheckBox();
```

The CheckBox class is used to create the CheckBox control and it provides various properties, methods, and events.

#### → Properties

The properties of the CheckBox class allow you to specify or retrieve the state of the control.

Table 2.4 lists the most commonly used properties of the CheckBox class.

Property	Description
Checked	Specifies or retrieves a value indicating whether the control is checked.
CheckState	Specifies or retrieves the state of the control. The control has three states Checked, Indeterminate, and Unchecked. When the control is in the Indeterminate state, the box is deactivated indicating that the control is not valid to be selected for a particular selection.
ThreeState	Specifies or retrieves a value indicating whether the control will allow three checkstates instead of two.

Table 2.4: Properties of the CheckBox Class

#### → Methods

The methods of CheckBox class are used to activate and show a CheckBox control to the user. Table 2.5 lists the most commonly used methods of the CheckBox class.

Method	Description
Select	Activates the control.
Show	Exhibits the control to the user.

Table 2.5: Methods of the CheckBox Class

#### → Events

Table 2.6 lists the most commonly used events of the CheckBox class.

Event	Description
CheckedChanged	Occurs when the Checked property of the control is changed.
Event	
CheckStateChanged	Occurs when the CheckState property of the control is changed.
Click	Occurs when the control is clicked.

Table 2.6: Events of the CheckBox Class

The following code demonstrates how to use the properties, methods, and events of the CheckBox class.

#### Code Snippet:

```
CheckBox chkCD = new CheckBox();
chkCD.Text = "CD";
chkCD.Checked = false;
chkCD.CheckState = CheckState.Indeterminate;
chkCD.Show();

CheckBox chkVCD = new CheckBox();
chkVCD.Text = "VCD";
```

```

chkVCD.Checked = false;
chkVCD.CheckState = CheckState.Unchecked;
chkVCD.Show();

public void chkCD_CheckedChanged(object sender, EventArgs e)
{
    if (chkCD.Checked == false)
    {
        MessageBox.Show("CD required only", "Customer Details",
        MessageBoxButtons.OK, MessageBoxIcon.Information);
        chkCD.Checked = false;
        chkCD.CheckState = CheckState.Indeterminate;
    }
}

```

The code creates two objects of the `CheckBox` class namely, `chkCD` and `chkVCD`. The `CheckState` property of the `chkCD` control is set to the `Indeterminate` state. The `Show()` method is used to display the controls on the form. The `CheckState` property is used to set the state of the controls using the values of the `CheckState` enumeration namely, `Unchecked` and `Indeterminate`. When the user clicks the `chkCD` control, the `CheckedChanged` event is raised and a message box is displayed to the user.

#### 2.1.4 CheckedListBox Control

The `CheckedListBox` control is a control that allows you to accept multiple values from a list. This control is similar to a `ListBox` control, but it displays each value in a list along with a check box, which can be checked or unchecked. The user can view all the values of the list using the scrollbar appearing on the right of the control.

The `CheckListBox` control is used when you want to display additional information to the user. For example, if you are uninstalling an application, you can display a `CheckedListBox` to list the steps and select those steps which have been successfully completed.

The following syntax is used to create the `CheckedListBox` control.

##### Syntax:

```
CheckedListBox <objectName> = new CheckedListBox();
```

where,

`CheckedListBox`: Is the built-in class used to create the `CheckedListBox` control.

The following code creates a `CheckedListBox` control named, `chlCity`.

##### Code Snippet:

```
CheckedListBox chlCity = new CheckedListBox();
```

The `CheckedListBox` class is used to create the `CheckedListBox` control and it provides various properties, methods, and events.

## → Properties

The properties of the `CheckedListBox` class allow you to specify and retrieve the collection of items and the selected items. Table 2.7 lists the most commonly used properties of the `CheckedListBox` class.

Property	Description
<code>CheckedIndices</code>	Indicates the collection of checked indexes in the control.
<code>CheckedItems</code>	Indicates the collection of checked items in the control.
<code>CheckOnClick</code>	Specifies or retrieves a value whether an item in the control should be ticked when clicked. By default, when you click the item, its respective check box is not selected. It is only selected when you click the selected item for the second time.
<code>Items</code>	Retrieves the collection of all the items in the control.
<code>SelectedIndex</code>	Specifies or retrieves the zero-based index of the currently selected item in the control.
<code>SelectedItem</code>	Specifies or retrieves the currently selected item in the control.
<code>SelectedItems</code>	Retrieves the collection of the currently selected items in the control.

Table 2.7: Properties of the `CheckListBox` Class

## → Methods

The methods of `CheckedListBox` class are used to check an item, retrieve the checked state of an item, and clear all the selected items.

Table 2.8 lists the most commonly used methods of the `CheckedListBox` class.

Method	Description
<code>ClearSelected</code>	Unselects the items in the control.
<code>GetItemChecked</code>	Retrieves a value, which indicates whether the specified item is checked.
<code>GetItemCheckState</code>	Retrieves a value, which indicates the check state of the current item.
<code>GetItemText</code>	Retrieves the text of the specified item.
<code>SetItemChecked</code>	Specifies the state for the specified item at the specified index to Checked using the <code>CheckState</code> enumeration. This enumeration defines different values to determine the check state of an item.
<code>SetItemCheckState</code>	Specifies the check state of the specified item at the specified index.

Table 2.8: Methods of the `CheckedListBox` Class

## → Events

Table 2.9 lists the most commonly used events of the `CheckedListBox` class.

Event	Description
<code>ItemCheck</code>	Occurs when the checked state of an item changes.
<code>MouseClick</code>	Occurs when the control is clicked using the mouse.
<code>SelectedIndexChanged</code>	Occurs when the <code>SelectedIndex</code> property of the control changes.
<code>SelectedValueChanged</code>	Occurs when the <code>SelectedValue</code> property of the control changes.

**Table 2.9: Events of the `CheckedListBox` Class**

The following code demonstrates how to use the properties, methods, and events of the `CheckListBox` class.

### Code Snippet:

```
CheckedListBox ch1LanguageList = new CheckedListBox();
ch1LanguageList.Items.Add("English");
ch1LanguageList.Items.Add("Japanese");
ch1LanguageList.Items.Add("Korean");
ch1LanguageList.Items.Add("Chinese");
ch1LanguageList.Items.Add("Spanish");
ch1LanguageList.CheckOnClick = true;

public void ch1LanguageList_ItemCheck(object sender,
    ItemCheckEventArgs e)
{
    if(ch1LanguageList.GetItemCheckState(0) ==
        CheckState.Checked)
    {
        MessageBox.Show("You have selected English");
    }
}
```

The code creates an object of the `CheckedListBox` class.

The `Items` property is used to add the items to the list. The `CheckOnClick` property is used to tick the item when it is selected by a single-click. When the user selects or unselects any item, the `ItemCheck` event is raised. When this event is raised, the check state of the first item called English is verified using the

`GetItemCheckState()` method. If it is checked, the `Checked` property returns `true` and a message box is displayed with the message, 'You have selected English'.

## Knowledge Check 1

1. Can you match the methods, properties, and events of the CheckBox control against their corresponding descriptions?

Description		Property, Method, and Event	
(A)	The property determines whether the control is checked, unchecked, or indeterminate.	(1)	Checked
(B)	The event occurs when the value of the Checked property is changed.	(2)	CheckStateChanged
(C)	The property determines whether the control is checked.	(3)	Select
(D)	The method activates the control.	(4)	CheckState
(E)	The event occurs when the value of the CheckState property is changed.	(5)	CheckedChanged

2. You want to create a `CheckedListBox` control, which will display the following three hobbies- reading, playing, and surfing. Which one of the following codes will help you to achieve this?

(A)	<pre>CheckedListBox chlHobbies = new CheckedListBox(); chlHobbies.Item.Add("Reading"); chlHobbies.Item.Add("Playing"); chlHobbies.Item.Add("Surfing"); Form.Add(chlHobbies);</pre>
(B)	<pre>CheckedListBox chlHobbies = new CheckedListBox(); chlHobbies.Items.Add("Reading"); chlHobbies.Items.Add("Playing"); chlHobbies.Items.Add("Surfing"); Controls.Add(chlHobbies);</pre>
(C)	<pre>CheckedListBox chlHobbies = new CheckedListBox(); chlHobbies.items.add("Reading"); chlHobbies.items.add("Playing"); chlHobbies.items.add("Surfing"); Control.Add(chlHobbies);</pre>

(D)   

```
CheckedListBox ch1Hobbies = new CheckedListBox();
ch1Hobbies.item.add(Reading);
ch1Hobbies.item.add(Playing);
ch1Hobbies.item.add(Surfing);
Form.Add(ch1Hobbies);
```

## 2.2 Grouping Controls

In this second lesson, **Grouping Controls**, you will learn to:

- List the grouping controls.
- Describe the `Panel` control.
- Explain the `GroupBox` control.
- Explain the `Tab` control.

### 2.2.1 Grouping Controls

Grouping controls are controls that function as a container for other controls. They allow you to logically organize controls in a group. This type of grouping is necessary when you want to take specific information based on a certain type or category. For example, you can use grouping controls to take or display a list of members in a company according to its branches or according to its departments.

You have come across the grouping controls frequently while using the various Windows applications. For example, consider Microsoft Word Print dialog box. The Printer and Zoom groups have various items that are logically placed depending upon the type of information they represent.

The common types of grouping controls are as follows:

- `Panel`
- `GroupBox`

### 2.2.2 Panel Control

A `Panel` control is a control that contains other controls. It allows you to place related controls to give a logical view to the user. This means that the `Panel` control acts as a container for a particular group of controls. For example, you can create a `Panel` control to display a list of employee's branch wise and another `Panel` control to list the employees department wise. By default, this control does not have any appearance on the form. To make the control visible, you need to set a border to the control.

The following syntax is used to create the `Panel` control.

**Syntax:**

```
Panel <objectName> = new Panel();
```

where,

`Panel`: Is the built-in class used to create the `Panel` control.

The following code creates the `Panel` control named, `pnlEmployeeDetails`.

**Code Snippet:**

```
Panel pnlEmployeeDetails = new Panel();
```

The `Panel` class is used to create the `Panel` control and it provides various properties, methods, and events.

→ **Properties**

The properties of the `Panel` class allow you to specify the name and the border style of a `Panel` control. Table 2.10 lists the most commonly used properties of the `Panel` class.

Property	Description
<code>AutoSize</code>	Automatically adjusts the size of the <code>Panel</code> control based on the number of controls it contains.
<code>AutoSizeMode</code>	Specifies the automatic sizing mode for the control. There are two types of modes defined in the <code>AutoSizeMode</code> enumeration, which are as follows: <code>GrowOnly</code> : Grows to adjust its contents but does not shrink. <code>GrowAndShrink</code> : Grows and shrinks to adjust its contents
<code>BorderStyle</code>	Specifies the style of the border for the control. There are three types of border styles defined in the <code>BorderStyle</code> enumeration, which are as follows: <code>None</code> : Indicates no border <code>Fixed3D</code> : Indicates three-dimensional border <code>FixedSingle</code> : Indicates single-line border

Table 2.10: Properties of the `Panel` Class

→ **Methods**

Table 2.11 lists the most commonly used methods of the `Panel` class.

Method	Description
<code>Select</code>	Activates the control.
<code>Show</code>	Exhibits the control to the user.

Table 2.11: Methods of the `Panel` Class

## → Events

The events of the `Panel` class allow the form to respond to actions such as change in the `Visible` property or in the style of a control. Table 2.12 lists the most commonly used events of the `Panel` class.

Event	Description
<code>StyleChanged</code>	Occurs when there is a change in the style of the control.
<code>VisibleChanged</code>	Occurs when there is a change in the value of <code>Visible</code> property of the control.

**Table 2.12: Events of the Panel Class**

The following code demonstrates the `Panel` class.

### Code Snippet:

```
Panel pnlCustomerDetails = new Panel();
pnlCustomerDetails.Name = "pnlCustomerDetails";
pnlCustomerDetails.BorderStyle = BorderStyle.FixedSingle;

private void pnlCustomerDetails_StyleChanged(object sender, EventArgs e)
{
    MessageBox.Show("Style Changed");
}
```

The code creates an object of the `Panel` class. The `Name` property is set to `pnlCustomerDetails`. The `BorderStyle` property is set to `FixedSingle` using the `BorderStyle` enumeration. When there is a change in the style of the control, the `pnlCustomerDetails_StyleChanged` event is raised, which displays a message box stating that the style is changed.

### 2.2.3 *GroupBox* Control

The `GroupBox` control is similar to the `Panel` control and is used to group controls. This control appears as frame around the controls that it holds. You can provide a caption to the `GroupBox` as a title to indicate the type of information to be taken or displayed. For example, you can create a group box control to select the gender and educational degree of employees.

For example, consider the Print dialog box of Microsoft Word application. Here, the Page range and Copies area depicts a `GroupBox` control.

The following syntax is used to create the `GroupBox` control.

#### Syntax:

```
GroupBox <objectName> = new GroupBox();
```

where,

`GroupBox`: Is the built-in class used to create the `GroupBox` control.

The following code creates the `GroupBox` control named, `grpGender`.

#### Code Snippet:

```
GroupBox grpGender = new GroupBox();
```

The `GroupBox` class is used to create the `GroupBox` control and it provides various properties, methods, and events.

#### → Properties

Table 2.13 lists the most commonly used properties of the `GroupBox` class.

Property	Description
<code>Anchor</code>	Specifies or retrieves the boundaries of the container to which a control is bound and identifies how a control is resized with its parent.
<code>AutoSize</code>	Specifies or retrieves a value whether the control resizes depending upon its contents.
<code>FlatStyle</code>	Specifies or retrieves the flat style of the control using the values defined in the <code>FlatStyle</code> enumeration.

Table 2.13: Properties of the `GroupBox` Class

#### → Methods

The methods of `GroupBox` class are used to exhibit or to hide the control. Table 2.14 lists the most commonly used methods of the `GroupBox` class.

Method	Description
<code>Hide</code>	Makes the control invisible to the user.
<code>Show</code>	Exhibits the control to the user.

Table 2.14: Methods of the `GroupBox` Class

#### → Events

The events of the `GroupBox` class allow the control to respond to the actions performed by the user. Table 2.15 lists the most commonly used events of the `GroupBox` class.

Event	Description
<code>Resize</code>	Occurs when there is a change in the value of the <code>Resize</code> property, which resizes the control.
<code>StyleChanged</code>	Occurs when there is a change in the style of the control.

Table 2.15: Events of the `GroupBox` Class

The following code demonstrates how to create a `GroupBox` control and add two radio button controls in it.

**Code Snippet:**

```
GroupBox grpGender = new GroupBox();
grpGender.Text = "Gender";
grpGender.Enabled = true;

RadioButton radMale = new RadioButton();
radMale.Text = "Male";

RadioButton radFemale = new RadioButton();
radFemale.Text = "Female";

grpGender.Controls.Add(radMale);
grpGender.Controls.Add(radFemale);
```

The code creates an object of the `GroupBox` class. The `Text` property is used to set as `Gender` of the `GroupBox` control. Two `RadioButtons` are added in the `GroupBox` control to accept the gender of the customer. The `Add()` method is used to add the controls in the `GroupBox` control.

#### 2.2.4 Comparison

The `GroupBox` control is similar to `Panel` control, but there are some differences between these two controls. Table 2.16 lists the differences between the two tables.

GroupBox Control	Panel Control
A <code>GroupBox</code> control can have a caption.	A <code>Panel</code> control does not have a caption.
A <code>GroupBox</code> control is always displayed with a border when placed on a form.	A <code>Panel</code> control is displayed without a border when placed on a form.
A <code>GroupBox</code> control does not support scrolling.	A <code>Panel</code> control supports scrolling.

Table 2.16: Differences Between the `GroupBox` and `Panel` Control

#### 2.2.5 TabControl

The `TabControl` is a grouping control that provides tabs for organizing the information into different logical parts. For example, consider the Borders and Shading dialog box of the Microsoft Word application. Here, Borders, Page Border, and Shading are different tab pages, which are logically placed into tabs depending upon the type of functionality they provide.

Figure 2.2 displays the Borders and Shading.

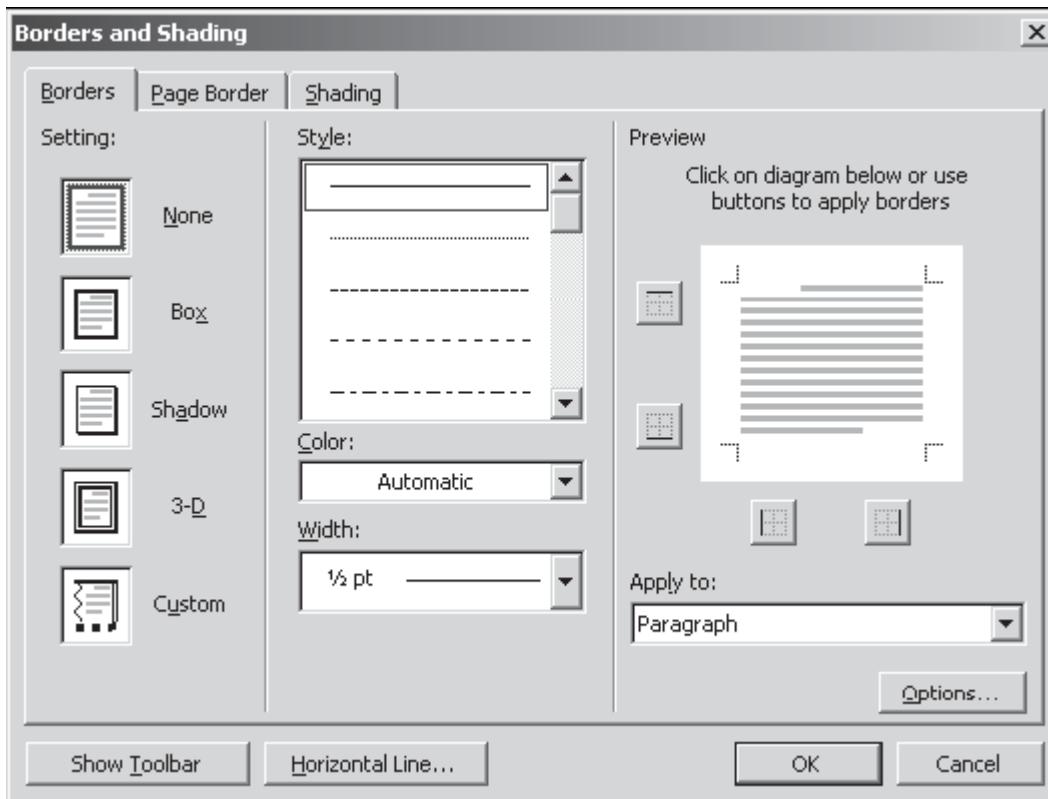


Figure 2.2: Borders and Shading Dialog Box

In .Net Framework 2.0, TabControl is created on a form using the `TabControl` class, which exists in `System.Windows.Forms` namespace. The tab pages contained on the `TabControl` are the objects of the `TabPage` class. The row of tab pages on the `TabControl` is known as a tab strip.

The `TabControl` class consists of various properties, methods, and events. They can be used to specify the name, tab pages, size of tabs, and to track any user generated events while interacting with the control. Table 2.17 lists the properties of `TabControl` class.

Property	Description
<code>Appearance</code>	Specifies or retrieves the appearance of tab pages in a <code>TabControl</code> .
<code>ImageList</code>	Specifies or retrieves the images to be displayed on the tab pages.
<code>RowCount</code>	Retrieves the number of rows displayed on the tab strips of a <code>TabControl</code> .
<code>SelectedIndex</code>	Specifies or retrieves the index of the currently selected tab page.
<code>SelectedTab</code>	Specifies or retrieves the currently selected tab page.
<code>SizeMode</code>	Specifies or retrieves the size of tabs in a <code>TabControl</code> .
<code>TabPage</code>	Retrieves the collection of tab pages in a <code>TabControl</code> .

Table 2.17 : Properties of `TabControl` Class

Table 2.18 lists the methods of TabControl class.

Method	Description
DeselectTab	Sets the tab following the specified tab, as the current tab.
GetControl	Retrieves a TabPage control at a particular location.
SelectTab	Specifies a tab as the current tab.

Table 2.18 : Methods of TabControl Class

Table 2.19 lists the events of TabControl class.

Event	Description
Deselected	Occurs when a tab in a TabControl is deselected.
Selected	Occurs when a tab in a TabControl is selected.

Table 2.19 : Events of TabControl Class

The following code demonstrates the use of some properties, methods, and events of the TabControl class.

#### Code Snippet:

```
TabControl tabSetting = new TabControl();
TabPage tbpStudent = newTabPage();
tbpStudent.Text = "Student Details";
TabPage tbpFaculty = newTabPage();
tbpFaculty.Text = "Faculty Details";
tabSetting.Controls.Add(tbpStudent);
tabSetting.Controls.Add(tbpFaculty);
tabSetting.Appearance = TabAppearance.Normal;
tabSetting.Show();
tabSetting.SelectTab(0);
private void tabSetting_Selected(object sender, TabControlEventArgs e)
{
    MessageBox.Show("Tab Selected");
}
```

In this code, the Text property of the TabPage class is used to set the caption on the two tabs as Student Details and Faculty Details. The tabs are added on the control using the Add() method. The Appearance property of the tabs is set to Normal using the TabAppearance enumeration. The Show() method is used to display the control on the form. By default, the first tab is selected using the SelectTab() method.

When the user clicks the tab, the `tabSetting_Selected` event is raised and a message box appears stating that the tab is selected.

## Knowledge Check 2

- Which of the following statements about the `Panel` and `GroupBox` controls are false?

(A)	The <code>VisibleChanged()</code> method of the <code>panel</code> control is invoked when the value of the <code>Enabled</code> property changes.
(B)	The <code>Anchor</code> property of the group box control is used to set the dimensions of the control.
(C)	A <code>Panel</code> control is displayed without a border on a form.
(D)	The <code>Resize</code> event of the group box control is invoked when the size of the control automatically changes.
(E)	The <code>FlatStyle</code> property of the <code>panel</code> control is used to specify the style using the <code>FlatStyle</code> enumeration.

- Can you match the methods, properties, and events of the `GroupBox` control against their corresponding descriptions?

Description		Property, Method, and Event	
(A)	Property used to set or get a value indicating if the control can respond to user's action.	(1)	<code>FlatStyle</code>
(B)	Property used to specify or retrieve the flat style of the control.	(2)	<code>Show</code>
(C)	Event that occurs when the style of control is modified.	(3)	<code>Resize</code>
(D)	Method used to display the control to the user.	(4)	<code>Enabled</code>
(E)	Event that occurs when the control is resized.	(5)	<code>StyleChanged</code>

### 2.3 Graphic Controls

In this third lesson, **Graphic Controls**, you will learn to:

- List the various types of graphic controls used for displaying images.
- Explain the use of `PictureBox` control.
- Describe the `ImageList` component.

### 2.3.1 Graphic Controls

Consider a scenario where a developer wants to create a desktop application for tracking the details of the books of a library.

The application must store book details along with a book album, which should contain the picture of front page of each book. To create the book album, the developer will need a control which can store images. This task can be achieved using the graphic controls provided in Windows Forms.

Graphic controls are the controls that are used to display graphics on a form. The types of graphic controls commonly used are as follows:

- ➔ PictureBox
- ➔ ImageList

### 2.3.2 PictureBox Control

A `PictureBox` control is used to display images on a form. You can display images as bitmap, GIF, JPEG, metafile, or icon file format using this control.

A `PictureBox` control can be visualized as a container that can hold only a single image at a time.

The following syntax is used to create the `PictureBox` control.

**Syntax:**

```
PictureBox <objectName> = new PictureBox();
```

where,

`PictureBox`: Is the built-in class used to create the `PictureBox` control.

**Code Snippet:**

```
PictureBox picGraphic = new PictureBox();
```

➔ **Properties**

The `PictureBox` class is used to create the `PictureBox` control and it provides various properties, methods, and events. Table 2.20 lists the properties of the `PictureBox` control.

Property	Description
Image	Specifies or retrieves the image displayed by the control.
ErrorImage	Specifies or retrieves the image to be displayed in the control, if an error has occurred while the image is being loaded or the loading process has been cancelled.
InitialImage	Specifies or retrieves the image that will be displayed in the control while the main image is loading.

Property	Description
SizeMode	Specifies how the image is displayed in the control. The values for this property are defined in the <code>PictureBoxSizeMode</code> enumeration, which are as follows: AutoSize: Fits the control automatically to the size of the image CenterImage: Displays the image in the center of the control, if the control's size is larger than the image Normal: Displays the image in the upper-left corner of the control. StretchImage: Stretches the image to fit within the size of the control. Zoom: Increases or decreases the size of an image maintaining the size ratio

Table 2.20: Properties of PictureBox Control

#### → Methods

The methods of `PictureBox` class are used to load the image in the control. Table 2.21 lists the most commonly used methods of the `PictureBox` class.

Method	Description
Load	Loads a graphic in the control.
LoadAsync	Loads a graphic in the control asynchronously.

Table 2.21: Methods of the PictureBox Class

#### → Events

Table 2.22 lists the most commonly used events of the `PictureBox` class.

Event	Description
Click	Occurs when the control is clicked.
Leave	Occurs when the input focus is taken away from the control.
LoadCompleted	Occurs when the asynchronous loading of a graphic in the control is completed, cancelled, or raised an error.
Paint	Occurs when the control is drawn again on the form.

Table 2.22: Events of the PictureBox Class

The following code demonstrates how to use the `PictureBox` control.

#### Code Snippet:

```
PictureBox picBook = new PictureBox();
picBook.Text = "Front page of the Book";
picBook.ImageLocation = "image01.jpg";
picBook.SizeMode = PictureBoxSizeMode.StretchImage;

public void picBook_Click(object sender, EventArgs e)
{
    picBook.SizeMode = PictureBoxSizeMode.CenterImage;
}
```

The code creates an object of the `PictureBox` class. The value of the `Text` property is set to `Front page of the Book`. The `SizeMode` property is set to `StretchImage` using the `PictureBoxSizeMode` enumeration, which will stretch the image to fit within the size of the control. When the user clicks the `PictureBox` control, the `Click` event is invoked and the `SizeMode` property of the control is changed to `CenterImage`, which will place the image in the center. Figure 2.3 displays the `PictureBox` control.



Figure 2.3 : PictureBox Control

### 2.3.3 *ImageList Component*

The `ImageList` component in Windows Forms is used to store a collection of images. The images stored in this component are displayed by other controls. This component is similar to the Microsoft Clip Art pane, which displays a set of images. The `ImageList` component can be visualized as a control that consists of multiple `PictureBox` controls to display multiple images.

The following syntax is used to create the `ImageList` component.

#### Syntax:

```
ImageList <objectName> = new ImageList();
```

where,

`ImageList`: Is the built-in class used to create the `ImageList` control.

The following code is used to create the `ImageList` component, `imglstCD`.

#### Code Snippet:

```
ImageList imglstCD = new ImageList();
```

**Note** - The Microsoft Clip Art pane is a utility of the Microsoft Office applications, which is used to search for different kind of images. It displays multiple images, photographs and drawings based on the search value.

The `ImageList` class is used to create the `ImageList` component and it provides various properties, methods, and events.

#### → Properties

The properties of the `ImageList` class allow you to specify the images, their size, and the color depth of the images in `ImageList` component. Table 2.23 lists the most commonly used properties of the `ImageList` class.

Property	Description
ColorDepth	Specifies or retrieves the depth of the color of the image list.
Images	Retrieves the <code>ImageList.ImageCollection</code> , which is a collection of the image objects in the existing control.
ImageSize	Specifies or retrieves the size of the images in the image list.
Name	Specifies or retrieves the name of the <code>ImageList</code> control.

Table 2.23: Properties of the `ImageList` Class

#### → Methods

The methods of `ImageList` class are used to draw the images on a form. Table 2.24 lists the most commonly used method of the `ImageList` class.

Method	Description
Draw	Draws the image.

Table 2.24: Method of the `ImageList` Class

#### → Events

The events of the `ImageList` class allow the control to respond to the actions performed by the user. Table 2.25 lists the most commonly used event of the `ImageList` class.

Event	Description
RecreateHandle	Occurs when the handle is recreated when the depth value of the image color changes.

Table 2.25: Event of the `ImageList` Class

The following code demonstrates how to use the `ImageList` control.

**Code Snippet:**

```
ImageList imglstBooks = new ImageList();
Image imgNovel01 = Image.FromFile("novel01.jpg");
Image imgStory01 = Image.FromFile("story01.jpg");
Image imgEncyclo01 = Image.FromFile("encyclo01.jpg");
imglstBooks.Images.Add(imgNovel01);
imglstBooks.Images.Add(imgStory01);
imglstBooks.Images.Add(imgEncyclo01);
```

The code creates an object of the `ImageList` class and three objects of the `Image` class. The objects of the `Image` class refers to three different images using the `FromFile()` method. These images are then, added to the `ImageList` component using the `Add()` method.

Figure 2.4 displays the `ImageList` control.

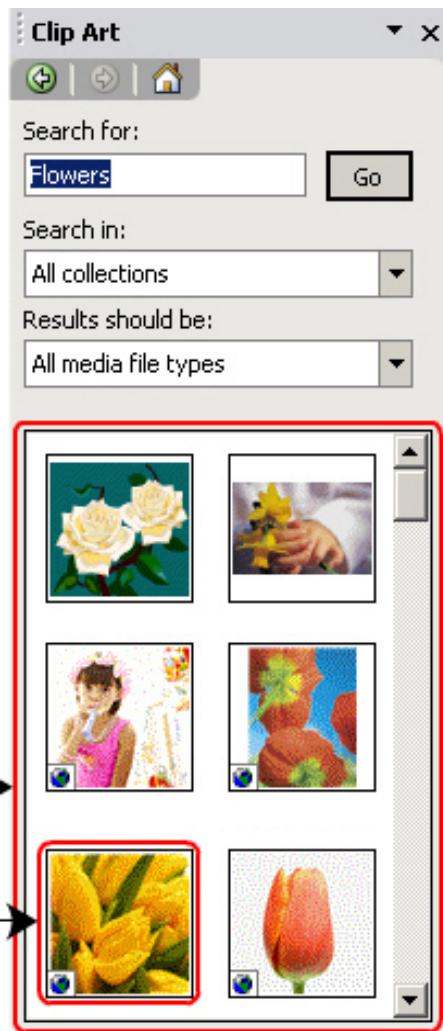


Figure 2.4: ImageList Control

## Knowledge Check 3

1. Can you match the methods, properties, and events of the `PictureBox` control against their corresponding descriptions?

Descriptions			Property, Method, and Event
(A)	Property used to set or get the image to be displayed in the control.	(1)	<code>SizeMode</code>
(B)	Event that occurs when the control is re-drawn on the form.	(2)	<code>LoadCompleted</code>
(C)	Event that occurs when the asynchronous loading of a graphic is cancelled.	(3)	<code>Image</code>
(D)	Method used to load a graphic asynchronously.	(4)	<code>Paint</code>
(E)	Property used to specify how the image will be displayed in the control.	(5)	<code>LoadAsysnc</code>

2. You want to load an image of a toy in a `PictureBox` control. Which one of the following codes will help you to achieve this?

(A)	<pre>PictureBox picBook = new PictureBox(); picBook.Name = "picToyImage"; picBook.Text = "TeddyBear"; picBook.SetImage = "image01.jpg"; picBook.SetBounds(10, 10, 150, 200); picBook.SizeMode = PictureBoxSizeMode.StretchImage;</pre>
(B)	<pre>PictureBox picBook = new PictureBox(); picBook.Name = "picToyImage"; picBook.Text = "TeddyBear"; picBook.ImageLocation = "image01.jpg"; picBook.SetBounds(10, 10, 150, 200); picBook.SizeMode = PictureBoxSizeMode.StretchImage;</pre>
(C)	<pre>PictureBox picBook = new PictureBox(); picBook.Name = "picToyImage"; picBook.Text = "TeddyBear"; picBook.SetImageLocation = "image01.jpg"; picBook.SetBounds(10, 10, 150, 200); picBook.SizeMode = PictureBoxSizeMode.StretchImage;</pre>

(D)

```
PictureBox picBook = new PictureBox();
picBook.Name = "picToyImage";
picBook.Text = "TeddyBear";
picBook.GetImageLocation = "image01.jpg";
picBook.SetBounds(10, 10, 150, 200);
picBook.SizeMode = PictureBoxSizeMode.StretchImage;
```

## 2.4 SplitContainer Control

In this fourth lesson, **SplitContainer** Control, you will learn to:

- Explain the **SplitContainer** control.
- Describe the properties, methods, and events of **SplitContainer** control.

### 2.4.1 SplitContainer Control

The **SplitContainer** control allows you to create complex user interfaces by dividing the form into two resizable panels, either horizontally or vertically. The two panels are separated by a movable bar called as the splitter bar. This bar is used to change the size of the display area of the individual panel. The **SplitContainer** control is used when you want to display the hierarchical information in one panel and its sub-contents in another panel.

The **SplitContainer** control makes the appearance of the form similar to the Windows Explorer window, which consists of two panels.

The **SplitContainer** class is used to create a **SplitContainer** control and it defines various properties, methods, and events.

#### → Properties

The properties of the **SplitContainer** class are used to change the appearance of panels and the splitter of a **SplitContainer** control. Table 2.26 lists the most commonly used properties of the **SplitContainer** class.

Property	Description
BorderStyle	Specifies or retrieves the border style of the control using the <b>BorderStyle</b> enumeration.
FixedPanel	Specifies or retrieves which panel of the control will not change in size when the container is resized.
IsSplitterFixed	Specifies or retrieves a value indicating whether the splitter present on the control is fixed or movable.

Property	Description
Panel1	Retrieves the left or top panel of the control depending on the vertical or horizontal orientation.
Panel2	Retrieves the right or bottom panel of the control depending on the vertical or horizontal orientation.
Orientation	Specifies or retrieves a value, which indicates the horizontal or vertical orientation of the panels.

Table 2.26: Properties of the SplitContainer Class

#### → Methods

The methods of the `SplitContainer` class are used to raise the events when the splitter of the control is moved. Table 2.27 lists the most commonly used methods of the `SplitContainer` class.

Method	Description
OnSplitterMoved	Triggers the <code>SplitterMoved</code> event.
OnSplitterMoving	Triggers the <code>SplitterMoving</code> event.

Table 2.27: Methods of the SplitContainer Class

#### → Events

The events of `SplitContainer` class allow you to handle user interaction with the control. Table 2.28 lists the most commonly used events of the `SplitContainer` class.

Event	Description
Click	Occurs when the control is clicked.
SplitterMoved	Occurs when the splitter is moved.
SplitterMoving	Occurs when a splitter is currently moving.

Table 2.28: Events of the SplitContainer Class

The following code demonstrates how to create a `SplitContainer` control.

#### Code Snippet:

```
SplitContainer spltcntrBookStore = new SplitContainer();
spltcntrBookStore.BorderStyle = BorderStyle.Fixed3D;
spltcntrBookStore.Size = new System.Drawing.Size(212, 435);
spltcntrBookStore.TabIndex = 0;
spltcntrBookStore.Orientation = Orientation.Horizontal;
Label lblName = new Label();
lblName.Text = "Test:";
spltcntrBookStore.Panel1.Controls.Add(lblName);
this.Controls.Add(spltcntrBookStore);
```

The code creates an object of `SplitContainer` class. The `BorderStyle` property is set to specify the border style of this control to `Fixed3D`. The `Size` property is used to set the size of the control. The orientation of the panels of the control is set to `Horizontal` using the `Orientation` property. A label with the caption `Test:` is created and added to the control. Finally, the `SplitContainer` control is added on the form using the `Add()` method. Figure 2.5 displays the `SplitContainer` control.

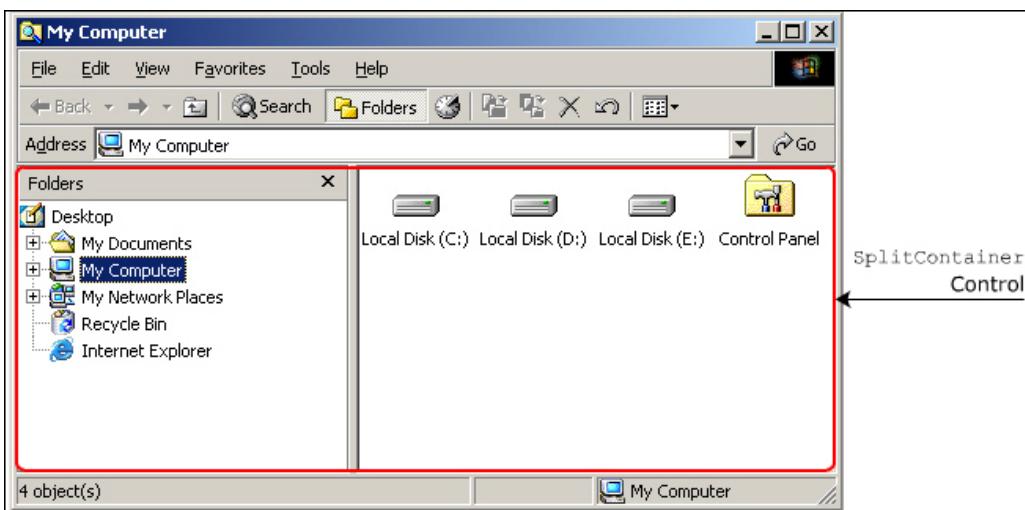


Figure 2.5: SplitContainer Control

## Knowledge Check 4

- Which of the following statements about the `SplitContainer` control are true?

(A)	The <code>SplitContainer</code> control is used to separate the form into horizontal or vertical panels.
(B)	The <code>Panel1</code> property is used to retrieve the right or the bottom panel of a <code>SplitContainer</code> control.
(C)	The <code>FixedPanel</code> property is used to specify which panel of a <code>SplitContainer</code> control is not resized while changing the size of the form.
(D)	The <code>SplitterMoving</code> event is raised when the splitter is moved.
(E)	The <code>Shown()</code> method is used to display the <code>SplitContainer</code> control to the user.

2. You want to display a form divided into two panels. The left pane of the form consists of a label. Which one of the following codes will help you to achieve this?

(A)	<pre>SplitContainer spltcntrBookStore = new SplitContainer; spltcntrBookStore.SplitterWidth = 5; Label lblBooks = new Label(); lblBooks.Text = "Books"; spltcntrBookStore.Panel1.Controls.Add(lblBooks); spltcntrBookStore.Show(); Controls.Add(spltcntrBookStore);</pre>
(B)	<pre>SplitContainer spltcntrBookStore = new SplitContainer(); spltcntrBookStore.SplitterWidth = 5; Label lblBooks = new Label(); lblBooks.Text = "Books"; spltcntrBookStore.Panel1.Controls.Add(lblBooks); spltcntrBookStore.Show(); Controls.Add(spltcntrBookStore);</pre>
(C)	<pre>SplitContainer spltcntrBookStore = new SplitContainer(); spltcntrBookStore.SplitterWidth = 5; Label lblBooks = new Label(); lblBooks.Text = "Books"; spltcntrBookStore.Panel1.Controls.Add(lblBooks); spltcntrBookStore.Show(); Controls.Add(spltcntrBookStore);</pre>
(D)	<pre>SplitContainer spltcntrBookStore = new SplitContainer(); spltcntrBookStore.SplitterWidth = 5; Label lblBooks = new Label(); lblBooks.Text = "Books"; spltcntrBookStore.Panel1.Controls.Add("lblBooks"); spltcntrBookStore.Show(); Controls.Add(spltcntrBookStore);</pre>

## Module Summary

In this module, **Grouping and Graphic Controls**, you learnt about:

→ **Value Setting Controls**

Value setting controls are controls that allow selection of the required value from a given list of values. The different types of value setting controls are the RadioButton control, CheckBox control, and CheckedListBox control.

→ **Grouping Controls**

Grouping controls are controls that function as a container for other controls. The different types of grouping controls are Panel and GroupBox controls.

→ **Graphic Controls**

Graphic controls are controls that are used to store a collection of images and display them. The PictureBox control is used to display a single image at a time. The ImageList component is used to store a set of images, which can be displayed using other controls.

→ **SplitContainer Control**

A SplitContainer control allows you to create complex user interfaces. This control divides the form into two panels, either horizontally or vertically. These panels are separated by a splitter bar, which is used to change the width of the panels.



Visit the  
**Frequently Asked Questions**  
section @

# Module - 3

## Advanced Controls

Welcome to the Module, **Advanced Controls**.

This module covers the selection list controls that are used for selecting a value from a list. The ListView control displays a collection of items within a collection. The TreeView control displays the data in a hierarchical manner. The RichTextBox control allows displaying, entering, and manipulating text within the control. The ProgressBar control is used in applications where you want to view the progress of an operation.

In this Module, you will learn about:

- ➔ Selection List Controls
- ➔ ListView and TreeView Controls
- ➔ RichTextBox Control
- ➔ ProgressBar Control

### 3.1 Introduction to Selection List Controls

In this first lesson, **Introduction to Selection List** Controls, you will learn to:

- List the various selection list controls.
- Describe the `NumericUpDown` control.
- Explain the `DomainUpDown` control.

#### 3.1.1 Selection List Controls

Consider a scenario where a developer wants to create a desktop application for tracking the order details of a bookstore. One of the requirements is that the application must provide the printing option to print the form. In addition, it must also, provide another option to select the number of copies to be printed. Another requirement is to allow the user to print maximum five copies at a time. Thus, the developer needs a control that will allow the user to select the number of copies to be printed from a given range. This can be achieved by using the selection list controls.

Selection list controls are controls used for selecting a value from a specified range of values. These controls allow you to select values from a range using the up and down arrows. There are two types of selection list controls, namely, `NumericUpDown` and `DomainUpDown`.

#### 3.1.2 NumericUpDown Control

The `NumericUpDown` control is a selection list control that allows you to select numeric values from a range of values. This control allows you to select only a single value at a time, such as the number of copies option in the Print dialog box.

This control is useful for collecting data such as number of employees, number of students in a class, and number of days from the user.

The following syntax is used to create the `NumericUpDown` control.

##### Syntax:

```
NumericUpDown <objectName> = new NumericUpDown();
```

where,

`NumericUpDown`: Is the built-in class used to create the `NumericUpDown` control.

`objectName`: Is the object name for `NumericUpDown` class.

`new`: Is the keyword used to allocate memory to the object of `NumericUpDown` class.

The following code creates the `NumericUpDown` control named, `updPrint`.

#### Code Snippet:

```
NumericUpDown updPrint = new NumericUpDown();
```

The `NumericUpDown` class is used for creating the `NumericUpDown` control and it provides various properties, methods, and events.

#### → Properties

The properties of the `NumericUpDown` class allow you to specify the minimum and maximum values for the control. Table 3.1 lists the most commonly used properties of the `NumericUpDown` class.

Property	Description
Increment	Specifies or retrieves the value by which the current value in the control will be incremented or decremented when up or down button is clicked.
Maximum	Specifies or retrieves the maximum value that the control can have in a particular range.
Minimum	Specifies or retrieves the minimum value that the control can have in a particular range.
ThousandsSeparator	Specifies or retrieves a value indicating whether a thousands separator is displayed in the up-down control.
Value	Specifies or retrieves the value assigned to the control.

Table 3.1: Properties of the `NumericUpDown` Class

#### → Methods

The methods of `NumericUpDown` class are used to modify the values present in the control. Table 3.2 lists the most commonly used methods of the `NumericUpDown` class.

Method	Description
DownButton	Decreases the value of the up-down control.
UpButton	Increases the value of the up-down control.

Table 3.2: Methods of the `NumericUpDown` Class

#### → Events

Table 3.3 lists the most commonly used events of the `NumericUpDown` class.

Event	Description
ValueChanged	Occurs when the Value property of the control has been changed.

Table 3.3: Events of the `NumericUpDown` Class

The following code demonstrates how to use the properties, methods, and events of the `NumericUpDown` class.

#### Code Snippet:

```
NumericUpDown updPrint = new NumericUpDown();
updPrint.Minimum = 0;
updPrint.Maximum = 100;
updPrint.Increment = 1;
updPrint.UpButton();

private void updPrint_ValueChanged(object sender, EventArgs e)
{
    // write code to perform some action
}
```

The code displays the `NumericUpDown` control that allows you to select a value from a range of 100 values. The `Maximum` and `Minimum` property sets the maximum value to 100 and minimum value to 0 respectively. The `Increment` property is used to increment the value in the control by one. The `UpButton()` method is used to increase the value within the control. When the `Value` property is modified, the `ValueChanged` event is fired. Figure 3.1 displays the `NumericUpDown` control.

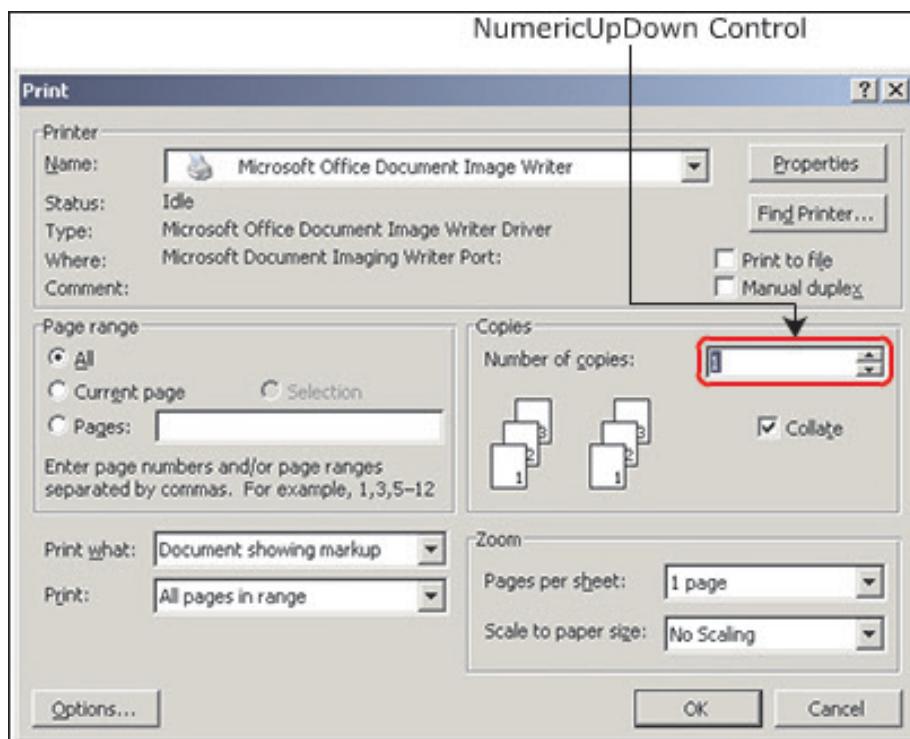


Figure 3.1: NumericUpDown Control

### 3.1.3 DomainUpDown Control

The `DomainUpDown` control is a selection list control that allows you to select text values from a range of values. This control allows you to select only a single value at a time and the value can be selected using the up and down arrows of the control.

This control is generally used when the range of values indicates an order such as days of a week and months of the year.

The following syntax is used to create the `DomainUpDown` control.

#### Syntax:

```
DomainUpDown <objectName> = new DomainUpDown();
```

where,

`DomainUpDown`: Is the built-in class used to create the `DomainUpDown` control.

The following code creates the `DomainUpDown` control named, `dudMonth`.

#### Code Snippet:

```
DomainUpDown dudMonth = new DomainUpDown();
```

The `DomainUpDown` class is used for creating the `DomainUpDown` control and it provides various properties, methods, and events.

#### → Properties

The properties of the `DomainUpDown` class allow you to select an item and wrap the text. Table 3.4 lists the most commonly used properties of the `DomainUpDown` class.

Property	Description
<code>Items</code>	Assigns a collection of objects to the control.
<code>MaximumSize</code>	Specifies or retrieves the maximum size of the up and down arrows.
<code>MinimumSize</code>	Specifies or retrieves the minimum size of the up and down arrows.
<code>ReadOnly</code>	Specifies or retrieves a value indicating whether the text in the control can be modified using the up and down buttons.
<code>SelectedItem</code>	Specifies or retrieves the selected item using the index value of the selected item in the collection.
<code>Wrap</code>	Specifies or retrieves a value indicating whether the text of each item, if bigger than the size of the control, can be shifted on the next line.

Table 3.4: Properties of the `DomainUpDown`

## → Methods

Table 3.5 lists the most commonly used methods of the `DomainUpDown` class.

Method	Description
DownButton	Displays the next item present in the object collection.
UpButton	Displays the previous item present in the object collection.

Table 3.5: Methods of the `DomainUpDown` Class

## → Events

Table 3.6 lists the most commonly used events of the `DomainUpDown` class.

Event	Description
Click	Occurs when the control is clicked.
SelectedIndexChanged	Occurs when the value of the <code>SelectedItem</code> property has been changed.

Table 3.6: Events of the `DomainUpDown` Class

The following code demonstrates how to use the properties, methods, and events of the `DomainUpDown` class.

### Code Snippet:

```
DomainUpDown dudMonth = new DomainUpDown();
dudMonth.Items.Add("January");
dudMonth.Items.Add("February");
dudMonth.Items.Add("March");
dudMonth.Items.Add("April");
dudMonth.MaximumSize = new Size(100, 100);
dudMonth.MinimumSize = new Size(50, 50);
dudMonth.DownButton();

private void dudMonth_SelectedIndexChanged(object sender, EventArgs e)
{
    // Write code to perform some action
}
```

The code displays the `DomainUpDown` control that contains maximum of four string items in it. The `MinimumSize` and `MaximumSize` property sets the minimum and maximum size of the up and down arrows in the control. The `DownButton()` method displays the next item in the object collection. When the selected item in the control is changed, the `SelectedIndexChanged` event occurs and the user can perform some operations.

## Knowledge Check 1

1. You want to store strings representing months in the DomainUpDown control. Which one of the following codes will help you to achieve this?

(A)	<pre>DomainUpDown dudReport = new DomainUpDown(); dudReport.Name() = "dudReport"; dudReport.Items.Add("August"); dudReport.Items.Add("October"); dudReport.Items.Add("December"); dudReport.Items.Add("March"); Controls.Add(dudReport);</pre>
(B)	<pre>DomainUpDown dudReport = new DomainUpDown(); dudReport.Name = "dudReport"; dudReport.Items.AddItem("August"); dudReport.Items.AddItem("October"); dudReport.Items.AddItem("December"); dudReport.Items.AddItem("March"); Controls.Add(dudReport);</pre>
(C)	<pre>DomainUpDown dudReport = new DomainUpDown(); dudReport.Name = "dudReport"; dudReport.Items.Add("August"); dudReport.Items.Add("October"); dudReport.Items.Add("December"); dudReport.Items.Add("March"); Controls.Add(dudReport);</pre>
(D)	<pre>DomainUpDown dudReport = new DomainUpDown(); dudReport.Name ("dudReport"); dudReport.Items.Add ("August"); dudReport.Items.Add ("October"); dudReport.Items.Add ("December"); dudReport.Items.Add ("March"); Controls.Add (dudReport);</pre>

2. Can you match the properties, methods, and events of the `NumericUpDown` class against their corresponding descriptions?

Description			Property, Method, and Event
(A)	Event that occurs when the value of the <code>Value</code> property is changed.	(1)	Increment
(B)	Method used to increase the value of the <code>NumericUpDown</code> control.	(2)	Maximum
(C)	Property used to retrieve or specify the increment or decrement number.	(3)	UpButton
(D)	Property used to specify the maximum value for the control.	(4)	Value
(E)	Method used to specify or retrieve the value assigned to the control.	(5)	ValueChanged

## 3.2. ListView and TreeView Controls

In this second lesson, **ListView and TreeView** Controls, you will learn to:

- Describe the `ListView` control.
- Explain the `TreeView` control.

### 3.2.1 ListView Control

The `ListView` control is used to display a collection of items in a list. This control allows you to add items to a collection and displays the items along with their icons, which can be small or large in size. This control appears similar to the Views icon on the standard toolbar of Windows Explorer. There are five main views of `ListView` control, which are listed as follows:

- **Tile**  
Items appear along with their full-sized icons and are arranged row-wise.
- **List**  
Items are displayed as a list along with their icons, which appear small in size.
- **Details**  
Items are displayed as a list but with extra details. For example, if items are files, then, the file size, type, and modified date are displayed and these details appear in columnar format.

→ **SmallIcon**

Each item appears as a small icon with a label to its right.

→ **LargeIcon**

Each item appears as a full-sized item with a label.

The following syntax is used to create the ListView control.

**Syntax:**

```
ListView <objectName> = new ListView();
```

where,

ListView: Is the built-in class used to create the ListView control.

The following code creates the ListView control named, lvwStudent.

**Code Snippet:**

```
ListView lvwStudent = new ListView();
```

Figure 3.2 displays the Views of ListView control.



Figure 3.2: Views of the ListView Control

The `ListView` class allows you to create the `ListView` control and it provides various properties, methods, and events.

#### → Properties

The properties of the `ListView` class allow you to retrieve the selected items and select multiple items in the collection. Table 3.7 lists the most commonly used properties of the `ListView` class.

Property	Description
Columns	Retrieves the collection of all column headers that are displayed in the control.
Items	Retrieves a collection that contains all items in the control.
MultiSelect	Specifies or retrieves a value that indicates whether multiple items can be selected.
SelectedItems	Retrieves the items that are selected in the control.
View	Specifies or retrieves how items are displayed in the control. The value of this property can be set using the different types of views defined in the <code>View</code> enumeration. The different types of views are <code>Tile</code> , <code>List</code> , <code>Details</code> , <code>SmallIcon</code> and <code>LargeIcon</code> .

Table 3.7: Properties of the `ListView` Class

#### → Methods

The methods of `ListView` class are used to arrange icons and sort items in the `ListView` control. Table 3.8 lists the most commonly used methods of the `ListView` class.

Method	Description
<code>ArrangeIcons</code>	Arranges icons in the control when they are displayed as icons.
<code>Clear</code>	Removes all items and columns from the control.
<code>GetItemAt</code>	Retrieves the item at a specified location.
<code>Sort</code>	Sorts the list view items.

Table 3.8: Methods of the `ListView` Class

#### → Events

Table 3.9 lists the most commonly used events of the `ListView` class.

Event	Description
<code>ColumnClick</code>	Occurs when the column header within the list view control is clicked.
<code>ItemCheck</code>	Occurs when the check state of an item is modified.
<code>ItemSelectionChanged</code>	Occurs when the selection state of an item is changed.
<code>SelectedIndexChanged</code>	Occurs when the index of the selected item in the list view control is modified.

Table 3.9: Events of the `ListView` Class

The following code demonstrates the `ListView` class.

**Code Snippet:**

```
ListView lvwStudent = new ListView();
lvwStudent.Items.Add("Adam");
lvwStudent.Items.Add("Nicole");
lvwStudent.Items.Add("Maria");
lvwStudent.Items.Add("Stephen");
lvwStudent.MultiSelect = true;

private void lvwStudent_Click(object sender, EventArgs e)
{
    if (lvwStudent.View.Equals(View.LargeIcon))
        lvwStudent.View = View.List;
    else if (lvwStudent.View.Equals(View.List))
        lvwStudent.View = View.LargeIcon;
}
```

The code displays the `ListView` control in which the items added. The `Items` property adds items to the list. The `MultiSelect` property specifies a value that indicates whether multiple items can be selected. When the user clicks the control, the view of the `ListView` control is changed.

### 3.2.2 TreeView Control

The `TreeView` control displays data in a hierarchical manner. This control is similar to the left pane of the Windows Explorer. The `TreeView` control has three types of nodes. These are as follows:

→ **Root**

The root node is the main node that has one or more child nodes but has no parent node.

→ **Parent**

The parent node is under the root node and has one or more child nodes.

→ **Leaf**

The leaf node has a parent node but no child nodes. This means that the leaf node is the last node in the hierarchy.

The following syntax is used to create the `TreeView` control.

**Syntax:**

```
TreeView <objectName> = new TreeView();
```

where,

`TreeView`: Is the built-in class used to create the `TreeView` control.

The following code creates the TreeView control named, `tvwProjects`.

**Code Snippet:**

```
TreeView tvwProjects = new TreeView();
```

**Note** - The node of a TreeView control is represented using the `TreeNode` class. The class is used to manage the nodes in the control. For example, you can navigate through the nodes and identify the state and position of a node using the properties of `TreeNode` class.

Figure 3.3 displays the TreeView control.

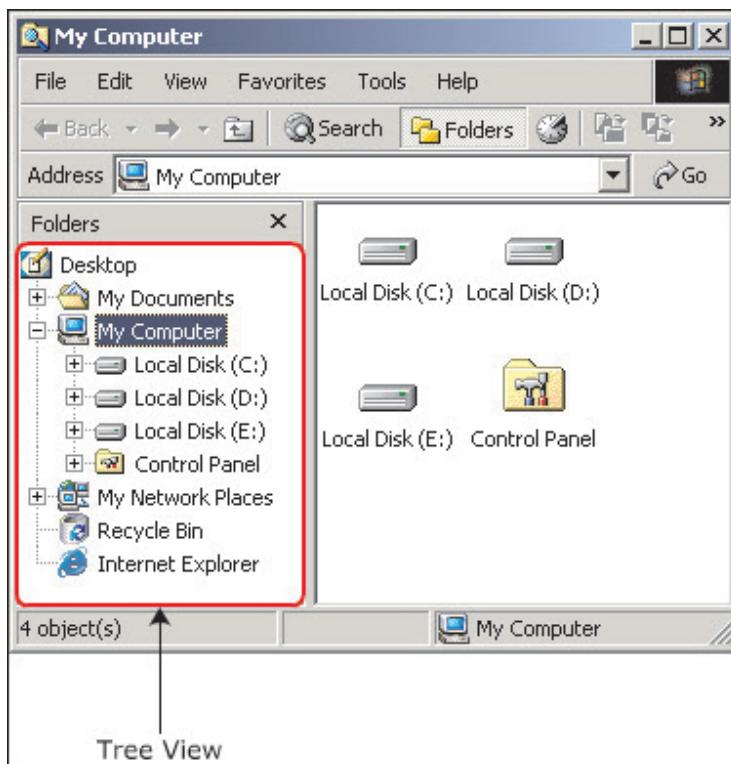


Figure 3.3: TreeView Control

The `TreeView` class allows you to create a `TreeView` control and it defines various properties, methods, and events.

→ **Properties**

The properties of the `TreeView` class allow you to retrieve the nodes and the selected node of the control. Table 3.10 lists the most commonly used properties of the `TreeView` class.

Property	Description
Nodes	Retrieves a set of <code>TreeNode</code> objects that represents the root nodes of the control.
SelectedNode	Specifies or retrieves the tree node that is currently selected in the control.

Property	Description
ShowPlusMinus	Specifies or retrieves a value, which indicates whether a plus sign (+) and minus sign (-) buttons are displayed next to tree nodes, which contain child tree nodes.
ShowRootLines	Specifies or retrieves a value, which indicates whether lines are drawn between the tree nodes that are located at the root of the tree view.
TopNode	Retrieves the first tree node, which is completely visible in the control.

Table 3.10: Properties of the TreeView Class

#### → Methods

The methods of TreeView class are used to expand and collapse the nodes of the control. Table 3.11 lists the most commonly used methods of the TreeView class.

Method	Description
CollapseAll	Collapses the nodes present in the control.
ExpandAll	Expands the nodes present in the control.
GetNodeAt	Retrieves the tree node at the specified location.
GetNodeCount	Retrieves the number of tree nodes, optionally including nodes in the subtrees.

Table 3.11: Methods of the TreeView Class

#### → Events

Table 3.12 lists the most commonly used events of the TreeView class.

Event	Description
AfterCollapse	Occurs after the tree node is collapsed.
AfterExpand	Occurs after the tree node within the control is expanded.
AfterSelect	Occurs after the tree node is selected.
ItemDrag	Occurs when the user drags a node.
NodeMouseClick	Occurs when the user clicks a TreeNode with the mouse.

Table 3.12: Events of the TreeView Class

The following code demonstrates how to use the properties, methods, and events of the TreeView class.

#### Code Snippet:

```
TreeView tvwProjects = new TreeView();
tvwProjects.Nodes.Add("Projects");
tvwProjects.Nodes[0].Nodes.Add("IT");
tvwProjects.Nodes[0].Nodes[0].Nodes.Add("Hi-Fly AirLines");
tvwProjects.Nodes[0].Nodes.Add("e-Commerce");
```

```

tvwProjects.Nodes[0].Nodes[1].Nodes.Add("Inova Bank");
tvwProjects.SelectedNode = tvwProjects.Nodes[0];
tvwProjects.Sort();

void tvwProjects_AfterExpand(object sender, TreeViewEventArgs e)
{
    tvwProjects.SelectedNode = tvwProjects.SelectedNode.Nodes[0];
}

```

The code displays the TreeView control in which the Nodes property creates a node called Projects. The code creates two child nodes called IT and e-Commerce. The node IT contains another child node called Hi-Fly AirLines and the node e-Commerce contains another node called Inova Bank. The SelectedNode property sets the Projects node to be selected by default. The Sort() method sorts the nodes in alphabetical order. The AfterExpand event occurs when any node in the TreeView control is expanded. When this event is raised, the first node in the control is selected.

## Knowledge Check 2

1. Which of these statements about ListView and TreeView controls are true?

(A)	The Detail property of ListView control displays items with medium icons.
(B)	The TreeView control displays data in hierarchical manner.
(C)	The ListView controls supports sorting of items.
(D)	The child node of the TreeView control consists of one or more child nodes and a parent node.
(E)	The ListView control does not allow displaying extra information about an item.

2. Which of the following snippets about the ListView control are correct?

(A)	ListView lvwDiary = new ListView();
(B)	lvwDiary.View ="List";
(C)	lvwDiary.Items.Add("Michael");
(D)	lvwDiary.Items.Add("Glenn");
(E)	lvwDiary.Items.AddNew("Smith");
(F)	lvwDiary.Items[0].Selected(true);
(G)	lvwDiary.SetFocus();
(H)	Controls.Add(lvwDiary);

### 3.3 RichTextBox Control

In this third lesson, **RichTextBox control**, you will learn to:

- State the need for a rich text control and explain the RichTextBox control.
- Describe the properties, methods, and events of RichTextBox control.

#### 3.3.1 RichTextBox Control

The RichTextBox control is used for displaying, entering, and manipulating text data. The control is similar to the WordPad application, as it allows text formatting such as making the text bold, applying bullets and links, and changing the font of the text. The control also, loads text and embedded images from a file, provides undo and redo editing operations, and allows you to find specified characters.

The RichTextBox control, by default, displays both horizontal and vertical scrollbars. The text in the RichTextBox control can either be accessed in text format or in rich text format (.rtf).

The following syntax is used to create the RichTextBox control.

##### Syntax:

```
RichTextBox <objectName> = new RichTextBox();
```

where,

**RichTextBox:** Is the built-in class used to create the RichTextBox control.

The following code creates the RichTextBox control named, `rtbFeedback`.

##### Code Snippet:

```
RichTextBox rtbFeedback = new RichTextBox();
```

The RichTextBox class is used to create the RichTextBox control and it provides various properties, methods, and events.

##### → Properties

The properties of the RichTextBox class allow you to set the font of the text, apply scrollbars and retrieve the selected text. Table 3.13 lists the most commonly used properties of the RichTextBox class.

Property	Description
Font	Specifies or retrieves the font of the text displayed in the control.
ScrollBars	Specifies or retrieves the type of scroll bars for the control.
SelectedText	Specifies or retrieves the selected text in the control.
SelectionFont	Specifies or retrieves the font of the selected text or at the insertion point.

Property	Description
SelectionLength	Specifies or retrieves the number of characters selected in the control.
Text	Specifies or retrieves the text in the control.
WordWrap	Determines whether the control automatically wraps the words towards the beginning of the next line whenever necessary.

Table 3.13: Properties of the RichTextBox Class

#### → Methods

The methods of `RichTextBox` class allow you to copy and paste the selected text and redo or undo an operation. Table 3.14 lists the most commonly used methods of the `RichTextBox` class.

Method	Description
AppendText	Adds text to the current text of a <code>RichTextBox</code> control.
Copy	Copies the selected text from the control to the clipboard.
Paste	Pastes the contents of the clipboard into the control.
Redo	Performs the last operation again that was undone in the control.
Undo	Undoes the last edit operation.
SelectAll	Selects all text in the control.

Table 3.14: Methods of the RichTextBox Class

#### → Events

Table 3.15 lists the most commonly used events of the `RichTextBox` class.

Event	Description
Click	Occurs when the control is clicked.
HScroll	Occurs when the user clicks the horizontal scroll bar of the control.
VScroll	Occurs when the user clicks the vertical scroll bar of the control.

Table 3.15: Events of the RichTextBox Class

The following code demonstrates the creation of a `RichTextBox` control.

#### Code Snippet:

```

RichTextBox rtbFeedback = new RichTextBox();
rtbFeedback.SelectedText = " This is a sample ";
rtbFeedback.SelectionColor = Color.Red;
rtbFeedback.SelectedText = " RichTextBox control ";
rtbFeedback.WordWrap = false;

private void rtbFeedback_TextChanged(object sender, EventArgs e)
{
    rtbFeedback.WordWrap = true;
}

```

The code displays a `RichTextBox` control that displays the text 'This is a sample `RichTextBox` control'. The `SelectionColor` property is used to set the color of the selected text to Red. The red color is applied to the selected text, 'RichTextBox control'. The `WordWrap` property is set to `false`, which means that the words will not be wrapped in the control. When the text in the control changes, the `TextChanged` event occurs and the `WordWrap` property is set to `true`. Figure 3.4 displays the `RichTextBox` control.

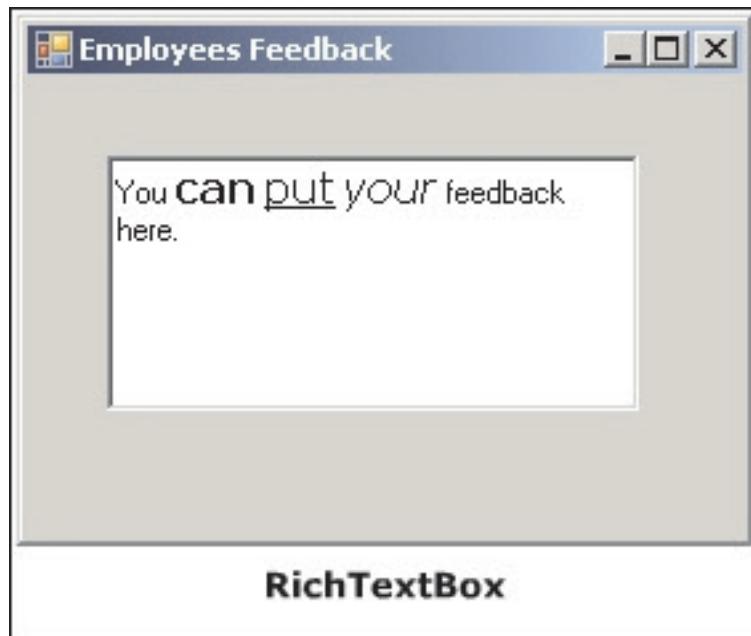


Figure 3.4: `RichTextBox` Control

### 3.3.2 Font and FontStyle Classes

You can specify the font, font style, and font size for the text in the `RichTextBox` control using the `SelectionFont` property and the `Font` class. The `Font` class exists in `System.Drawing` namespace. The `FontStyle` enumeration is used to specify the style of the font such as bold, italics, and underline. The following code demonstrates how to apply a font, font style, and font size to the text in the `RichTextBox` control.

#### Code Snippet:

```
RichTextBox rtbFeedback = new RichTextBox();
rtbFeedback.Text = "You can put your feedback here.";
rtbFeedback.Select(4, 3);
rtbFeedback.SelectionFont = new Font("Verdana", 12, FontStyle.Bold);
rtbFeedback.Select(8, 3);
rtbFeedback.SelectionFont = new Font("Verdana", 12, FontStyle.Underline);
rtbFeedback.Select(12, 4);
rtbFeedback.SelectionFont = new Font("Verdana", 12, FontStyle.Italic);
```

The code creates an instance of the `RichTextBox` control. The `Text` property displays the specified text in the control.

The `Select()` method selects three characters from the fourth position within the text in the control. The `SelectionFont` property is used to set the font, its size, and style by invoking the constructor of the `Font` class. The `FontStyle` enumeration is used to specify the font style. Different strings are selected and are made bold, italic, and underlined.

### Knowledge Check 3

- Can you match the properties, methods, and events of the `RichTextBox` control against their corresponding descriptions?

Description		Property, Method, and Event	
(A)	This event occurs when the value of <code>Text</code> property value is modified.	(1)	<code>SelectAll</code>
(B)	This method selects all text in the control.	(2)	<code>TextChanged</code>
(C)	This method adds text after the current text to the control.	(3)	<code>Text</code>
(D)	This property specifies or retrieves the selected text within the control.	(4)	<code>AppendText</code>
(E)	This property specifies or retrieves the text in the control.	(5)	<code>SelectedText</code>

- You want to display a rich text box on a form that allows users to enter their suggestion text. Which one of the following codes will help you to achieve this?

(A)	<pre>RichTextBox rtbSuggestion = new RichTextBox(); rtbSuggestion.Text = "Need to improve the attendance process"; rtbSuggestion.Width = 500; rtbSuggestion.MaxLength = 100; rtbSuggestion.WordWrap(true); rtbSuggestion.Clear(); Controls.Add(rtbSuggestion);</pre>
(B)	<pre>RichTextBox rtbSuggestion = new RichTextBox(); rtbSuggestion.Text = "Need to improve the attendance process"; rtbSuggestion.Width = 500; rtbSuggestion.MaxLength = 100; rtbSuggestion.WordWrap = true; rtbSuggestion.Clear(); Controls.Add(rtbSuggestion);</pre>

(C)	<pre>RichTextBox rtbSuggestion = new RichTextBox(); rtbSuggestion.Text = "Need to improve the attendance process"; rtbSuggestion.Width = 500; rtbSuggestion.MaxLength = 100; rtbSuggestion.WordWrap(true); rtbSuggestion.Clear; Controls.Add(rtbSuggestion);</pre>
(D)	<pre>RichTextBox rtbSuggestion = new RichTextBox(); rtbSuggestion.Text = "Need to improve the attendance process"; rtbSuggestion.Width = 500; rtbSuggestion.MaxLength = 100; rtbSuggestion.WordWrap = true; rtbSuggestion.Clear = true; Controls.Add(rtbSuggestion);</pre>

### 3.4 *ProgressBar Control*

In this fourth lesson, **ProgressBar Control**, you will learn to:

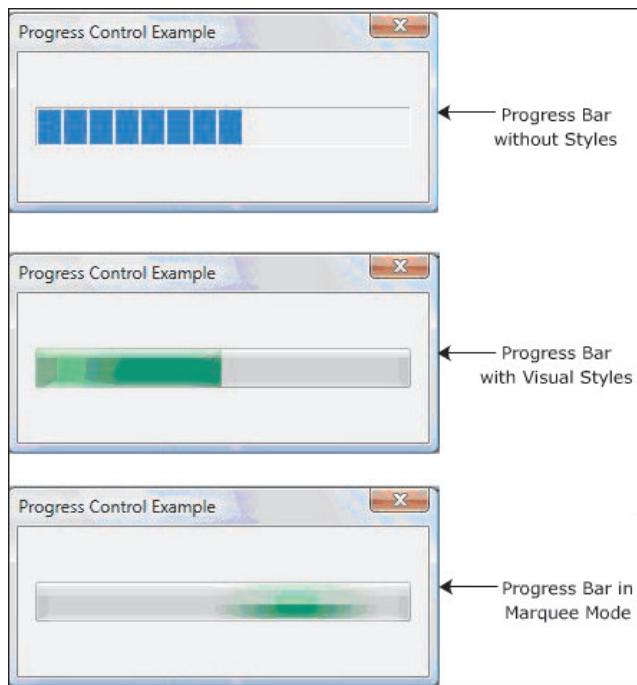
- Describe the need for **ProgressBar** control and state its use.
- Explain the properties, and methods of **ProgressBar** control.

#### 3.4.1 *ProgressBar Control*

The **ProgressBar** control is a window that can be used to indicate the progress of an operation in an application. The window consists of a rectangular area within which animated vertical bars appear indicating the progress of the operation. This gives you an idea about how long will it take for the process to get complete.

The **ProgressBar** control can be implemented with or without using visual styles. These visual styles implement the current theme and visual effects of the operating system. The **ProgressBar** control can also, be animated in such a way that the proportion of the task that is complete is not indicated in the rectangle. This type of progress bar is called **Marquee ProgressBar**. It only indicates that the operation is in process but it does not indicate its progress.

Figure 3.5 displays the `ProgressBar` control.



**Figure 3.5: ProgressBar Control**

The following syntax is used to create the `ProgressBar` control.

**Syntax:**

```
ProgressBar <objectName> = new ProgressBar();
```

where,

`ProgressBar`: Is the built-in class used to create the `ProgressBar` control.

The following code is used to create the `ProgressBar` control.

**Code Snippet:**

```
ProgressBar pbrSaving = new ProgressBar();
```

### 3.4.2 Properties and Methods

The `ProgressBar` class is used to create the `ProgressBar` control and it provides various properties, methods, and events.

Table 3.16 lists the most commonly used properties and methods of the `ProgressBar` class.

Name	Description
Increment	This method advances the current position of the progress bar by the specified value.
Maximum	This property specifies or retrieves the maximum value of the range of the control.
Minimum	This property specifies or retrieves the minimum value of the range of the control.
PerformStep	This method advances the current position of the progress bar by the value specified by the <code>Step</code> property.
Step	This property specifies or retrieves a value by which a call to the <code>PerformStep()</code> method will increase the current position of a <code>ProgressBar</code> control.
Style	This property specifies or retrieves the way in which progress should be indicated on a progress bar.
Value	This property specifies or retrieves the current position of the progress bar.

**Table 3.16: Methods and Properties of the `ProgressBar`**

The following code demonstrates the `ProgressBar` control.

**Code Snippet:**

```
ProgressBar pbrSaving = new ProgressBar();
pbrSaving.Maximum = 100;
pbrSaving.Minimum = 1;
pbrSaving.Style = ProgressBarStyle.Continuous;
pbrSaving.Increment(1);
```

The code creates the `ProgressBar` control. The `Maximum` and `Minimum` property sets the maximum and minimum value of the range of the control. The `Style` property sets the progress to be shown as `Continuous`, which is a value of the `ProgressBarStyle` enumeration. The `Increment()` method increases the current position of the control by 1.

## Knowledge Check 4

- You want to display a progress bar on a particular form. Which one of the following codes will help you to achieve this?

(A)

```
ProgressBar pbrProgress = new ProgressBar();
pbrProgress.Maximum = 0;
pbrProgress.Minimum = 30;
pbrProgress.Step = 10;
pbrProgress.PerformStep = 2;
pbrProgress.Increment();
```

(B)	<pre>ProgressBar pbrProgress = new ProgressBar(); pbrProgress.Maximum = 30; pbrProgress.Minimum = 0; pbrProgress.Step = 50; pbrProgress.PerformLayout = 2; pbrProgress.Increment();</pre>
(C)	<pre>ProgressBar pbrProgress = new ProgressBar(); pbrProgress.Maximum = 30; pbrProgress.Minimum = 0; pbrProgress.Step = 10; pbrProgress.PerformStep(); pbrProgress.Increment(2);</pre>
(D)	<pre>ProgressBar pbrProgress = new ProgressBar(); pbrProgress.Maximum = 30; pbrProgress.Minimum = 0; pbrProgress.Step = 10; pbrProgress.PerformLayout(); pbrProgress.Increment();</pre>

2. Which of these statements about the `ProgressBar` control are false?

(A)	The <code>Value</code> property gets or sets the minimum and maximum values of the <code>ProgressBar</code> control.
(B)	The <code>Increment</code> method sets the current position of the <code>ProgressBar</code> control.
(C)	The <code>Step</code> property increments the current position of the <code>ProgressBar</code> by the specified value.
(D)	The <code>Marquee</code> <code>ProgressBar</code> gives an idea about how long the process will take to complete.
(E)	The <code>ProgressBar</code> control implements using visual styles.

## Module Summary

In this module, **Advanced Controls**, you learnt about:

### → **SelectionList Controls**

Selection list controls allow you to select a value from a range of values. The two types of selection list controls are NumericUpDown and DomainUpDown control. The NumericUpDown control allows you to select numeric values from the specified range, while the DomainUpDown control selects text values.

### → **ListView and TreeView Controls**

ListView control allows you to display a collection of items similar to the right pane of the Windows Explorer. The five main views of the ListView control are Tile, List, Details, SmallIcon and LargeIcon. The TreeView control is used to display hierarchical data similar to the left pane of the Windows Explorer. The three types of nodes of the TreeView control are Root, Parent, and Leaf.

### → **RichTextBox Control**

The RichTextBox control is used for entering, formatting, and manipulating text. It provides various options for displaying and loading text, embedding images from a file, and performing undo and redo operations.

### → **ProgressBar Control**

The ProgressBar control gives you an idea about the progress of an operation in an application. The marquee progress bar shows action without indicating the completion percentage of the task. This type of progress bar indicates that the operation is in process but it does not indicate its progress.

# ASK to LEARN

**Questions**  
*in your*  
**mind?**



**are here to HELP**

Post your questions in the **ASK to LEARN** section  
for solutions.

# Module - 4

## Date Controls and Timer Component

Welcome to the Module, **Date Controls and Timer Component**.

Date controls allow you to select the required date and time and validates the input to avoid any invalid values. The MonthCalendar control allows you to select multiple dates and to display multiple months at a time. The Timer component is similar to a stop watch that allows you to track the time for a particular task.

In this Module, you will learn about:

- ➔ Date Controls
- ➔ MonthCalendar Control
- ➔ Timer Component

## 4.1 Date Controls

In this first lesson, **Date Controls**, you will learn to:

- Explain the need for Date and time controls.
- Describe the DateTimePicker control.
- Describe the properties, methods, and events of DateTimePicker control.
- Explain how to display date using CustomFormat property.
- Describe how to display time with DateTimePicker control.

### 4.1.1 Purpose

Consider a scenario where a manufacturing firm wants to automate the process of ordering raw materials by managing order details. One of the requirements of the firm is that the application must be able to accept and display the accurate order and delivery dates for a particular order. This will help in avoiding manual errors in typing dates. The developer can achieve this task using the date controls provided by Microsoft Visual Studio 2005.

Date and time controls are used to select or display date and time. These controls provide a calendar in a graphical format, which allows you to easily select a particular date. This helps in avoiding manual mistakes for date and time by restricting the user from entering any invalid date.

There are two types of date controls. They are as follows:

- DateTimePicker
- MonthCalendar

### 4.1.2 DateTimePicker Control

The DateTimePicker control allows you to select an item from a list of dates or times. It appears as a text box along with a drop-down calendar and displays the current date in a particular format. The calendar allows you to view the days of a week or browse through the months, as if you are turning the pages of a calendar.

Apart from choosing the date from the calendar, the control allows you to type the date in its text box region. However, for both the ways of entering date, the date is always validated to check for any invalid inputs.

The following syntax is used to create the DateTimePicker control.

#### Syntax:

```
DateTimePicker <objectName> = new DateTimePicker();
```

where,

`DateTimePicker`: Is the built-in class used to create the `DateTimePicker` control.

The following code creates the `DateTimePicker` control named, `dtpOrderDate`.

#### Code Snippet:

```
DateTimePicker dtpOrderDate = new DateTimePicker();
```

**Note** - The `DateTimePicker` control displays the date according to the computer's settings.

The `DateTimePicker` class is used to create the `DateTimePicker` control and it provides various properties, methods, and events.

#### → Properties

The properties of the `DateTimePicker` class allow you to set the date format, minimum date and maximum date. Table 4.1 lists the commonly used properties of the `DateTimePicker` class.

Property	Description
CustomFormat	Specifies or retrieves the custom date or time format string.
Format	Specifies or retrieves the format of date and time displayed in the control.
MinDate	Specifies or retrieves the minimum date and time that you can select in the control.
MaxDate	Specifies or retrieves the maximum date and time that you can select in the control.
ShowCheckBox	Specifies or retrieves a value, which indicates whether the <code>CheckBox</code> control should appear at the left side of the date, which is selected.
Value	Specifies or retrieves the date or time assigned to the control.

Table 4.1: Properties of the `DateTimePicker` Class

#### → Methods

Table 4.2 lists the most commonly used methods of the `DateTimePicker` class.

Method	Description
Focus	Sets the input focus on a <code>DateTimePicker</code> control.
Show	Displays a <code>DateTimePicker</code> control on the form.

Table 4.2: Methods of the `DateTimePicker` Class

→ Events

The events of the `DateTimePicker` class allow you to handle the events, which occur when the user interacts with the control. Table 4.3 lists the most commonly used events of the `DateTimePicker` class.

Event	Description
Click	Occurs when the user clicks the control.
FormatChanged	Occurs when the value of the <code>Format</code> property is changed.
ValueChanged	Occurs when the value of the <code>Value</code> property is changed.

Table 4.3: Events of the `DateTimePicker` Class

The following code demonstrates how to use the `DateTimePicker` class.

**Code Snippet:**

```
DateTimePicker dtpOrderDate = new DateTimePicker();
dtpOrderDate.MinDate = new DateTime(1980, 01, 01);
dtpOrderDate.MaxDate = new DateTime(2008, 12, 31);

private void dtpOrderDate_ValueChanged (object sender, EventArgs e)
{
    MessageBox.Show(dtpOrderDate.Text.ToString());
}
```

The code creates an object of the `DateTimePicker` class and the `MinDate` and `MaxDate` properties are set to two different dates. When the user selects or changes the date, the `Value` property changes, which trigger the `ValueChanged` event. This leads to the execution of the `dtpOrderDate_ValueChanged` event handler which displays the date in a message box. Figure 4.1 displays the `DateTimePicker` control.

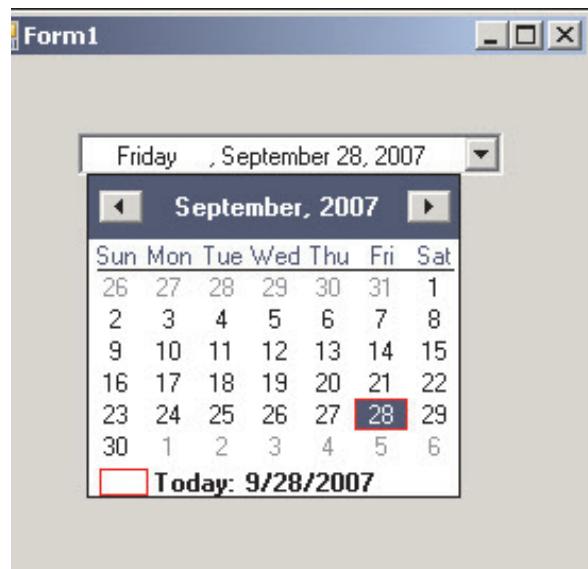


Figure 4.1: `DateTimePicker` Control

### 4.1.3 CustomFormat Property

The `CustomFormat` property of `DateTimePicker` class is used to specify different user-defined formats for displaying the date and time. There are various date and time format strings that allow you to specify the format for date and time.

Table 4.4 displays the commonly used format strings of `CustomFormat` property.

Format String	Description	Example
d	Displays one- or two-digit day.	If the day is 8, this format will display the day as '8' instead of '08'.
h	Displays one- or two-digit hour in 12-hour format.	If the time is 1:10 PM, this format will display the hour as '1' instead of '13'.
H	Displays one- or two-digit hour in 24-hour format.	If the time 1:30 PM, this format will display the hour as '13' instead of '1'.
m	Displays one- or two-digit minute.	If the time 2:05 PM, this format will display the minutes as '5' instead of '05'.
M	Displays one- or two-digit number indicating the month.	If the month is 9, this format will display the month as '9' instead of '09'.
s	Displays one- or two-digit seconds.	If the time is 1:45:02 P.M, this format will display the seconds as '2' instead of '02'.
y	Displays one-digit year.	If the year is 2008, this format will display the day as '8' instead of '2008'.

Table 4.4: Formatting Strings of `CustomFormat` Property

The following snippet displays the date using `CustomFormat` property.

#### Code Snippet:

```
DateTimePicker dtpStartingDate = new DateTimePicker();
dtpStartingDate.Format = DateTimePickerFormat.Custom;
dtpStartingDate.CustomFormat = "d-M-yyyy";
```

The code uses the `CustomFormat` property to set the date in the user-defined format. Suppose, if the date is 15th August, 2007; the control will display the date as 15- 8- 2007 as per the specified format. The month and year will be displayed after leaving a space.

**Note** - You must set the value of the `Format` property to `DateTimePickerFormat.Custom` for this property to affect the formatting of the displayed date and time.

### 4.1.4 Format Property

The `Format` property of `DateTimePicker` control allows you to specify the type of format for displaying the date and time. This format could be the standard format or the custom format specified by the user. There are four different formats defined in the `DateTimePickerFormat` enumeration, which you can specify for the `Format` property.

Table 4.5 displays the various options defined in the `DateTimePickerFormat` enumeration.

Name	Description
Custom	Used to display the date or time value in custom format.
Long	Used to display the date or time value in the long date format, which is set by the user's operating system.
Short	Used to display the date or time value in the short date format, which is set by the user's operating system.
Time	Used to display the date or time value in the time format, which is set by the user's operating system.

Table 4.5: `DateTimePickerFormat` Enumeration

The following snippet displays the time in a `DateTimePicker` control using the `Format` property.

**Code Snippet:**

```
DateTimePicker dtpDeliveryTime = new DateTimePicker();
dtpDeliveryTime.Format = DateTimePickerFormat.Time;
```

The code uses the `Format` property to display only time in the `DateTimePicker` control. To do so, the `DateTimePickerFormat` enumeration is used.

**Note** - You must set the value of the `Format` property to `DateTimePickerFormat.Custom` for the `CustomFormat` property to affect the formatting of the displayed date and time.

## Knowledge Check 1

1. Which of these statements about Date controls are true?

(A)	The <code>DateTimePicker</code> control is used to select a date and time from a specified range of dates and times.
(B)	The <code>DateTimePickerFormat</code> is used to specify a date format for the <code>DateTimePicker</code> control.
(C)	The <code>CustomFormat</code> property is used to display only date in various formats.
(D)	The <code>Format</code> property of the <code>DateTimePicker</code> control is used to display standard and custom date formats.
(E)	The <code>Short</code> value of the <code>Format</code> property is used to display date in dd-mm-yyyy format.

2. Can you match the format strings of the CustomFormat property against their corresponding descriptions?

Description			Format String
(A)	Displays time representing 24 hour format.	(1)	h
(B)	Displays the number indicating the month.	(2)	m
(C)	Displays the number indicating minutes.	(3)	M
(D)	Displays the starting letter of A.M. or P.M.	(4)	H
(E)	Displays time representing 12 hour format.	(5)	t

## 4.2 MonthCalendar Control

In this second lesson, **MonthCalendar Control**, you will learn to:

- Describe the need for MonthCalendar control and its features.
- List and describe the properties, methods, and events of MonthCalendar control.
- Explain how to change the appearance of MonthCalendar control.
- Describe how to display specific days in bold using MonthCalendar control.
- Describe how to display more than one month using MonthCalendar control.

### 4.2.1 MonthCalendar Control

The MonthCalendar control allows you to view and select a date with the help of a graphical calendar. The calendar appears as drop-down grid of rows and columns displaying the dates of a particular month. The MonthCalendar control is similar to the DateTimePicker control, but allows you to select multiple dates simultaneously. Unlike the DateTimePicker control, the MonthCalendar control cannot display time and does not allow you to type dates.

The following syntax is used to create the MonthCalendar control.

#### Syntax:

```
MonthCalendar <objectName> = new MonthCalendar();
```

where,

MonthCalendar: Is the built-in class used to create the MonthCalendar control.

The following code creates the MonthCalendar control named, calMonth.

#### Code Snippet:

```
MonthCalendar calMonth = new MonthCalendar();
```

**Note** - You have used the MonthCalendar control while setting the date and time for the system using the Date/Time Properties dialog box.

Figure 4.2 displays the MonthCalendar control.

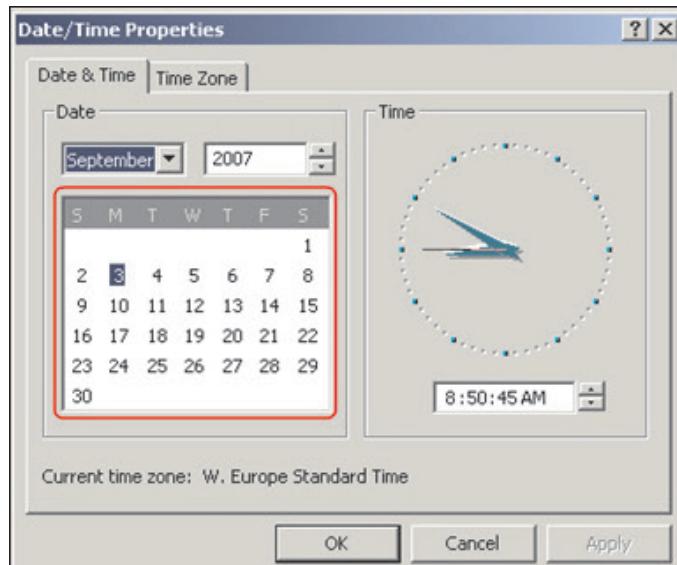


Figure 4.2: MonthCalendar Control

The MonthCalendar class is used to create the MonthCalendar control and it provides various properties, methods, and events.

#### → Properties

The properties of the MonthCalendar class allow you to specify and retrieve the number of rows and columns for the calendar, today's date, and week numbers. Table 4.6 displays the commonly used properties of the MonthCalendar class.

Property	Description
CalendarDimensions	Specifies or retrieves the total number of rows and columns of the months appearing in the calendar.
ShowToday	Specifies or retrieves a value indicating whether the date value in the TodayDate property will appear at the bottom of the control.
ShowTodayCircle	Specifies or retrieves a value, indicating whether today's date is circled.
ShowWeekNumbers	Specifies or retrieves a value indicating whether a month calendar control displays the week numbers (1-52) to the left of each row of days.
TodayDate	Specifies or retrieves a value that is used as the current date by the month calendar control.

Table 4.6: Properties of the MonthCalendar Class

## → Methods

The methods of the `MonthCalendar` class allow you to add and remove bolded dates month-wise and year-wise. Table 4.7 displays the commonly used methods of the `MonthCalendar` class.

Method	Description
<code>AddAnnuallyBoldedDate</code>	Adds a day that is displayed in bold in the month calendar control on an annual basis.
<code>AddBoldedDate</code>	Adds a day to be displayed in bold in the month calendar control.
<code>AddMonthlyBoldedDate</code>	Adds a day that is displayed in bold in the month calendar control on a monthly basis.
<code>RemoveAnnuallyBoldedDate</code>	Removes the given date from the list of annual bold dates.
<code>RemoveBoldedDate</code>	Removes the given date from the list of nonrecurring bold dates.
<code>RemoveMonthlyBoldedDate</code>	Removes the given date from the list of monthly bold dates.
<code>SetDate</code>	Specifies a date as the currently selected date.
<code>UpdateBoldedDates</code>	Repaints the dates in bold to reveal the date set in the list of bold dates.

Table 4.7: Methods of the `MonthCalendar` Class

## → Events

Table 4.8 displays the commonly used events of the `MonthCalendar` class.

Event	Description
<code>Click</code>	Occurs when the user clicks the control.
<code>DateChanged</code>	Occurs when the selected date in the month calendar control is changed.
<code>DateSelected</code>	Occurs when the user selects a date in the month calendar control using the mouse.

Table 4.8: Events of the `MonthCalendar` Class

The following code demonstrates the properties, methods, and events of the `MonthCalendar` class.

### Code Snippet:

```
MonthCalendar calBirthDate = new MonthCalendar();
calBirthDate.ShowToday = true;
calBirthDate.ShowWeekNumbers = true;
calBirthDate.AddBoldedDate(new DateTime(2007, 08, 15));
```

```
private void calBirthDate_DateSelected(object sender,
DateRangeEventArgs e)
{
    MessageBox.Show("You have selected " + calBirthDate.SelectionEnd.
Day);
}
```

The code creates an object of the MonthCalendar class. The ShowToday property is used to display today's date at the bottom of the control. The ShowWeekNumbers property is used to display the week numbers from 1 to 52 at the left side of the rows. The AddBoldedDate() method is used to add and display the date 15th August, 2007 in bold. When the selected date in the MonthCalendar control is changed, the DateSelected event is raised and a message box is displayed with the selected date. This is done using the SelectionEnd property, which is used to retrieve the last selected date from the range of selected dates.

#### 4.2.2 Appearance of MonthCalendar Control

The MonthCalendar class provides various properties to change the appearance of the control. Table 4.9 lists the properties used to change the appearance of the control.

Property	Description
TitleBackColor	Specifies or retrieves a value that indicates the background color of the title area of the calendar.
TitleForeColor	Specifies or retrieves a value that indicates the foreground color of the title area of the calendar.
TrailingForeColor	Specifies or retrieves a value, which indicates the color of days in months which are partially displayed in the control.
ShowTodayCircle	Specifies or retrieves a value that indicates today's date with a circle.

Table 4.9: Properties of the MonthCalendar Control

The following code is used to change the appearance of MonthCalendar control.

##### Code Snippet:

```
MonthCalendar calBankCalendar = new MonthCalendar();
calBankCalendar.TitleBackColor = Color.Blue;
calBankCalendar.TrailingForeColor = Color.Green;
calBankCalendar.TitleForeColor = Color.Yellow;
```

The code uses three different properties to change the appearance of the MonthCalendar control. The background color of the title area of the calendar is changed to blue color using the TitleBackColor property. The fore color for the trailing dates in the calendar is set to green using the TrailingForeColor property. The title of the calendar is set to yellow using the TitleForeColor property.

### 4.2.3 Displaying Days in *BOLD*

The MonthCalendar control allows you to display the days in bold to draw the attention of the user towards special dates, such as holidays and weekends. To do so, the MonthCalendar class provides you with `BoldedDates`, `AnnuallyBoldedDates`, and `MonthlyBoldedDates` properties.

The `BoldedDates` property contains single dates in bold. The `AnnuallyBoldedDates` property sets or retrieves the dates appearing in bold every year and the `MonthlyBoldedDates` property sets or gets the dates appearing in bold every month.

The following code is used to display a specific day in bold using MonthCalendar control.

#### Code Snippet:

```
MonthCalendar calVacation = new MonthCalendar();
DateTime myVacation1 = new DateTime(2007, 8, 10);
DateTime myVacation2 = new DateTime(2007, 8, 3);
DateTime[] vacationDates = { myVacation1, myVacation2 };
calVacation.BoldedDates = vacationDates;
```

The code creates two objects of the `DateTime` class and takes the date in `yyyy-M-d` format as the parameter. The objects are stored in the `DateTime` array, `vacationDates`. The `BoldedDates` property of the MonthCalendar class is used to display the dates in bold. Figure 4.3 displays the days in bold.

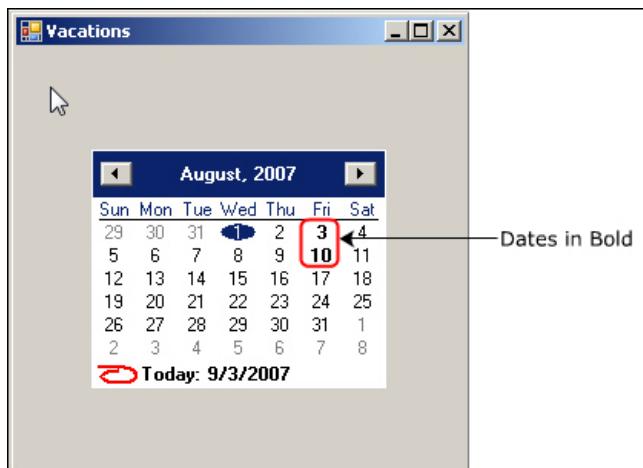
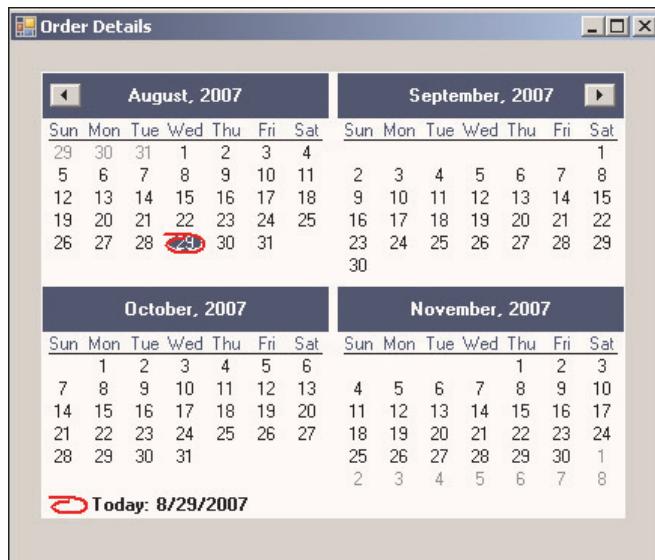


Figure 4.3: Days in Bold

### 4.2.4 Displaying Multiple Months

The MonthCalendar control by default, displays only one month. You can specify the number of months to be displayed at a time and how to arrange them within the control. To display multiple months, the MonthCalendar class provides you with the `CalendarDimensions` property to set the number of months to be displayed horizontally and vertically.

Figure 4.4 displays the multiple months.



**Figure 4.4: Multiple Months**

The following code is used to display four months in MonthCalendar control using the `CalendarDimensions` property.

**Code Snippet:**

```
MonthCalendar calCompanyCalendar = new MonthCalendar();
calCompanyCalendar.CalendarDimensions = new Size(2, 2);
```

The code uses the `CalendarDimensions` property to set the dimension of the calendar. The constructor of the `Size` class is invoked that takes two parameters. The first parameter indicates the number of columns and the second parameter indicates the number of rows. Two months will be displayed in the first row and the remaining two will be displayed in the second row.

**Note** - There should be enough space on the form while changing the calendar dimensions, as the control is resized.

## Knowledge Check 2

- Can you match the properties, methods, and events of the MonthCalendar control against their corresponding descriptions?

Description			Property, Method, and Event
(A)	Event that is raised when a date is selected using the mouse.	(1)	UpdateBoldedDates
(B)	Property used to set the background color of the title region.	(2)	DateSelected
(C)	Property used to identify the date with a circle.	(3)	TitleBackColor

Description		Property, Method, and Event	
(D)	Method used to redraw the dates in bold to view the list of bolded dates.	(4)	AddAnnuallyBoldedDate
(E)	Method used to display a date in bold on an annual basis.	(5)	ShowTodayCircle

2. You want to display six calendars at a time, each having three columns and two rows. Which one of the following codes will help you to achieve this?

(A)	MonthCalendar calUniversityCalendar = new MonthCalendar(); calUniversityCalendar.CalendarDimensions = new Size(2, 3);
(B)	MonthCalendar calUniversityCalendar = new MonthCalendar(); calUniversityCalendar.CalendarDimensions = new Size(3, 2);
(C)	MonthCalendar calUniversityCalendar = new MonthCalendar(); calUniversityCalendar.CalendarDimensions = new MonthCalendar(2, 3);
(D)	MonthCalendar calUniversityCalendar = new MonthCalendar(); calUniversityCalendar.CalendarDimensions = new MonthCalendar(2, 3);

### 4.3 Timer Component

In this third lesson, **Timer Component**, you will learn to:

- State and describe the use of the Timer component.
- List and explain properties, methods, and events of Timer component.
- Identify the limitations of the Interval property.

#### 4.3.1 Timer Component

The Timer component is a component that generates an event at regular intervals. This is similar to a stopwatch, which is used during the running competition to identify the time taken by each player to reach the destination.

For example, you can use the component to set a time within which the user has to complete answering the required number of questions. When the specified time interval is over, you can perform certain tasks such as asking the user to stop answering the questions. There are various uses for timer component. These are as follows:

- Setting an interval between two events
- Monitoring the time taken to complete a particular task

- Firing a particular action after a given time

The following syntax is used to create the component using the two constructors of the `Timer` class.

**Syntax:**

```
Timer <objectName> = new Timer();
Timer <objectName> = new Timer(IContainer container);
```

where,

`Timer`: Is the built-in class used to create the `Timer` component.

`IContainer`: Specifies the container that holds the `Timer` component.

For example, the form on which the component exists becomes the container of the control.

The following code creates a timer component in the current form.

**Code Snippet:**

```
Timer tmrCounter = new Timer(this);
```

### 4.3.2 Properties, Methods, and Events

The `Timer` class is used to create the `Timer` component and it exists in `System.Windows.Forms` namespace. The class defines various properties, methods, and events used to set the time intervals the behavior of the control.

→ **Properties**

The properties of the `Timer` class allow you to set the time intervals between the occurrences of two events. Table 4.10 displays the commonly used properties of the `Timer` class.

Property	Description
<code>Enabled</code>	Specifies or retrieves a value indicating whether the timer is running.
<code>Interval</code>	Specifies or retrieves the time in milliseconds between the timer ticks.

Table 4.10: Properties of the Timer Class

→ **Methods**

The methods of the `Timer` class allow you to start and stop the timer. Table 4.11 displays the commonly used methods of the `Timer` class.

Method	Description
<code>Start</code>	Starts the timer.
<code>Stop</code>	Stops the timer.

Table 4.11: Methods of the Timer Class

→ **Events**

The event of the `Timer` class allows you to raise an event after a particular time interval. Table 4.12 displays the commonly used event of `Timer` class.

Event	Description
Tick	Occurs when a particular time interval is over and the <code>Enabled</code> property of the component is <code>true</code> .

**Table 4.12: Events of the Timer Class**

The following code demonstrates how to create and use a timer component.

**Code Snippet:**

```
Timer tmrCounter = new Timer();
tmrCounter.Interval = 1000;
tmrCounter.Enabled = true;
tmrCounter.Start();
private void tmrCounter_Tick(Object sender, EventArgs e)
{
    // write code to perform some action
}
```

The code creates a timer control using the `Timer` class. The `Interval` property is set to 1000 milliseconds (one second). The `Enabled` property is set to `true`. The `Start()` method is used to start the timer. The `Tick` event is raised after every one second.

### 4.3.3 Limitations of Interval Property

The `Interval` property of `Timer` component is used to set the number of milliseconds to raise the `Tick` event. However, there are a few limitations for `Interval` property. These limitations are as follows:

- If an application is making heavy requests to the system such as long loops or intensive calculations, the application may not get the timer events as frequently as specified in the `Interval` property.
- The `Interval` property can have a maximum interval of 64,767 milliseconds (about 64.8 seconds).
- The interval might not elapse exactly on time. Therefore, to ensure accuracy, the timer should check the system clock, instead of tracking the accumulated time internally.
- The system can only generate 18 clock ticks per second. Even if the value of the `Interval` property is in milliseconds, the exact precision of the component will be only one-eighteenth of a second approximately.

## Knowledge Check 3

1. Which of these statements about the `Timer` component are false?

(A)	The <code>Timer</code> component does not provide with the explicit functionality to stop the passage of time once it has started.
(B)	The <code>Start()</code> method starts the <code>Timer</code> component.
(C)	The <code>Interval</code> property supports only a maximum interval of 64,766 milliseconds.
(D)	The <code>Enabled</code> property indicates whether the <code>Timer</code> component is running.
(E)	The <code>Timer</code> component raises the <code>Ticks</code> event before the specified interval has been elapsed.

2. You want to display a message box stating that 'Five seconds are over' at every five second interval using a timer. Which one of the following codes will help you to achieve this?

(A)	<pre>Timer tmrCounter = new Timer(); tmrCounter.Interval = 5000; tmrCounter.Start();  private void tmrCounter_Ticks(Object sender, EventArgs e) {     MessageBox.Show("Five seconds are over"); }</pre>
(B)	<pre>Timer tmrCounter = new Timer(); tmrCounter.Interval = 500; tmrCounter.Start();  private void tmrCounter_Tick(Object sender, EventArgs e) {     MessageBox.Show("Five seconds are over"); }</pre>

(C)	<pre>Timer tmrCounter = new Timer();  tmrCounter.Interval = 500;  tmrCounter.Start();   (C) private void tmrCounter_Ticks(Object sender, EventArgs e) {      MessageBox.Show("Five seconds are over");  }</pre>
(D)	<pre>Timer tmrCounter = new Timer();  tmrCounter.Interval = 5000;  tmrCounter.Start();   (D) private void tmrCounter_Tick(Object sender, EventArgs e) {      MessageBox.Show("Five seconds are over");  }</pre>



## Module Summary

In this module, **Date Controls and Timer Component**, you learnt about:

→ **Date Controls**

Date controls are used to select or display a date and time in an application. By using these controls, you can restrict the user from entering invalid dates. The DateTimePicker control is used to display date and time in various formats with the help of Format and CustomFormat properties.

→ **MonthCalendar Control**

MonthCalendar control is used to display and select multiple dates. The MonthCalendar class provides various properties and methods to display specific dates in bold, multiple calendars, and to change appearance of the control.

→ **Timer Component**

Timer component is used to set a time interval after which the Tick event will be raised to notify about the elapsed time or to perform some tasks. The Timer class has many in-built properties and methods; which are used to specify the time interval, and start and stop the timer.

# Module - 5

## Dialog Boxes

Welcome to the Module, **Dialog Boxes**.

A dialog box is a special window, which can be used to interact with the user and display information. It acts as a container that can hold various controls to retrieve information. Dialog boxes are divided into two categories, common and custom. Common dialog boxes are system-defined dialog boxes and custom dialog boxes are user-defined dialog boxes.

In this Module, you will learn about:

- ➔ Types of Dialog Boxes
- ➔ System Defined Dialog Boxes
- ➔ User Defined Dialog Boxes
- ➔ Retrieving Information from a Dialog Box

## 5.1 XML Parsing

In this first lesson, **Types of Dialog Boxes**, you will learn to:

- Describe the need for dialog boxes.
- Explain the features of dialog boxes.
- Describe modal and modeless dialog boxes.

### 5.1.1 Introduction to Dialog Boxes

Consider a scenario where a developer wants to create an application for tracking details of students who have passed in a school. These details should include the student name, subjects and scores, year of passing, and the percentage scored. Scores and percentage greater than 80 should be accepted and displayed in a different font, size, style, and color. To accomplish this task, the developer can use dialog boxes.

A dialog box is a window that allows you to collect and display related information using various controls. It allows you to group common functionalities within a single window and explore them. For example, according to the scenario, the developer requires the **Font** dialog box that will provide all the options related to the font, size, style, and color.

**Note** - Some of the dialog boxes that you have used in Windows are **Save As**, **Color**, **Print**, and **Page Setup**.

#### → Elements

You can use a dialog box to take input or display some important information when the user is working with the application. A dialog box consists of:

- Title bar displaying caption and the close icon
- Instruction text informing what the user should do (optional)
- Content with controls to select or display information
- Action area having buttons and link labels
- Footnote area explaining about the window (optional)

### 5.1.2 Features

A dialog box is a special window, which is always invoked and accessed through forms. The window is

called a dialog box because it is a medium through which a dialogue takes place between the user and the application. A dialog box is different from forms because of some basic features.

These are as follows:

- **Non-resizable:** A dialog box is not resizable and cannot be maximized or minimized.
- **Usually Modal:** A dialog box is usually of modal type. This means that the user is not allowed to focus on any other window until the dialog box is closed.
- **Common Set of Buttons and Icons:** Generally, a dialog box consists of some standard buttons such as **OK** and **Cancel**. It also consists of the Help and Close icons on its upper-right corner.

### 5.1.3 Modal Dialog Box

Dialog boxes are displayed on top of the particular application window. There are two types of dialog boxes namely, modal and modeless.

A modal dialog box does not allow the user to activate any other window in the application unless the dialog box is closed. These dialog boxes are used when you want to take or display some critical information, while the user is working with the application. It is also used when some information is required to complete a particular task. For example, to save a Word document, you use the **Save As** dialog box to provide a filename and save the file. You cannot work with the Word document that is currently open unless you close the **Save As** dialog box. This means that the **Save As** dialog box is a modal dialog box. It prompts you to first save the document and then, continue working with it.

#### → Modeless Dialog Box

A modeless dialog box allows user to focus and work with the same application without closing the dialog box. A modeless dialog box allows you to switch between the application window and the dialog box. Modeless dialog boxes are used when you require them more frequently to perform repetitive tasks. For example, the **Find and Replace** dialog box of Microsoft Word is a modeless dialog box. You can search for the required words using this dialog box and can work with the document simultaneously.

### Knowledge Check 1

1. Which of the following statements regarding dialog boxes are true?

(A)	A dialog box is always modal.
(B)	The <b>Find</b> dialog box of Internet Explorer is a modal dialog box.
(C)	Modeless dialog boxes allow you to work with the application on which the dialog box is placed.
(D)	A dialog box is a window that cannot be maximized, minimized, or restored.
(E)	The <b>Font</b> dialog box in the Paint program is a modeless dialog box.

## 5.2 System-Defined Dialog Boxes

In this second lesson, **System-Defined Dialog Boxes**, you will learn to:

- Summarize the classification of dialog boxes.
- Describe the `ColorDialog` component.
- Explain the `FontDialog` component.
- Describe the `OpenFileDialog` and `SaveFileDialog` components.
- Explain the `PageSetupDialog` component.
- Explain the `PrintDialog` component.
- Describe the `PrintPreviewDialog` component.

### 5.2.1 Common and Custom Dialog Boxes

A dialog box can be classified into two types, namely, Common dialog box and Custom dialog box.

Common dialog boxes are system-defined dialog boxes such as **Open**, **Save**, and **Print**. These dialog boxes are reusable and hence, can be used across various applications.

Custom dialog boxes are user-defined dialog boxes. These dialog boxes are useful when the system-defined dialog boxes do not fulfill certain requirements.

#### → **CommonDialog Class**

The `CommonDialog` class is the base class for all common dialog boxes. This class exists in the `System.Windows.Forms` namespace. The `CommonDialog` class contains various properties, methods, and events that are inherited by other system-defined dialog boxes. Table 5.1 lists the most commonly used properties, methods, and events of the `CommonDialog` class.

Name	Description
Tag	This property specifies or retrieves an object that contains data about the control.
Reset	This method resets the properties of a common dialog to their default values.
ShowDialog	This method runs a common dialog box.
HelpRequest	This event occurs when the <b>Help</b> button of a common dialog box is clicked.

Table 5.1: CommonDialog Class Members

The following code demonstrates the `CommonDialog` class and its members.

#### Code Snippet:

```
CommonDialog cmndlgCommon = new ColorDialog();
cmndlgCommon.Reset();
cmndlgCommon.ShowDialog();

private void cmndlgCommon_HelpRequest(object sender, EventArgs e)
{
    MessageBox.Show(cmndlgCommon.Tag.ToString());
}
```

The code displays a **color** dialog box as a common dialog control. The `Reset()` method resets the properties of the **color** dialog box to its default values. The `ShowDialog()` method displays the **color** dialog box to the user. When the user clicks the **Help** button present in the **color** dialog box, the `HelpRequest` event is fired. This event displays a message that provides information about the **color** dialog box by using the `Tag` property.

### 5.2.2 *ColorDialog Component*

The `ColorDialog` component is a dialog box that allows you to select system-defined colors from the color palette. It also allows you to use custom colors by defining the RGB color values. The `ColorDialog` component is similar to the **Color** dialog box of Microsoft Word that allows you to select colors.

**Note** - RGB color model allows you to produce different colors by using the combination of red, green, and blue colors in various ways. Each of these three colors allows you to select values ranging from 0 to 256.

#### → **ColorDialog Class**

The `ColorDialog` class allows you to create the `ColorDialog` component.

Table 5.2 lists the most commonly used properties and methods in the `ColorDialog` class.

Name	Description
AllowFullOpen	This property specifies or retrieves a value that indicates whether the user can use the dialog box to define custom colors.
AnyColor	This property specifies or retrieves a value that indicates whether the dialog box can display all available colors in the set of basic colors.
Color	This property specifies or retrieves the selected color.
CustomColors	This property specifies or retrieves the set of custom colors that are displayed in the <b>color</b> dialog box.
Reset	This method resets all options to their default values. The last selected color is set to black and the custom colors are set to their default values.

Table 5.2: `ColorDialog` Class Members

Figure 5.1 displays the `ColorDialog`.

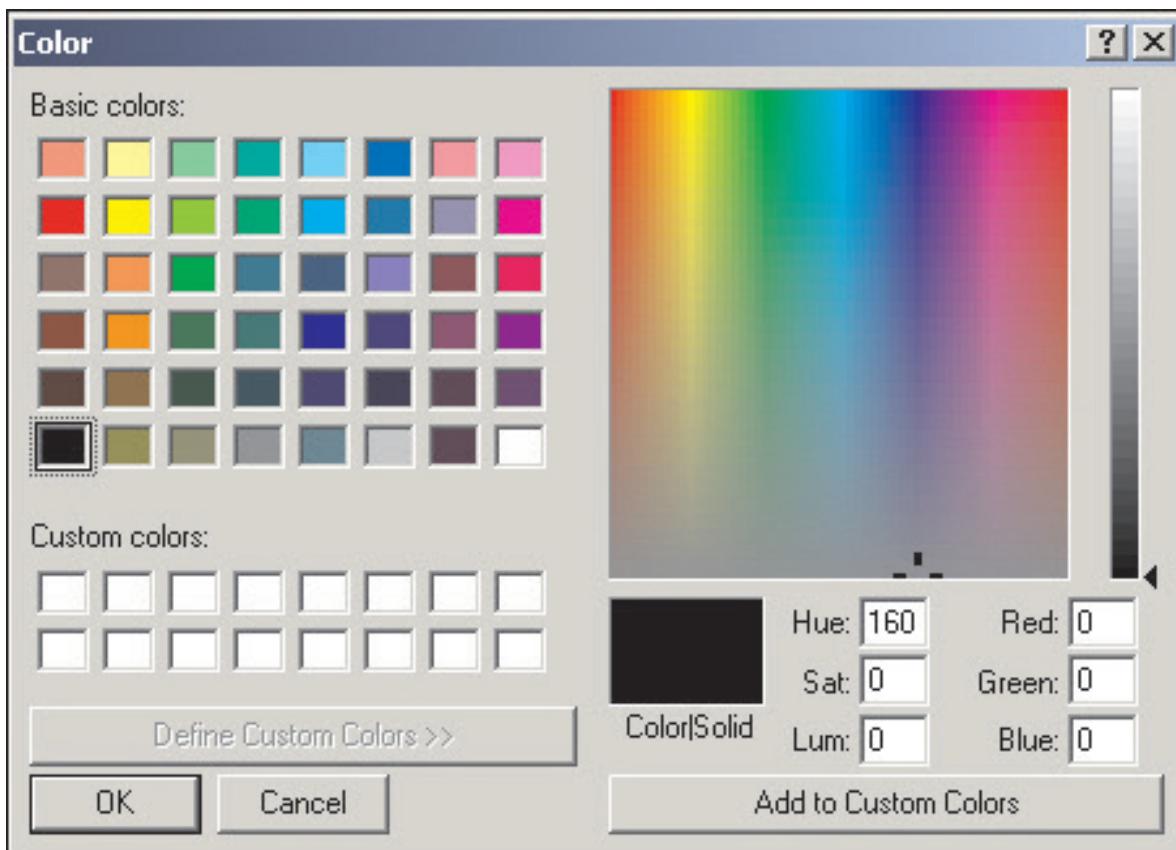


Figure 5.1: `ColorDialog`

The following code demonstrates the `ColorDialog` class and its members.

#### Code Snippet:

```
ColorDialog cdlgColor = new ColorDialog();
cdlgColor.Reset();
cdlgColor.AllowFullOpen = false;
cdlgColor.Color = Color.Red;
cdlgColor.AnyColor = true;
cdlgColor.ShowDialog();
```

The code creates an instance of the `ColorDialog` class. The `Reset()` method resets all the options of the dialog box to their default values. The `AllowFullOpen` property value is set to `false`, which indicates that the user cannot use the dialog box to define custom colors. The default color is set to `Red` by using the `Color` property. The dialog box will display all available colors in the set of basic colors by setting the `AnyColor` property to `true`.

**Note** - Once the `ColorDialog` component has been added to a form, it appears in the tray at the bottom of the Windows Forms Designer.

### 5.2.3 *FontDialog Component*

The `FontDialog` component allows you to select different fonts that are installed in the system. This component can also be used to modify the appearance of the text in an application. It displays a list of various fonts, font styles, and sizes. It also allows you to specify various other effects, such as subscripts, superscripts, shadows, and so on.

#### → **FontDialog Class**

The `FontDialog` class allows you to create the `FontDialog` component. Table 5.3 lists the most commonly used properties and events of the `FontDialog` class.

Name	Description
Color	This property specifies or retrieves the selected font color.
Font	This property specifies or retrieves the selected font.
MaxSize	This property specifies or retrieves the maximum point size of the font that a user can select.
MinSize	This property specifies or retrieves the minimum point size of the font that a user can select.
ShowApply	This property specifies or retrieves a value that indicates whether the dialog box contains the <b>Apply</b> button.
ShowEffects	This property specifies or retrieves a value that indicates whether the dialog box allows you to specify strikethrough, underline, and text color options.
Apply	This event occurs when the <b>Apply</b> button in the <code>font</code> dialog box is clicked.

Table 5.3: `FontDialog` Class Members

Figure 5.2 displays the `FontDialog`.

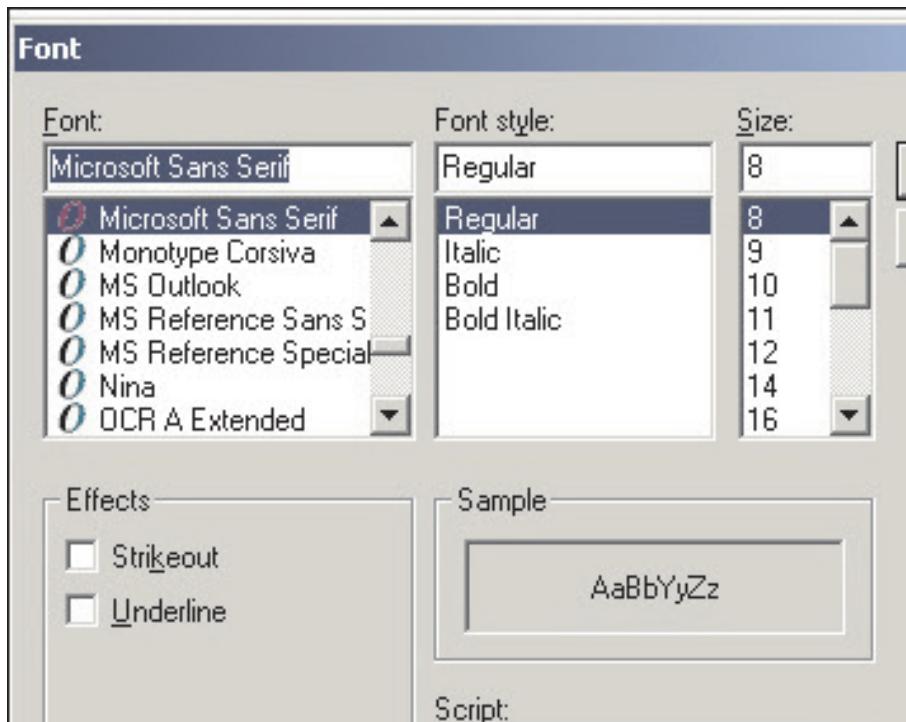


Figure 5.2: `FontDialog`

The following code demonstrates the `FontDialog` class and its members.

#### Code Snippet:

```
FontDialog fdlgFont = new FontDialog();
fdlgFont.Color = Color.Red;
fdlgFont.Font = new Font("Verdana", 18);
fdlgFont.MaxSize = 20;
fdlgFont.MinSize = 5;
fdlgFont.ShowApply = true;
fdlgFont.ShowEffects = false;
fdlgFont.ShowDialog();

private void fdlgFont_Apply(object sender, EventArgs e)
{
    // Write code to perform some action.
}
```

The code creates an instance of the `FontDialog` class. The default font color is set to Red by using the `Color` property. The `Font` property specifies the font as `Verdana` and size to 18. The `MaxSize` and `MinSize` properties set the maximum and minimum point size of the font to 20 and 5 respectively. The **Apply** button is displayed on the control by setting the `ShowApply` property to true. The `ShowEffects` property is set to false, which hides the strikethrough, underline, and text color options. When the user clicks the **Apply** button, the `Apply` event is fired.

**Note** - Once the `FontDialog` component has been added to a form, it appears in the tray at the bottom of the Windows Forms Designer.

### 5.2.4 OpenFileDialog and SaveFileDialog Components

The `OpenFileDialog` and `SaveFileDialog` are the most commonly used dialog boxes in applications. It allows you to save a file. You have used these dialog boxes in Microsoft Word while opening the file and saving it.

#### → OpenFileDialog Class

The `OpenFileDialog` class allows you to create the `OpenFileDialog` component. Table 5.4 lists the most commonly used properties, methods, and events of `OpenFileDialog` class.

Name	Description
<code>CheckFileExists</code>	This property specifies or retrieves a value that indicates whether the dialog box displays a warning when you specify a non-existing file name.
<code>FileName</code>	This property specifies or retrieves a string that contains the selected file name.
<code>Filter</code>	This property specifies or retrieves the current file name string that allows you to determine the choices appearing in the 'Save as file type' or 'Files of type' box.
<code>MultiSelect</code>	This property specifies or retrieves a value that indicates whether you can select multiple files.
<code>Title</code>	This property specifies or retrieves the title of the <code>file</code> dialog box.
<code>OpenFile</code>	This method opens the selected file with read-only permission.
<code>FileOk</code>	This event occurs when the Open or Save button on a <code>file</code> dialog box is clicked.

Table 5.4: `OpenFileDialog` Class Members

Figure 5.3 displays the OpenFileDialog class.

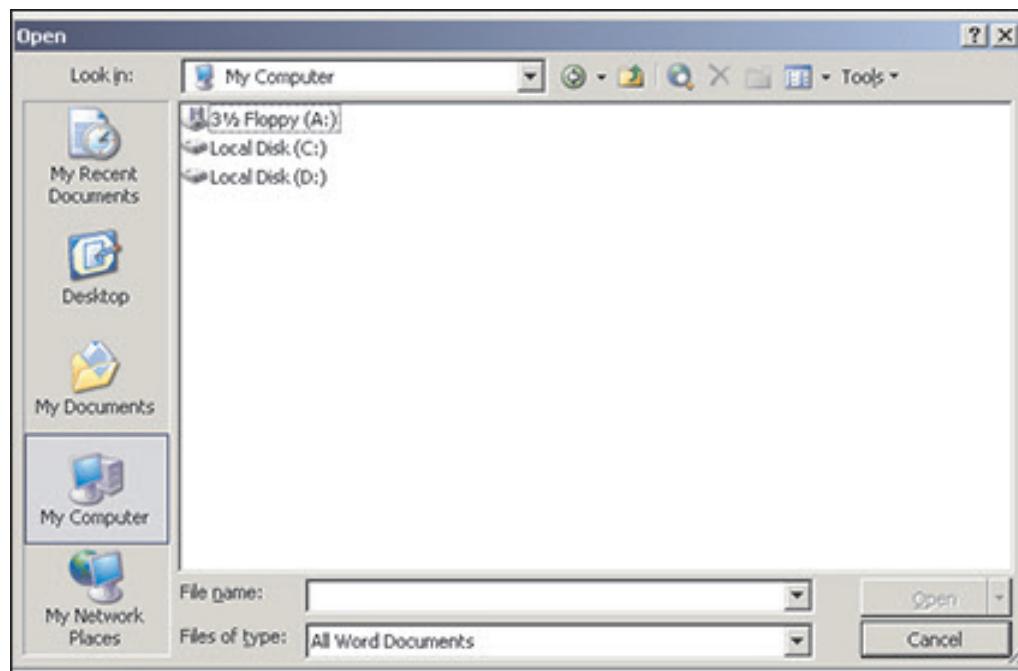


Figure 5.3: OpenFileDialog

The following code demonstrates the OpenFileDialog class and its members.

#### Code Snippet:

```
 OpenFileDialog odlgFile = new OpenFileDialog();
odlgFile.CheckFileExists = true;
odlgFile.FileName = "OpenFileDialogDemo.txt";
odlgFile.Filter = "Text files (*.txt) | *.txt | Rich Text Format files (*.rtf) | *.rtf";
odlgFile.Multiselect = false;
Stream stream = odlgFile.OpenFile();
odlgFile.Title = "Select The File";
private void odlgFile_FileOk(object sender, CancelEventArgs e)
{
    // Specified file can be loaded in the RichTextBox control
}
```

The code creates an instance of the OpenFileDialog class. As the CheckFileExists property value is set to true, the dialog box will display the warning when you specify a file name that does not exist. The FileName property indicates that OpenFileDialogDemo.txt is the file selected in the file dialog box. The Filter property is set to .txt and .rtf format. This allows you to open the text files and rich text format files only. As the MultiSelect property is set to false, you cannot select multiple files from the dialog box. The OpenFile() method opens the OpenFileDialogDemo.txt file with read-only permission. When the user clicks the Open button in the dialog box, the FileOk event is triggered.

**Note** - Once the `OpenFileDialog` component has been added to a form, it appears in the tray at the bottom of the Windows Forms Designer.

### → SaveFileDialog Class

The `SaveFileDialog` class allows you to create the `SaveFileDialog` component. Table 5.5 lists the most commonly used properties, methods, and events of the `SaveFileDialog` class.

Name	Description
CreatePrompt	This property specifies or retrieves a value that indicates whether the dialog box prompts you for permission while creating a file that does not exist.
OverwritePrompt	This property specifies or retrieves a value that indicates whether a warning is displayed when the user specifies a file name that already exists.
OpenFile	This method opens the selected file with read/write permission.
FileOk	This event occurs when Save button on a <b>file</b> dialog box is clicked.

Table 5.5: `SaveFileDialog` Class Members

The following code demonstrates the `SaveFileDialog` class and its members.

#### Code Snippet:

```
SaveFileDialog sdlgSave = new SaveFileDialog();
sdlgSave.CreatePrompt = true;
sdlgSave.OverwritePrompt = true;
sdlgSave.ShowDialog();
Stream stream = sdlgSave.OpenFile();

private void sdlgSave_FileOk(object sender, CancelEventArgs e)
{
    MessageBox.Show("File has been saved.");
}
```

The code creates an instance of the `SaveFileDialog` class. As the `CreatePrompt` property is set to `true`, the dialog box will prompt for permission to create a file that does not exist. The `OpenFile()` method opens the selected file with read/write permission. As the `OverwritePrompt` property is set to `true`, a warning is generated if you save a file with the existing file name. When the user clicks the **Save** button in the dialog box, the `FileOk` event is triggered. This event displays a message stating that the file has been saved.

Figure 5.4 displays the SaveFileDialog.

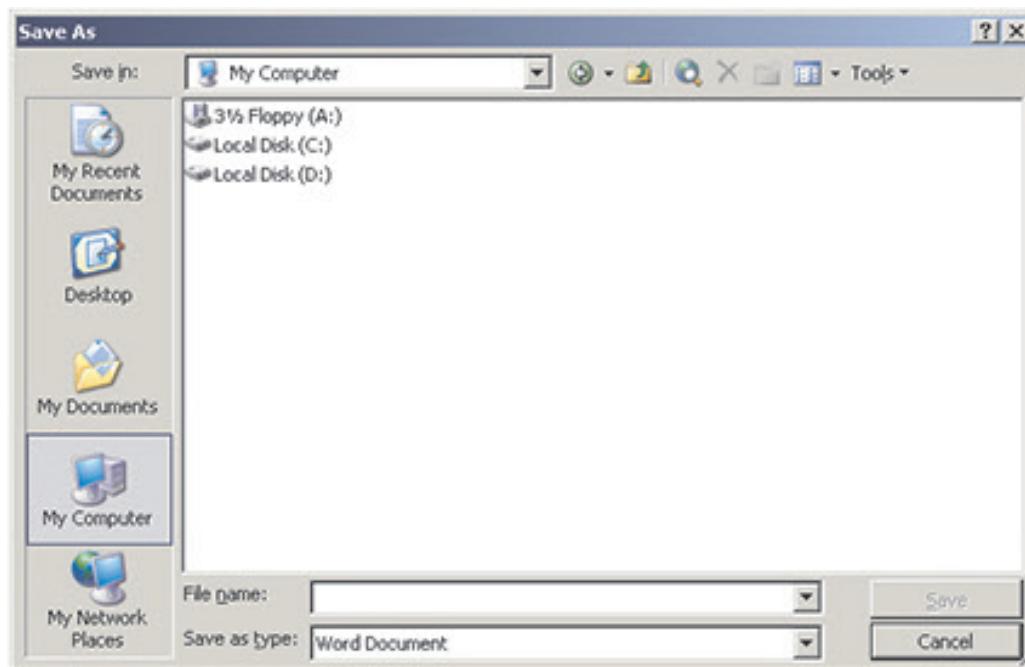


Figure 5.4: SaveFileDialog

**Note** - Once the SaveFileDialog component has been added to a form, it appears in the tray at the bottom of the Windows Forms Designer.

### 5.2.5 PageSetupDialog Component

The PageSetupDialog component is a system-defined dialog box, which allows the user to set the page layout for printing. It provides various options such as the border and margin adjustments, headers and footers, and portrait and landscape orientation for the page to be printed. This is similar to the **Page Setup** dialog box of Microsoft Word.

#### → PageSetupDialog Class

The PageSetupDialog class allows you to create the PageSetupDialog component. Table 5.6 lists the most commonly used properties of the PageSetupDialog class.

Name	Description
AllowOrientation	This property specifies or retrieves a value, which indicates whether the user can enable to set the orientation section using the orientation area of the dialog box. The two orientation options displayed are Landscape and Portrait.
AllowPrinter	This property specifies or retrieves a value that indicates the enabled or disabled status of the Printer button.

Name	Description
Document	This property specifies or retrieves a value that indicates the object of PrintDocument class from which the page settings can be retrieved.
PageSettings	This property specifies or retrieves a value, which indicates the settings of the page to be changed.
PrinterSettings	This property specifies or retrieves the printer settings, which are changed upon the click event generated, when the user clicks the Printer button in the dialog box.

Table 5.6: PageSetupDialog Class Members

The following code demonstrates the PageSetupDialog class and its members.

**Code Snippet:**

```
PageSetupDialog psdlgSetup = new PageSetupDialog();
psdlgSetup.AllowOrientation = true;
psdlgSetup.Document = new System.Drawing.Printing.PrintDocument();
psdlgSetup.ShowDialog();
```

The code creates an object of the PageSetupDialog class named, psdlgSetup. The AllowOrientation property is set to true that indicates the orientation section of the dialog box is enabled. The Document property is used to retrieve the document from which the settings are retrieved. This is done by invoking the constructor of the PrintDocument class.

Figure 5.5 displays the **PageSetupDialog**.

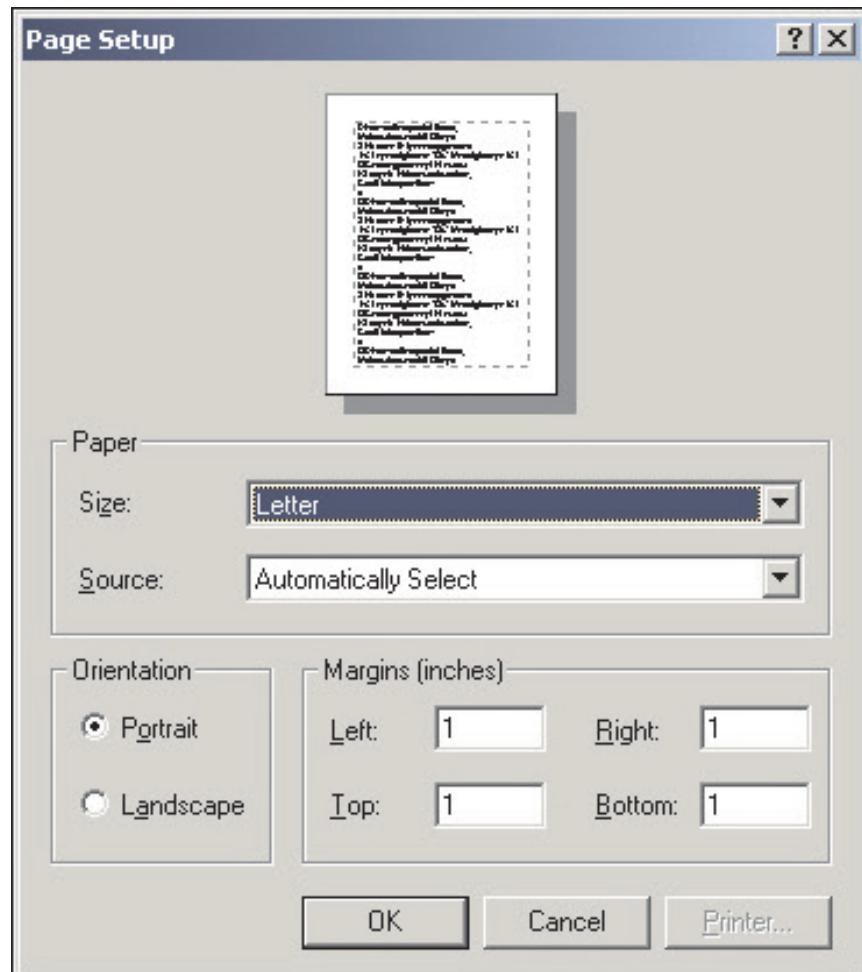


Figure 5.5: **PageSetupDialog** Class

**Note** - Once the **PageSetupDialog** component has been added to a form, it appears in the tray at the bottom of the Windows Forms Designer.

### 5.2.6 PrintDialog Component

The **PrintDialog** component is a system-defined dialog box that allows you to print the documents. It allows you to select a printer, choose the pages to print, and decide other print-related settings. You can print either all pages of the document, or a specified page range, or can even print the selected pages. This is similar to the **Print** dialog box of Microsoft Word.

#### → PrintDialog Component

The **PrintDialog** class allows you to create the **PrintDialog** component.

Table 5.7 lists the most commonly used properties of the `PrintDialog` class.

Name	Description
AllowPrintToFile	This property specifies or retrieves a value, which indicates whether the <b>Print to file</b> check box is enabled.
Document	This property specifies or retrieves a value, which represents the <code>PrintDocument</code> that is used to obtain the settings of the printer.
PrinterSettings	This property specifies or retrieves the settings of the printer, which is modified by the dialog box.
PrintToFile	This property specifies or retrieves a value, which indicates whether user has selected the <b>Print to file</b> check box.

Table 5.7: `PrintDialog` Class Members

Figure 5.6 displays the `PrintDialog`.

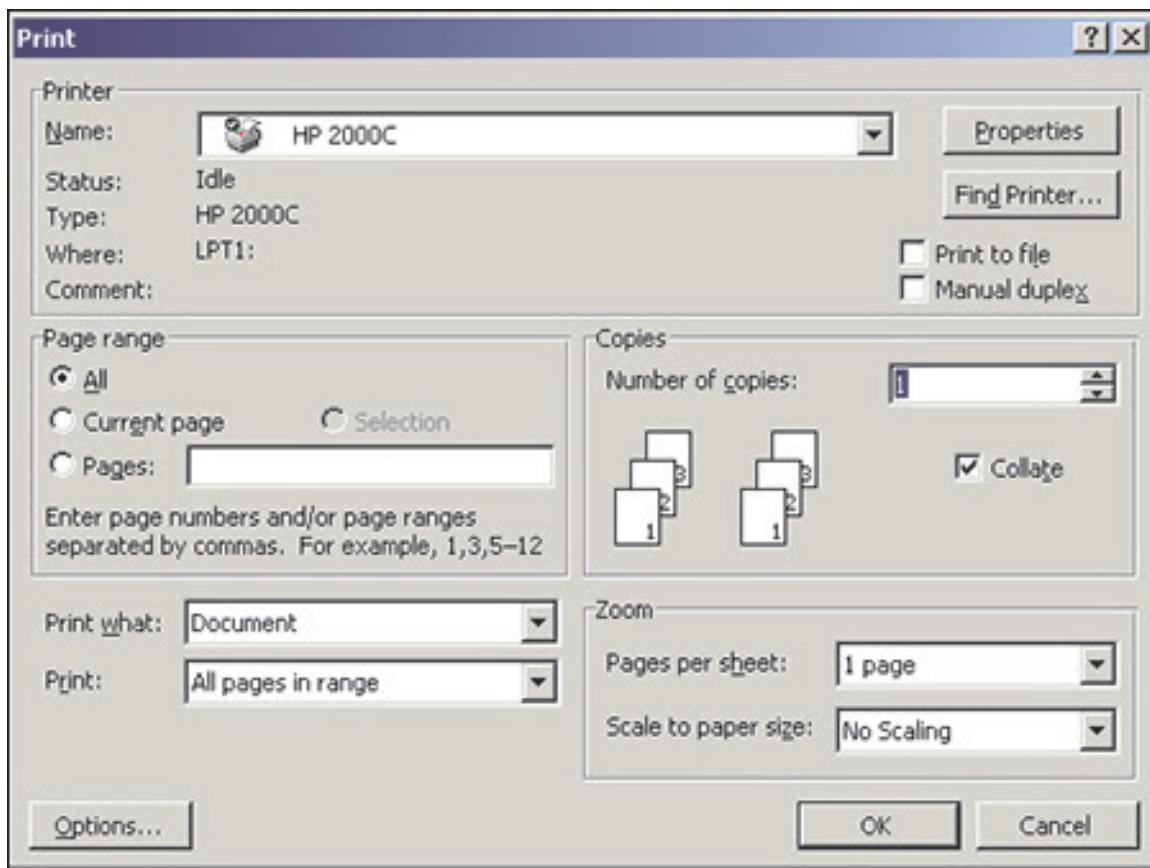


Figure 5.6: `PrintDialog` Class Members

The following code demonstrates the `PrintDialog` class and its members.

**Code Snippet:**

```
PrintDialog pdlgPrint = new PrintDialog();
pdlgPrint.PrintToFile = true;
pdlgPrint.ShowDialog();
```

The code creates an object of the `PrintDialog` class. The `PrintToFile` property is set to `true`, which indicates that the **Print to file** check box is selected.

**Note** - Once the `PrintDialog` component has been added to a form, it appears in the tray at the bottom of the Windows Forms Designer.

### 5.2.7 PrintPreviewDialog Control

The `PrintPreviewDialog` control is used to display the way a document will appear when printed. The control contains icons for printing, zooming in, displaying one or multiple pages, and closing the control. The control is used to display a document before it is to be printed.

→ **PrintPreviewDialog Class**

The `PrintPreviewDialog` class allows you to create the `PrintPreviewDialog` control. Table 5.8 lists the most commonly used properties of `PrintPreviewDialog` class.

Name	Description
CancelButton	This property specifies or retrieves the Cancel button for the <code>PrintPreviewDialog</code> component.
Document	This property specifies or retrieves a value, which represents the document to be previewed.
PrintPreviewControl	This property retrieves a value that indicates the presence of the <code>PrintPreviewControl</code> in the form.

Table 5.8: `PrintPreviewDialog` Class Members

The following code demonstrates the `PrintPreviewDialog` class and its members.

**Code Snippet:**

```
PrintPreviewDialog ppdlgPreview = new PrintPreviewDialog();
ppdlgPreview.Document = new System.Drawing.Printing.PrintDocument();
ppdlgPreview.ShowDialog();
```

The code creates an object of the `PrintPreviewDialog` class. The `Document` property is used to specify the document to be previewed. This is done by invoking the constructor of the `PrintDocument` class.

**Note** - Once the `PrintPreviewDialog` component has been added to a form, it appears in the tray at the bottom of the Windows Forms Designer.

## Knowledge Check 2

1. You want to create an `OpenFileDialog` dialog box within the application, which will allow you to open .txt and .rtf files. Which one of the following codes will help you to achieve this?

(A)	<pre>OpenFileDialog odlgFile = new OpenFileDialog(); odlgFile.Filter = "Text files (*.txt) *.txt Rich Text Format files (*.rtf) *.rtf"; odlgFile.ShowDialog();</pre>
(B)	<pre>OpenFileDialog odlgFile = new OpenFileDialog(); odlgFile.AutoFilter = "Text files (*.txt) *.txt Rich Text Format files (*.rtf) *.rtf"; odlgFile.ShowDialog();</pre>
(C)	<pre>OpenFileDialog odlgFile = new OpenFileDialog(); odlgFile.Filters = "Text files (*.txt) *.txt Rich Text Format files (*.rtf) *.rtf"; odlgFile.ShowDialog();</pre>
(D)	<pre>OpenFileDialog odlgFile = new OpenFileDialog(); odlgFile.FilterSelect = "Text files (*.txt) *.txt Rich Text Format files (*.rtf) *.rtf"; odlgFile.ShowDialog();</pre>

2. Which of the following statements regarding the common dialog boxes are false?

(A)	The <code>ShowDialog()</code> method is used to display the dialog box to the user.
(B)	The <code>CustomColors</code> property is used to set or get the custom colors of the <code>ColorDialog</code> component.
(C)	The <code>Forts</code> property is used to set or get the selected font style of the <code>FontDialog</code> component.
(D)	The <code>OpenFiles()</code> method is used to open the specified file with the read-only permission.
(E)	The <code>AllowOrientation</code> property is used to set or get a value whether the user can specify the orientation of a page.

### 5.3 User-Defined Dialog Boxes

In this third lesson, **User-Defined Dialog Boxes**, you will learn to:

- ➔ State the need for user-defined dialog boxes.
- ➔ Outline the steps to create a user-defined dialog box.

### 5.3.1 User-Defined Dialog Boxes

Consider a scenario where a developer has to create a quiz application for school students, which they can play during their lab sessions. The game is only meant for children belonging to the age group of 8-14 years. One of the requirements of the application is to take the user name and age from the user. In such case, the developer needs to create a custom dialog box which will ask the student to enter his/her name and age to start with the quiz. The dialog box will only contain a `TextBox` control, a `NumericUpDown` control for age field, and the **OK** and **Cancel** buttons.

**Note** - Dialog boxes can be customized as modal as well as modeless.

### 5.3.2 Creating a User-Defined Dialog Box

A system-defined dialog box does not always fulfill the requirements of the developers and users. In such cases, a developer needs to create custom dialog boxes to fulfill the requirements. To do so, you can convert a form into a user-defined dialog box. To create a user-defined dialog box:

- The `FormBorderStyle` property of the form should be set to `FixedDialog`. This property changes the borders of the form to the standard borders of a dialog box.
- The `ControlBox`, `MinimizeBox`, and `MaximizeBox` properties should be set to `False`. This causes the window to display only the system-defined Close button.
- The form must provide the **OK** and **Cancel** buttons to either close the dialog box or cancel any tasks within the dialog box.
- The `AcceptButton` property of the form should be used to set the **OK** button.
- The `CancelButton` property of the form should be used to set the **Cancel** button.

**Note** - Dialog boxes generally do not consist of menu bars, window scroll bars, Minimize and Maximize buttons, or status bars. They neither have resizable borders.

### Knowledge Check 3

1. Which of the following snippets about the user-defined dialog box are correct?

(A)	<code>Form frmDialog = new Form();</code>
(B)	<code>frmDialog.MinimizedBox = false;</code>
(C)	<code>frmDialog.Border = Border.FixedDialog;</code>
(D)	<code>frmDialog.MaximizedBox = false;</code>
(E)	<code>frmDialog.ShowDialog();</code>
(F)	<code>frmDialog.ShowDialogBox();</code>

(G)	<code>frmDialog.AcceptButton = btnOK;</code>
(H)	<code>frmDialog.CancelButton = btnCancel;</code>

2. Which of the following statements about user-defined dialog boxes are true?

(A)	A user-defined dialog box can be defined as modal and modeless.
(B)	A user-defined dialog box can be made using the <code>FormBorderStyle</code> property of a form.
(C)	A user-defined dialog box can be created by setting the <code>MaximizeBox</code> and <code>MinimizeBox</code> properties to <code>True</code> .
(D)	A user-defined dialog box can have the Cancel button using the <code>CancelButtons</code> property.
(E)	A user-defined dialog box can have an <b>OK</b> button using the <code>AcceptButtons</code> property.

## 5.4 Retrieving Information from a Dialog Box

In this last lesson, **Retrieving Information from a Dialog Box**, you will learn to:

- Explain how to retrieve the results of a dialog box.
- Describe the  `DialogResult` value.

### 5.4.1 Retrieving Information from a Dialog Box

The form that invokes the dialog box can retrieve the information from it once the dialog box is closed. The form that displays the dialog box can retrieve the result of that dialog box by referencing its  `DialogResult` property. The result can also be retrieved by referencing the return value of a call to the `ShowDialog()` method. The form that displayed the dialog box then, responds according to the value returned.

For example, the message box used in Windows applications is a modal dialog box. It has **OK** and **Cancel** buttons. The form that displays the message box can track the button the user has clicked and can perform the required tasks.

### 5.4.2 DialogResult Enumeration

The  `DialogResult` property is used to retrieve information from the dialog box. You can specify the value of the  `DialogResult` property using the  `DialogResult` enumeration. The enumeration defines various identifiers to retrieve values from the dialog box.

Table 5.9 lists the different identifiers defined in the `DialogResult` enumeration.

Name	Description
Abort	The return value of the dialog box is Abort.
Cancel	The return value of the dialog box is Cancel.
Ignore	The return value of the dialog box is Ignore.
No	The return value of the dialog box is No.
None	The return value of the dialog box is None. This means that the modal dialog box is running.
OK	The return value of the dialog box is OK.
Retry	The return value of the dialog box is Retry.
Yes	The return value of the dialog box is Yes.

Table 5.9: Identifiers

The following code uses the `DialogResult` enumeration to retrieve values from the user-defined dialog box.

**Code Snippet:**

```
Form frmCustomDialog = new Form();
Button btnOK = new Button();
Button btnCancel = new Button();
frmCustomDialog.FormBorderStyle = FormBorderStyle.FixedDialog;
frmCustomDialog.MaximizeBox = false;
frmCustomDialog.MinimizeBox = false;
frmCustomDialog.Text = "Custom Dialog";
btnOK.Text = "OK";
btnCancel.Text = "Cancel";
btnOK.Location = new Point(10, 100);
btnCancel.Location = new Point(100, 100);
frmCustomDialog.Controls.Add(btnCancel);
frmCustomDialog.Controls.Add(btnOK);
frmCustomDialog.AcceptButton = btnOK;
frmCustomDialog.CancelButton = btnCancel;

if (frmCustomDialog.ShowDialog() == DialogResult.Cancel)
{
    // write code to perform some action.
}
```

The code creates a custom dialog box by customizing the form. Two buttons named OK and Cancel are added to the dialog box. The `FormBorderStyle` of the dialog box is set to `FixedDialog`. The `MaximizeBox` and `MinimizeBox` properties of the dialog box are set to `false`. The `Text` property of the dialog box is set to 'Custom Dialog'. The OK button is set a new location (10, 100) and the Cancel button is set to location (100, 100) on the form. The `AcceptButton` property is used to set the `btnOK`

button as the **OK** button.

Similarly, the `CancelButton` property is used to set the `btnCancel` button as the Cancel button.

The `Cancel` identifier of the `DialogResult` enumeration is used to track the `Click` event for the Cancel button. When the user clicks the **Cancel** button, the `if` condition becomes true and the user can perform some actions.

## Knowledge Check 4

1. You want to check whether the user has clicked the Yes button of the message box. Which one of the following codes will help you achieve this?

(A)	<pre>if (MessageBox.ShowDialog("Do you want to close the application ?",     "Message Box", MessageBoxButtons.YesNo) == DialogResult.Yes)  {     Application.Exit(); }  else  {     this.Hide(); }</pre>
(B)	<pre>if (MessageBox.Show("Do you want to close the application ?", "Message Box", MessageBoxButtons.YesNo) = "Yes")  {     Application.Exit(); }  else  {     this.Hide(); }</pre>

(C)	<pre>if (MessageBox.Show("Do you want to close the application?", "Message Box", MessageBoxButtons.YesNo) == Yes) {     Application.Exit(); } else {     this.Hide(); }</pre>
(D)	<pre>if (MessageBox.Show("Do you want to close the application?", "Message Box", MessageBoxButtons.YesNo) == DialogResult.Yes) {     Application.Exit(); } else {     this.Hide(); }</pre>

## Module Summary

In this module, **Dialog Boxes**, you learnt about:

### → **Types of Dialog Boxes**

Dialog boxes are used to interact with the user and retrieve information. The two types of dialog boxes are modal and modeless. A modal dialog box does not allow you to continue working with the rest of the application unless the dialog box has been closed or hidden.

A modeless dialog box allows you to continue working with the application while the dialog box is displayed. Modeless dialog boxes enable shifting focus between the dialog box and another application without having to close the first application.

### → **System-defined Dialog Boxes**

Common dialog boxes are system-defined dialog boxes that can be reused across various applications. All the system-defined dialog boxes are inherited from the base class `CommonDialog` class. Some of the common dialog boxes are `ColorDialog`, `FontDialog`, `OpenFileDialog`, `SaveFileDialog`, `PageSetupDialog`, `PrintDialog`, and `PrintPreviewDialog`.

### → **User-defined Dialog Boxes**

You can create your own dialog boxes by using certain properties of the form. These properties are `FormBorderStyle`, `ControlBox`, `MinimizeBox`, `MaximizeBox`, `AcceptButton`, and `CancelButton`. Dialog boxes can be customized into modal as well as modeless styles.

### → **Retrieving Information from a Dialog Box**

Information taken through the dialog box can be retrieved and used for further processing. The `DialogResult` property and the `ShowDialog()` method are used for this purpose.

GROWTH  
RESEARCH  
OBSERVATION  
UPDATES  
PARTICIPATION



# Module - 6

## MDI Applications and Menus

Welcome to the Module, **MDI Applications and Menus**.

This module covers about the different document layouts used in applications, their various types, and their uses. MDI applications provide simultaneous access to multiple documents in a single window. They normally consist of menus to navigate through different child forms. The ToolStrip and StatusStrip controls are used to provide functionalities similar to the status bar and toolbar.

In this Module, you will learn about:

- Multiple Document Interface Applications
- Menus
- ToolStrip Control
- StatusStrip Control

## 6.1 Multiple Document Interface Applications

In this first lesson, **Multiple Document Interface Applications**, you will learn to:

- Define Multiple Document Interface.
- Explain MDI parent forms.
- Explain MDI child forms.
- List and explain the properties, methods, and events for implementing MDI forms.
- Explain how to activate and deactivate child forms.

### 6.1.1 Document Interfaces

An application can contain multiple forms, which are used for collecting and displaying varied information. The manner in which you display and use these forms is implemented by designing a document interface. A document interface is the layout of the application window, which is used for opening and organizing windows forms in a desired and suitable manner. There are basically two types of document interfaces namely, Single Document Interface (SDI) and Multiple Document Interface (MDI).

Consider a scenario where you are working with a Windows Notepad application. In this application, you can open and work only with a single file at a time. To work with multiple files, you need to open multiple instances of Notepad application. This application is an example of Single Document Interface.

**Note** - There are two more types of document interfaces namely, the Explorer Style interface and Tabbed interface. The explorer style interface is similar to the Windows Explorer, where you use the ListView and TreeView controls to display hierarchical information. The tabbed interface is created using the Tab control to display different tabs.

#### → Multiple Document Interfaces

Consider a scenario where you are working with Microsoft Word application. In this application, you can open multiple documents simultaneously within the single instance of the application. The application displays multiple child windows in the working area of the application. You can work on any of the child windows by just switching between them. This is an example of Multiple Document Interface. An MDI application allows you to open multiple documents simultaneously.

There are various advantages of using MDI applications. Some of these are as follows:

- **Systematic Organization**

An MDI application helps in arranging all the related forms in an organized manner.

- **Increased Efficiency**

An MDI application allows easy access to multiple forms through a single toolbar. This increases the speed and efficiency of accessing multiple forms.

- **Less Memory Usage**

An MDI application has the capability of opening multiple forms within its single window. This helps in reducing the usage of memory as against SDI, which needs more memory to open multiple instances to work with multiple files.

- **Easy Implementation**

An MDI application facilitates easy and simultaneous editing in multiple forms.

There are some disadvantages of using MDI applications. They are as follows:

- **Minimize Data Inconsistency**

An MDI application must at least have one child form. In case of multiple child forms, it becomes difficult for the user to work with multiple forms, and to switch between forms to make the desired changes. This increases the risk of inconsistent data and the user might be confused and lose track of which forms were accessed to make changes.

### 6.1.2 MDI Parent Forms

An MDI application consists of a parent form and multiple child forms. The parent form acts as the base window through which the user interacts with the MDI application. All the child forms are opened within this form. Thus, it acts as a container that can hold all child forms. An MDI application can consist of only one parent form. Figure 6.1 displays the MDI parent.



Figure 6.1: MDI Parent

For example, when you open the Microsoft Word application, you can open more than one document within the parent window.

The `IsMdiContainer` property of the form can be used to create an MDI form. The following code demonstrates the MDI form.

#### Code Snippet:

```
this.IsMdiContainer = true;
```

The code uses `this` keyword that refers to the current form. The `IsMdiContainer` property of the form is set to `true`. This means that the current form will behave as the parent form.

### 6.1.3 MDI Child Forms

The MDI child forms are the sub-forms that are opened within the parent form. The user actually works with these child forms. A parent form is just a medium through which the user accesses the child forms. When the parent form is closed, all the child forms associated with it are also closed. However, if all the child forms are closed, the parent form is not closed automatically. Figure 6.2 displays the MDI child form.

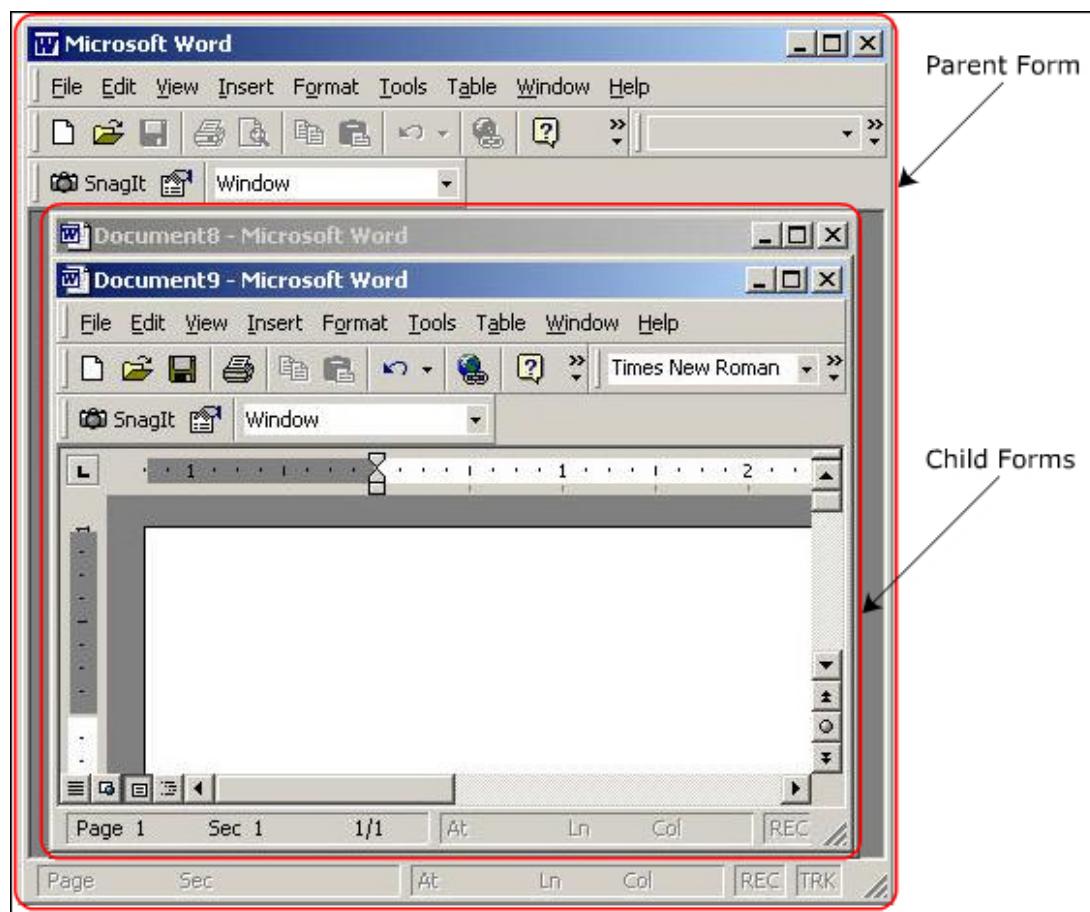


Figure 6.2: MDI Child Form

The following code creates an MDI child form.

**Code Snippet:**

```
this.IsMdiContainer = true;
Form frmChildOne = new Form();
frmChildOne.MdiParent = this;
frmChildOne.Show();
```

The `IsMdiContainer` property sets the current form as the MDI container. The code creates an object of the `Form` class namely, `frmChildOne`. The `MdiParent` property of the form is set to `this`, which represents the current form, which is set as the parent form.

#### 6.1.4 Properties, Methods, and Events

The `Form` class is used to implement MDI parent and child forms. You can create and display MDI parent and child forms using the various properties, methods, and events of the `Form` class.

→ **Properties**

The properties of the `Form` class can be used to specify or retrieve the parent or child form. Table 6.1 lists the commonly used properties of the `Form` class.

Property	Description
<code>ActiveMdiChild</code>	Retrieves the active MDI child window.
<code>IsMdiChild</code>	Retrieves a value, which indicates whether the form is a multiple-document interface child form.
<code>IsMdiContainer</code>	Specifies or retrieves a value, which indicates whether the form is a container to hold multiple-document interface child forms.
<code>MdiChildren</code>	Retrieves an array of forms that are child forms of the current parent form.
<code>MdiParent</code>	Specifies or retrieves the present parent form of the current form.

Table 6.1: Form Properties

→ **Methods**

Table 6.2 lists the most commonly used methods of the `Form` class.

Method	Description
<code>ActivateMdiChild</code>	Activates the specified MDI child form.
<code>LayoutMdi</code>	Provides a layout to organize the child forms within the parent form. Different layouts are defined in the <code>MdiLayout</code> enumeration, which are <code>ArrangeIcons</code> , <code>Cascade</code> , <code>TileHorizontal</code> , and <code>TileVertical</code> .
<code>Show</code>	Displays the form.

Table 6.2: Form Methods

→ Events

Table 6.3 lists the most commonly used events of the `Form` class.

Event	Description
MdiChildActivate	Occurs when the child form is activated or closed.

Table 6.3: Form Events

The following code demonstrates creating and managing MDI parent forms.

**Code Snippet:**

```
Form frmMDI = new Form();
frmMDI.IsMdiContainer = true;
frmMDI.LayoutMdi(MdiLayout.Cascade);
frmMDI.Show();

private void frmMDI_Activated(object sender, EventArgs e)
{
    frmMDI.LayoutMdi(MdiLayout.TileHorizontal);
}
```

The code creates an object of the `Form` class, namely `frmMDI`. The `IsMdiContainer` property is set to true, which means that the form is the parent form. The `LayoutMdi()` method is used to set the layout of the form as Cascade using the `MdiLayout` enumeration, which arranges the forms one over the other. When the form is activated by the user, the `Activated` event occurs, which changes the layout of the form to `TileHorizontal` so that the forms are tiled horizontally.

### 6.1.5 Activating and Deactivating Forms

You can activate the child forms by using the `ActivateMdiChild()` method of the `Form` class or by invoking the `Activate()` method. When the user activates the child form, the `Activated` event occurs for the child form. When the focus is taken away from the child form, the `Deactivate` event of the child form is fired.

The following code demonstrates creating MDI forms.

**Code Snippet:**

```
frmChildOne fOne = new frmChildOne();
fOne.MdiParent = this;
fOne.Show();
frmChildTwo fTwo = new frmChildTwo();
fTwo.MdiParent = this;
fTwo.Show();
this.MdiChildren[0].Activate();
```

The code creates two different objects of forms named, `frmChildOne` and `frmChildTwo`. The current form is assigned as the parent of both these child forms.

The `Show()` method is used to show the form and `Activate()` method is used to activate the first child form of the parent form.

## Knowledge Check 1

1. Which of these statements about MDI are true?

(A)	An MDI application supports maximum two parent forms.
(B)	A child form remains open even when the parent form is closed.
(C)	A parent form remains open even if all the child forms are closed.
(D)	An MDI application increases the risk of data inconsistency while working with multiple child forms.
(E)	The <code>IsMdiFormContainer</code> property is used to create a parent form.

2. Can you match the properties, methods, and events of the `Form` class against their corresponding descriptions?

Description		Property, Method, and Event
(A)	Method used to organize the child forms within the parent form.	(1) <code>IsMdiChild</code>
(B)	Property used to retrieve the collection of child forms of the current parent form.	(2) <code>ActiveMdiChild</code>
(C)	Event that occurs when the child form is activated or closed.	(3) <code>LayoutMdi</code>
(D)	Property used to retrieve the active child form.	(4) <code>MdiChildActivate</code>
(E)	Property used to determine whether the form is a child form.	(5) <code>MdiChildren</code>

## 6.2 Menus

In this second lesson, **Menus**, you will learn to:

- List and describe briefly the menu system in Windows Forms.
- Describe the `MenuStrip` control, its properties, methods, and events.
- Describe the `ContextMenuStrip` control and its properties.
- Explain the menu classes such as `MainMenu`, `ContextMenu` and `MenuItem`.

### 6.2.1 Menu System

Consider a scenario where a developer is creating an Employee Details application. This application must allow the user to access multiple forms. It must allow the user to access the personal details, salary details, leave details, and so on. It must also allow the user to exit the application. This can be achieved by creating menus, which allow the user to switch between applications. This is similar to the menus appearing in Microsoft Word application. Figure 6.3 displays the menu system.

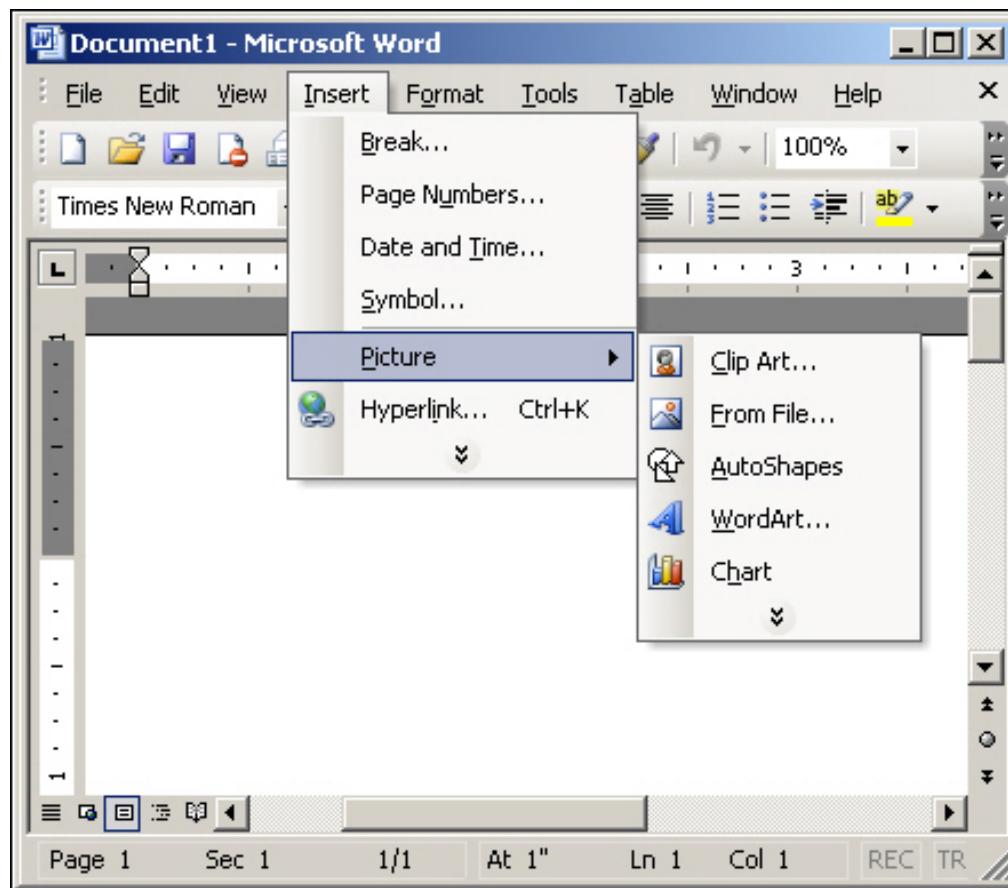


Figure 6.3: Menu System

The menu system in Windows Forms consists of the main menu panel. The main menu panel consists of menu bars and various sub-menus.

#### → Menu Structures

There are two types of menus in Windows Forms namely, main menu and context menu. The main menu, also, known as anchored menus, appears on the menu bar of the form. The main menu consists of a set of items that are displayed horizontally across the top of most applications. Context menu, also known as popup menu, are menus that appear when you click the right mouse button. The context menus are the short-cut menus that contain the frequently used commands.

## 6.2.2 MenuStrip Control

The `MenuStrip` control is a new control introduced in Visual Studio 2005 and .NET Framework 2.0. The `MenuStrip` control helps you to add new menus, modify and reorder existing menus, and delete old menus. You can also use the access keys, check marks, images, and separator bars to improve the usability of menus.

The `MenuStrip` control supports MDI, merging menus, and tool tips.

### → Properties, Methods, and Events

The `MenuStrip` class is used to create the `MenuStrip` control and it exists in the `System.Windows.Forms` namespace. It is inherited from the `ToolStrip` class.

Table 6.4 lists the most commonly used properties, methods, and events of `MenuStrip` class.

Name	Description
<code>AllowMerge</code>	This property specifies or retrieves a value, which indicates whether multiple <code>MenuStrip</code> , <code>ToolStripDropDownMenu</code> , <code>ToolStripMenuItem</code> , and other types can be merged.
<code>LayoutStyle</code>	This property specifies or retrieves a value, which indicates how items should be placed in the <code>ToolStrip</code> .
<code>ShowItemToolTips</code>	This property specifies or retrieves a value whether tool tips can be displayed for the control.
<code>Stretch</code>	This property specifies or retrieves a value, which indicates whether the <code>MenuStrip</code> can stretch itself in the container.
<code>GetItemAt</code>	This method retrieves the item at the given index location.
<code>MenuActivate</code>	This event occurs when the user activates the menu using the keyboard or mouse.
<code>MenuDeactivate</code>	This event occurs when the menu is deactivated.

Table 6.4: `MenuStrip` Class Members

The following code creates the `MenuStrip` control.

#### Code Snippet:

```
MenuStrip mnstrupStripDemo = new MenuStrip();
mnstrupStripDemo.ShowItemToolTips = true;
this.MainMenuStrip = mnstrupStripDemo;
```

The code creates an object of `MenuStrip` class namely, `mnstrupStripDemo`. The `ShowItemToolTips` property is set to true, which specifies that tool tips will be shown for the `MenuStrip`.

### → ToolStripMenuItem Control

The `ToolStripMenuItem` class provides various properties, which enable you to modify the appearance and functionality of a menu item. This class is a member of `System.Windows.Forms` namespace. Table 6.5 lists the most commonly used properties and methods of `ToolStripMenuItem` class.

Name	Description
Checked	This property specifies or retrieves a value, which indicates whether the <code>ToolStripMenuItem</code> is checked.
CheckState	This property specifies or retrieves a value, which indicates whether a <code>ToolStripMenuItem</code> is in the checked, unchecked, or indeterminate state.
Select	This method is used to select the item.

Table 6.5: `ToolStripMenuItem` Properties and Methods

The following code creates the `ToolStripMenuItem` control and uses its properties and methods.

#### Code Snippet:

```
MenuStrip mnustrpMainMenu = new MenuStrip();
this.MainMenuStrip = mnustrpMainMenu;
ToolStripMenuItem tlstrpmnuFile = new ToolStripMenuItem("File");
ToolStripMenuItem tlstrpmnuExit = new ToolStripMenuItem("Exit");
mnustrpMainMenu.Items.Add(tlstrpmnuFile);
mnustrpMainMenu.Items.Add(tlstrpmnuExit);
mnustrpMainMenu.Dock = DockStyle.Top;
this.Controls.Add(mnustrpMainMenu);
```

The code creates an object of `MenuStrip` class namely, `mnustrpMainMenu`. This `MenuStrip` is assigned to current form using `MainMenuStrip` property. Two instances of `ToolStripMenuItem` are created, which are `tlstrpmnuFile` and `tlstrpmnuExit`, respectively. These two objects are added to `MenuStrip` object using `Add()` method. The `Dock` style of this `MenuStrip` is set to `Top` value of `DockStyle` enumeration and the `MenuStrip` is added to current form.

### 6.2.3 ContextMenuStrip Control

The `ContextMenuStrip` control is used to provide a short cut for accessing menu options displayed on the menu bar. It is similar to the context menu that appears in Microsoft Word application, when you right-click the document. It allows you to logically group the related options from the menus, thereby, allowing you to quickly perform the tasks.

The control is useful when you want to perform certain tasks frequently.

### → Properties, Methods, and Events

The `ContextMenuStrip` control is a new control introduced in .NET Framework 2.0. The class exists in the `System.Windows.Forms` namespace.

Table 6.6 lists the most commonly used properties of the ContextMenuStrip control.

Name	Description
SourceControl	This property retrieves the last control that caused this ContextMenu Strip to be displayed.
TextDirection	This property sets the direction in which the text will appear in the control.

Table 6.6: ContextMenuStrip Properties

Figure 6.4 displays the ContextMenuStrip control.

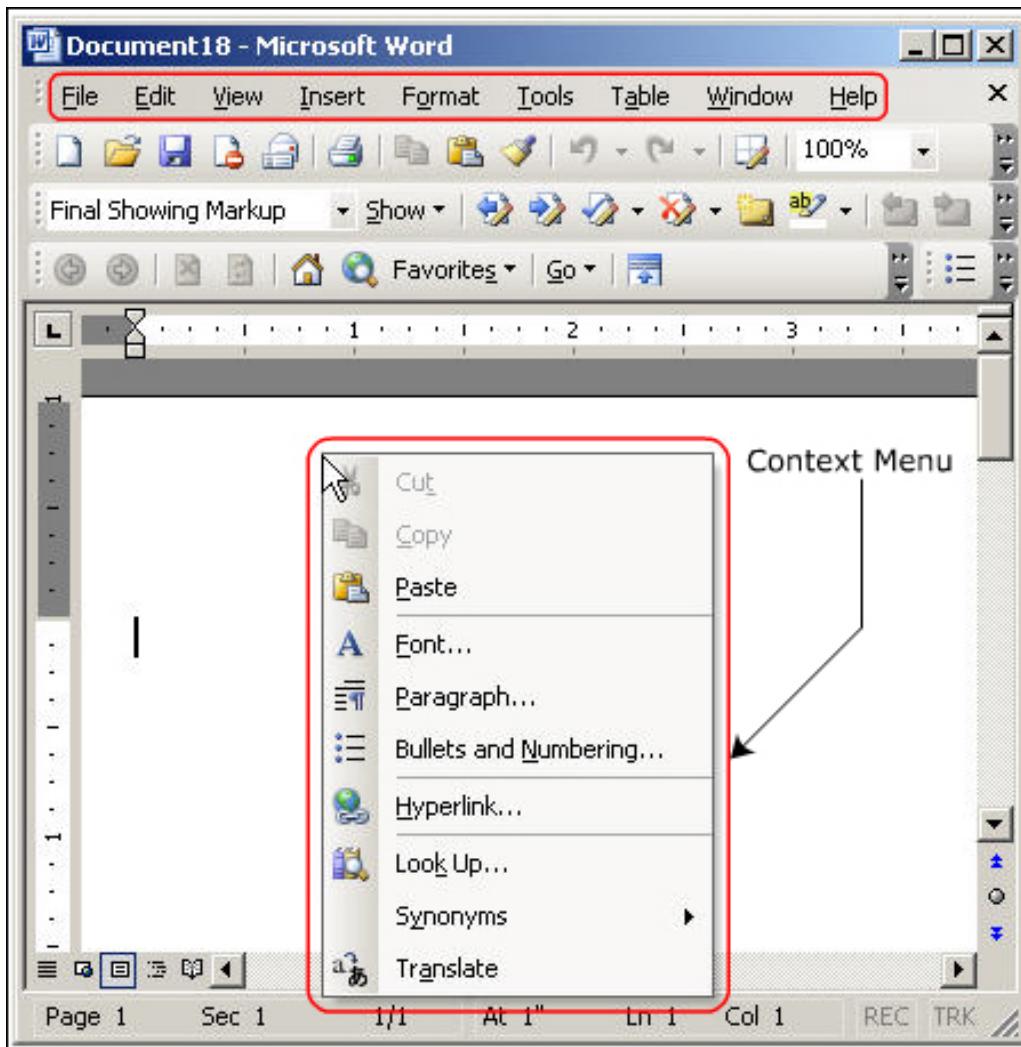


Figure 6.4: ContextMenuStrip Control

The following code creates the ContextMenuStrip control and assign it to form's ContextMenuStrip property.

**Code Snippet:**

```
ContextMenuStrip cmnustrpContext = new ContextMenuStrip();
cmnustrpContext.Items.Add("Copy");
cmnustrpContext.Items.Add("Paste");
cmnustrpContext.Items.Add("Exit");
this.ContextMenuStrip = cmnustrpContext;
```

The code creates an object of the ContextMenuStrip class namely, cmnustrpContext. The Add() method adds menu items to the ContextMenuStrip control. The created ContextMenuStrip is assigned to the current form using ContextMenuStrip property.

#### 6.2.4 Menu Class

The Menu class allows you to create menus. This class is the base class for MainMenu, MenuItem, and ContextMenu classes. The Menu class exists in the System.Windows.Forms namespace. Table 6.7 lists the most commonly used properties and methods of the Menu class.

Name	Description
IsParent	This read-only property retrieves a value that indicates whether the specified menu contains any menu items.
MdiListMenuItem	This property retrieves a value that indicates the MenuItem that is used to display a list of MDI child forms.
MenuItems	This property retrieves a value that indicates the items in the specified menu.
GetContextMenu	This method retrieves the ContextMenu that is associated with the specified menu.
GetMainMenu	This method retrieves the MainMenu that is associated with the specified menu.
MergeMenu	This method combines the objects of the MenuItem class of one menu with the current menu.

Table 6.7: Menu Properties

The following code creates a menu and attaches it to the current form.

**Code Snippet:**

```
Menu mnuMenu = new MainMenu();
this.Menu = (MainMenu)mnuMenu;
```

The code creates an object of MainMenu class. The object, mnuMenu is typecasted to MainMenu class and assigned as the main menu of the default form.

→ **MainMenu Class**

The MainMenu class contains a collection of objects of the MenuItem class.

Thus, the class represents the menu structure of a form. The `MainMenuItem` class exists in the `System.Windows.Forms` namespace. Table 6.8 lists the most commonly used properties, methods, and events of the `MainMenuItem` class.

Name	Description
MenuItems	This property retrieves a value that indicates the collection of <code>MenuItem</code> objects present in the menu.
RightToLeft	This property specifies or retrieves whether the text on the control is displayed from right to left.
GetForm	This method retrieves an instance of the <code>Form</code> class, which contains the specified menu.
GetMainMenu	This method retrieves an instance of the <code>MainMenuItem</code> class, which contains the specified menu.
Collapse	This event occurs when the main menu is hidden.

Table 6.8: `MainMenuItem` Members

The following code demonstrates how to create a `MainMenuItem` component.

#### Code Snippet:

```
Menu mnuMenu = new MainMenuItem();
mnuMenu.MenuItems.Add("File");
mnuMenu.MenuItems.Add("Edit");
this.Menu = (MainMenuItem)mnuMenu;
```

The code creates an instance of the `MainMenuItem` class. The menu items `File` and `Edit` are added to the component by using the `MenuItems` property. The object created is assigned as the main menu of the default form.

#### → ContextMenuItem Class

The `ContextMenuItem` class allows you to create a shortcut menu, which appears when you right-click the mouse. The `ContextMenuItem` class exists in the `System.Windows.Forms` namespace.

Table 6.9 lists the most commonly used properties, methods, and events of the `ContextMenuItem` class.

Name	Description
SourceControl	This property retrieves the control which is displaying the shortcut menu.
FindMenuItem	This method retrieves an instance of the <code>MenuItem</code> class, which contains the specified value.
Show	This method displays the shortcut menu at the specified position.
Popup	This event occurs before the shortcut menu is displayed.

Table 6.9: `ContextMenuItem` Members

The following code demonstrates how to create a `ContextMenu` component.

#### Code Snippet:

```
ContextMenu cmnuContext = new ContextMenu();
cmnuContext.MenuItems.Add("Copy");
cmnuContext.MenuItems.Add("Paste");

private void cmnuContext_Popup(object sender, EventArgs e)
{
    MessageBox.Show(cmnuContext.GetContextMenu().ToString());
}
```

The code creates an instance of the `ContextMenu` class. The menu items `Copy` and `Paste` are added to the component by using the `MenuItems` property. Before the context menu is displayed, the `Popup` event is fired. This event displays the context menu.

#### → MenuItem Class

The `MenuItem` class represents an individual item present within the main menu or context menu. Table 6.10 lists the most commonly used properties, methods, and events of the `MenuItem` control.

Name	Description
MdiList	This property specifies or retrieves a value, which indicates whether the menu item will contain a list of the child windows.
MdiListItem	This property retrieves a value that indicates an instance of the <code>MenuItem</code> class, which contains a list of MDI child forms.
Shortcut	This property specifies or retrieves a value indicating the shortcut key attached with the item.
ShowShortcut	This property specifies or retrieves a value, which indicates that the shortcut key related to the menu item is displayed next to the menu item caption.
MergeMenu	This method merges the specified menu item with another menu item.
PerformClick	This method generates a <code>Click</code> event for the <code>MenuItem</code> , simulating a user click.
PerformSelect	This method generates the <code>Select</code> event for this menu item.
Click	This event occurs when the menu item is clicked or selected using a shortcut key or access key, which is defined for the menu item.
Select	This event occurs when the cursor is placed over the menu item.

Table 6.10: MenuItem Members

The following code demonstrates how to create a `MenuItem` and use its properties and methods.

**Code Snippet:**

```
MainMenu mnuMain = new MainMenu();
MenuItem mnuiitemFile = new MenuItem("File");
MenuItem mnuiitemExit = new MenuItem("Exit");
mnu.MenuItems.Add(mnuiitemFile);
mnu.MenuItems.Add(mnuiitemExit);
this.Menu = mnuMain;
mnuiitemFile.PerformClick();
private void mnuiitemFile_Click(object sender, System.EventArgs e)
{
    MessageBox.Show("You clicked the File menu.", "The Event
Information");
}
```

The code creates an instance of the `MainMenu` class namely, `mnuMain`. It also creates two instances of `MenuItem` class, which is `mnuiitemFile` and `mnuiitemExit` respectively. These menu items are added to main menu using the `Add()` method of the `MenuItems` property. Then, it assigns the `mnuMain` object as the main menu of the current form. When the `PerformClick()` method is called, the `Click` event occurs and the message is displayed.

## Knowledge Check 2

- Which of these statements about menus are false?

(A)	A context menu appears when the left mouse button is clicked.
(B)	The <code>Shortcut</code> property of the <code>MenuItem</code> class specifies the right-click as the short cut key to access the menu.
(C)	The <code>ContextMenu</code> class exists in the <code>System.Windows.Forms</code> namespace.
(D)	The <code>IsParent</code> property of the <code>MainMenu</code> class determines whether the menu consists of any child forms.
(E)	The <code>MenuStrip</code> control helps you to add menus, modify, and reorder menus.

2. Can you match the properties, methods, and events of the `MainMenu`, `ContextMenu` and `MenuItem` classes against their corresponding descriptions?

Description		Properties, Methods, and Events	
(A)	This method retrieves the <code>MenuItem</code> that contains the value specified.	(1)	<code>Popup</code>
(B)	This event occurs before the shortcut menu is displayed.	(2)	<code>GetForm</code>
(C)	This property indicates whether the text is displayed from right to left.	(3)	<code>PerformClick</code>
(D)	This method generates a <code>Click</code> event for the <code>MenuItem</code> .	(4)	<code>RightToLeft</code>
(E)	This method retrieves the form that contains this control.	(5)	<code>FindMenuItem</code>

## 6.3 ToolStrip Control

In this third lesson, **ToolStrip Control**, you will learn to:

- Describe the use of `ToolStrip` control.
- Explain the properties, methods, and events of `ToolStrip` control.

### 6.3.1 ToolStrip Control

Consider a scenario where you are formatting the content of a document. The application provides all the formatting options in menus and in the toolbar. However, you will find it faster and more convenient to use the icons on the toolbar instead of the menus. In Visual Studio 2005, the `ToolStrip` control serves the purpose of the toolbar. It acts as a container to host menu items and other user-defined controls. The various features of `ToolStrip` control are as follows:

- Providing a common user interface across forms.
- Allowing dragging of items from one `ToolStrip` control to another.
- Supporting run-time ordering of items.
- Resembling the appearance and behavior of the operating system.

**Note** - `ToolStrip` is the base class for `MenuStrip`, `StatusStrip`, and `ContextMenuStrip` classes.

## → Properties, Methods, and Events

The ToolStrip control is a new control in .NET Framework 2.0. The ToolStrip class is used to create the ToolStrip control. When this control is placed on the form you can add buttons, labels, text boxes, separators, combo boxes, and drop-down buttons using the ToolStrip control. Table 6.11 lists the most commonly used properties, methods, and events of the ToolStrip class.

Name	Description
ImageList	This property specifies or retrieves the list of images, which contains the image displayed on a ToolStrip item.
Items	This property retrieves all the items of the control.
LayoutStyle	This property specifies or retrieves a value, which indicates how the items will be displayed on the control. The different layouts are defined in the ToolStripLayoutStyle enumeration.
GetNextItem	This method retrieves the next ToolStripItem from the specified reference point.
ItemAdded	This event occurs when a new ToolStripItem is inserted to the ToolStripItemCollection.
ItemClicked	This event occurs when an item on the control is clicked.
ItemRemoved	This event occurs when a ToolStripItem is removed from the ToolStripItemCollection.

Table 6.11: ToolStrip Class Members

The following code demonstrates the use of properties and methods of the ToolStrip class.

### Code Snippet:

```
ToolStrip tlstrpTool = new ToolStrip();
ToolStripButton tlstrpNewFile = new ToolStripButton(Image.
FromFile(@"c:\NewFile.jpg"));
ToolStripButton tlstrpFileOpen = new ToolStripButton(Image.
FromFile(@"c:\FileOpen.jpg"));
tlstrpTool.Items.Add(tlstrpNewFile);
tlstrpTool.Items.Add(tlstrpFileOpen);
tlstrpTool.LayoutStyle = ToolStripLayoutStyle.Flow;
this.Controls.Add(tlstrpTool);
```

The code creates an object of ToolStrip class namely, tlstrpTool. Two objects of ToolStripButton class are created with two different images. The Add() method of Items property adds these objects to the ToolStrip control. The LayoutStyle property sets the layout style of ToolStrip control to Flow layout. Finally, the ToolStrip control is added to the current form by using the Add() method.

## Knowledge Check 3

1. Which of these statements about ToolStrip control are false?

(A)	A ToolStrip control can host menu items, which is similar to the menu bar.
(B)	You cannot drag items from one ToolStrip to another.
(C)	A ToolStrip control can reorder items at runtime.
(D)	ToolStrip is the base class for MenuStrip, StatusStrip, and ContextMenuStrip classes.
(E)	A ToolStrip control can create toolbars that provides a common look and feel across the forms.

2. Can you match the properties, methods, and events of the ToolStrip class against their corresponding descriptions?

Description		Property, Method, and Event
(A)	Event that occurs when an item is clicked.	(1) Items
(B)	Property that is used to identify the displayed image.	(2) LayoutStyle
(C)	Method that displays the control to the user.	(3) ImageList
(D)	Property that retrieves all the items of the control.	(4) ItemClicked
(E)	Property that identifies the layout style of the items.	(5) Show

### 6.4 StatusStrip Control

In this fourth lesson, **StatusStrip Control**, you will learn to:

- ➔ Describe the use of StatusStrip control.
- ➔ Explain the properties and events of StatusStrip control.

#### 6.4.1 StatusStrip Control

Consider a scenario where you are working with Microsoft Word document having 150 pages. You can identify the page number and line number by viewing the status bar of the Microsoft Word application. You can also double-click the status bar to go to a particular page by entering the page number. The StatusStrip control of Visual Studio 2005 serves the purpose of the status bar.

The StatusStrip control is used to display useful information about the task being carried out or various controls in use. It is displayed at the bottom of the form. It can be used to display a progress bar too.

### 6.4.2 Properties and Events

The `StatusStrip` control is a new control introduced in .NET Framework 2.0. The default `StatusStrip` control has no panels.

To add panels to a `StatusStrip` control, you use the `ToolStripItemCollection.AddRange` method, or use the `StatusStrip` Items Collection Editor at design time to add, remove, or reorder items and modify properties. Table 6.12 lists the most commonly used properties and event of the `StatusStrip` class.

Name	Description
<code>LayoutStyle</code>	This property specifies or retrieves value indicating how the items are displayed in the control. The different layouts are defined in the <code>ToolStripLayoutStyle</code> enumeration.
<code>Stretch</code>	This property specifies or retrieves a value indicating whether the control stretches from one end to another within the container that holds the control.
<code>Dock</code>	This property specifies or retrieves, which <code>StatusStrip</code> borders are docked to its parent and shows how a <code>StatusStrip</code> is resized with its parent.
<code>ItemAdded</code>	This event occurs when an item is added on the control.

Table 6.12: `StatusStrip` Control Properties and Events

The following code demonstrates how to create and use a `StatusStrip` control.

#### Code Snippet:

```
StatusBar ststrpStatus = new StatusStrip();
ststrpStatus.LayoutStyle = ToolStripLayoutStyle.Table;
ststrpStatus.Stretch = false;
ststrpStatus.Items.Add(DateTime.Now.ToShortDateString());
this.Controls.Add(ststrpStatus);
```

This code creates an object of `StatusStrip` class namely, `ststrpStatus`. The `LayoutStyle` property sets the layout style to Table layout. The `Stretch` property is set to false, which indicates that the control cannot be stretched from end to end within the container. When the control is displayed, today's date is inserted in the control. The `StatusStrip` control is added to the current form using the `Add()` method.

1. You want to display the `StatusStrip` control with the Table layout and also, add 'Page number' text as an item. Which one of the following codes will help you to achieve this?

(A)	<pre>StatusStrip ststrpPage = new StatusStrip(); ststrpPage.Items.Add("Page number"); ststrpPage.LayoutStyle = ToolStripLayoutStyle.Table;</pre>
(B)	<pre>StatusStrip ststrpPage = new StatusStrip(); ststrpPage.Item.Add("Page number"); ststrpPage.LayoutStyle = ToolStripLayoutStyle.Table;</pre>
(C)	<pre>StatusStrip ststrpPage = new StatusStrip(); ststrpPage.Items.Add("Page number"); ststrpPage.LayoutStyle = ToolStripLayoutStyle.Table;</pre>
(D)	<pre>StatusStrip ststrpPage = new StatusStrip(); ststrpPage.Items.Add("Page number"); ststrpPage.LayoutStyle() = ToolStripLayoutStyle.Table;</pre>

## Module Summary

In this module, **MDI Applications and Menus**, you learnt about:

→ **Multiple Document Interface Applications**

Windows Forms allow you to create SDI and MDI applications. MDI applications allow you to access multiple documents simultaneously. MDI applications can have a single parent form, which can invoke multiple child forms. The MDI parent form acts as a base form and the child forms act as sub-forms.

→ **Menus**

Menus are options that allow you to group related options. Menus can be of two types, main menus and context menus. The MenuStrip, StatusStrip, ToolStrip, Menu, ContextMenu classes help you to create and manage menus.

→ **ToolStrip Control**

The ToolStrip control is similar to the toolbar of the Microsoft Word application. It acts as a container to hold related options giving different functionalities similar to holding icons of the toolbar.

→ **StatusStrip Control**

The StatusStrip control is similar to the status bar of Microsoft Word application. It is used to display some critical or useful information to the user. The control contains various panels inside them to group information.

# Technowise



Are you a  
**TECHNO GEEK**  
looking for updates?

Login to

**[www.onlinevarsity.com](http://www.onlinevarsity.com)**

# Module - 7

## Introduction to ADO.NET

Welcome to the Module, **Introduction to ADO.NET**.

This module will explain the role of ActiveX Data Objects.NET (ADO.NET) in providing data access for the .NET-based applications. The module also explains about the different data providers that allow you to establish and maintain connection with the database. Finally, you will learn about data access components that allow you to retrieve data from any data source.

In this Module, you will learn about:

- ➔ ADO.NET Architecture
- ➔ .NET Data Providers
- ➔ Data Access Components

## 7.1 ADO.NET Architecture

In this first lesson, **ADO.NET Architecture**, you will learn to:

- Explain the role of ADO.NET.
- Describe data access architecture in .NET.
- Differentiate between DAO, RDO, ADO, and ADO.NET.
- List the benefits of ADO.NET 2.0.
- Describe the disconnected data access approach.

### 7.1.1 Database Overview

Consider a scenario of a high school, where the school authorities have to manage the personal and academic details of each student. Ever wondered how the school authorities manage such huge records? This can be done by storing the student records in the database.

A database is a collection of related records. The information in the database is stored in such a way that it is easier to access, manage, and update the data. Data from the database can be accessed using any one of the following architectures:

#### → Single-tier Architecture

Single-tier architecture is used to work with local databases. A local database resides on a single machine and thus, any manipulation performed on this database takes place immediately.

#### → Two-tier Architecture

Two-tier architecture is a client/server architecture in which the database is stored and maintained on the server. The client accesses the database by connecting to the database server using database drivers such as the Open Database Connectivity (ODBC) driver.

#### → Three-tier Architecture

Three-tier architecture is a multi-tier client/server architecture in which the client connects to one or more application servers, which in turn connect to the database server. The application server holds the application logic and the database server stores the database.

### 7.1.2 ADO.NET

ADO.NET is the data access technology, which allows you to access data residing in various data sources.

ADO.NET is a part of the .NET Framework, which means that the technology can be used for all .NET-based applications.

ADO.NET technology allows you to connect to database systems such as Microsoft SQL Server, Oracle, Microsoft Access, and Sybase using Open Database Connectivity (ODBC) and Object Linking and Embedding Database (OLE DB) drivers respectively. OLE DB and ODBC are database drivers that are used to connect to the data source. By connecting to the databases, you can thus, retrieve, manipulate, or update the data in the database.

**Note** - Data source is the name of the server to which connection is to be established.

### 7.1.3 Features

ADO.NET supports disconnected data architecture. This means, that the connection to the data source is established only when it is required. Apart from that, ADO.NET uses Extensible Markup Language (XML) to interact with the database. All the data in the database is converted into XML format for database related operations. In order to meet these goals, ADO.NET introduces various features, some of which are listed as follows:

→ **Asynchronous Processing**

Asynchronous processing enables applications to run time-consuming operations in the background and allow the foreground processes to be active throughout the operation.

→ **Multiple Active Result Sets (MARS)**

Multiple active result sets allows the application to execute multiple batches on a single connection. This increases the efficiency of the operation.

→ **XML Data Support**

ADO.NET provides XML support through the `System.XML` namespace. This namespace is used to expose the XML values, which can be easily integrated with the .NET programming model.

→ **Bulk Copy Operations**

Bulk copy operations allow the applications to copy large files into tables or views in SQL Server databases.

→ **Batch Processing**

The INSERT, UPDATE, and DELETE operations are grouped and sent together to the server instead of sending only one operation at a time. This improves the performance of the database operation.

→ Tracing

ADO.NET introduces a new built-in tracing feature that provides information on important matters, such as database unavailability, network library problems, or incorrect programming logic. The tracing feature is used to monitor the execution of code. It helps you to identify problems while executing the code and fix them without interrupting the system that is running.

→ Connection Pooling Control

Connection pooling control collects all the opened database connections in a pool. Connection pool is a list of all available connections, so when a client request a connection it is served directly from the available pool rather than creating a new connection. This helps in cost reduction of repeatedly opening connections.

### 7.1.4 Data Access Architecture

The .NET Framework allows you to access application data either from the local machine or remote machine. ADO.NET data access model and its components are used for processing the data in a database. The two important components of ADO.NET are data providers and Datasets.

Data providers are used for providing and maintaining connection to the database. Datasets can be described as the required portion in the database that is extracted and maintained in the form of a table as a local copy in the client system. This local copy of data can be manipulated and updated independent of the original data source.

### 7.1.5 Different Data Access Models

Data access models allow you to access data from the source. **Data Access Objects (DAO)** was the first data access model that was created for accessing local databases in the Microsoft Jet Database Engine format. **Remote Data Objects (RDO)** and **ActiveX Data Objects (ADO)** were the second and third data access models respectively. Both these models were based on two-tier client-server architecture. ADO provided a connected data access, which means the connection to the database remains open till the application is closed. ADO.NET is the latest data access model that supports disconnected database access. Table 7.1 lists the major differences between the four data access models.

Access Model	Description
DAO	Allows connection for small databases in single-system application files. It is useful to connect to <b>Microsoft Access</b> tables and other databases using ODBC drivers.
RDO	Provides objects that allow connection to a database, execution of queries to work with data, and update the changes to the server.
ADO	Implements connected data access approach. It does not allow transmitting data through firewalls.
ADO.NET	Implements disconnected data access approach. It allows transmitting data through firewalls since the data is in XML format.

Table 7.1: Data Access Models

**Note** - A database engine is a part of Database Management System (DBMS) used to create, retrieve, update, and delete data from a database. Microsoft Jet Database Engine is a database engine on which several Microsoft products were built.

### 7.1.6 Benefits of ADO.NET

ADO.NET 2.0 provides various advantages over its previous versions. Some of the advantages of ADO.NET 2.0 are as follows:

#### → Simplified Programming Model

ADO.NET uses simplified programming model steps such as establishing a connection to a data source, then, creating and executing commands, and finally processing the retrieved results.

#### → Interoperability

ADO.NET widely supports the XML format while working with data access components. XML is the default format used for transmitting datasets across the network, any component that can read the XML format is able to process data. Thus, ADO.NET promotes interoperability.

#### → Maintainability

Sometimes an application may undergo some architectural changes such as increasing the number of tiers for data access. Such changes can be easily incorporated in ADO.NET without affecting the process of data exchange. This is possible because ADO.NET makes use of datasets, which make the data communication through tiers easier.

#### → Programmability

ADO.NET makes use of data commands and data classes that help the user to program quickly and with fewer mistakes.

#### → Performance

ADO.NET offers optimum performance as compared to ADO. This is because ADO.NET does not require data-type conversion while transmitting data through the tiers in the data access architecture.

#### → Scalability

ADO.NET is capable of serving large number of users as it consumes limited resources for data access. Since ADO.NET supports disconnected approach, it does not retain active database connections for long time and can cater to the demands of several users for data connection.

### 7.1.7 Connected Data Access

ADO uses connected data access approach. In this approach, connection to a database is established

when requested by an application. This connection is kept open till the application is closed.

As the connection is left open for the lifetime of the application, it poses problems for database security and also creates unnecessary network traffic. Also, this approach cannot ensure good productivity if the number of users is increased. To overcome these problems, ADO.NET was developed, which uses disconnected data approach.

### 7.1.8 Disconnected Data Access

Consider a multi-national bank that records customer transactions on a central server. These transactions are recorded by establishing a database connection to the server, which stores the account information of the customer. Once the transaction is complete, the connection must be closed. If the connections are not closed, there is a risk of unauthorized access. Such unauthorized access can be minimized using the disconnected data access approach in ADO.NET.

In disconnected data access approach, the connection to the database is established when the application forwards a request. Once the request is processed, the connection to the original database is automatically closed. Thus, ADO.NET conserves system resources and provides maximum security for databases as the connection to the data source is kept open only for a minimum period of time.

### Knowledge Check 1

- Can you match the features of ADO.NET against their corresponding descriptions?

Description		Feature
(A)	Requires no data-type conversion.	(1) Interoperability
(B)	Supports the XML format.	(2) Maintainability
(C)	Serves large number of users.	(3) Programmability
(D)	Unaffected by architectural changes.	(4) Performance
(E)	Uses data commands and data classes.	(5) Scalability

- Which of the following statements about ADO.NET are false?

(A)	Bulk copy operations save time for lengthy operations.
(B)	The connection to original database is closed once a request is processed in disconnected data access model.
(C)	.NET Framework allows you to access application data only from local machine.
(D)	RDO and ADO depends on two-tier client-server architecture.
(E)	ADO.NET technology provides connection to various database systems.

## 7.2 .NET Data Providers

In this second lesson, **.NET Data Providers**, you will learn to:

- Identify the various .NET data providers.

- Describe when to use the appropriate data provider.

### 7.2.1 Data Provider

Data providers are used for providing and maintaining connection to the database. They are a set of related components that work together to provide data in an efficient manner. Data providers allow you to perform different operations in the database and retrieve the results. The data provider, OLE DB, was used earlier for all data sources. Now, the .NET Framework provides different data providers, which are listed as follows:

- **.NET Framework Data Provider for SQL Server**

Data provider for SQL Server allows you to access Microsoft SQL Server databases.

- **.NET Framework Data Provider for OLE DB**

Data provider for OLE DB uses native OLE DB driver to enable data access from relational databases such as Microsoft Access and Oracle. It supports both local and distributed transaction.

- **.NET Framework Data Provider for ODBC**

Data provider for ODBC provides access to ODBC data sources using ODBC Driver Manager. The ODBC Driver Manager allows you to specify the ODBC drivers for your application. It supports both local and distributed transaction.

- **.NET Framework Data Provider for Oracle**

Data provider for Oracle helps you to access Oracle data sources with the help of Oracle client connectivity software. The Oracle client connectivity software consists of network libraries that are used to provide connection to the Oracle database. It supports both local and distributed transaction.

### 7.2.2 System.Data Namespace

The System.Data namespace consists of classes, interfaces, and enumerations that represent the ADO.NET architecture. These classes, interfaces, and enumerations are associated with .NET data providers. These data providers are located in their respective namespaces, which are listed as follows:

- **System.Data.SqlClient**

The System.Data.SqlClient namespace contains .NET Framework data provider for SQL Server. This namespace consists of classes that help you to access and work with SQL Server databases.

- **System.Data.OracleClient**

The System.Data.OracleClient namespace contains .NET Framework data provider for Oracle. This namespace consists of classes that help you to access and work with Oracle databases.

→ **System.Data.OleDb**

The `System.Data.OleDb` namespace contains .NET Framework data provider for OLE DB. This namespace consists of classes that help you to access and work with OLE DB data sources.

→ **System.Data.Odbc**

The `System.Data.Odbc` namespace contains .NET Framework data provider for ODBC. This namespace consists of classes that help you to access and work with ODBC data sources.

### **7.2.3 Choosing Data Provider**

.NET Framework data providers should be appropriately chosen before using them in your application. You can select the different data providers by considering the design and data source of the application.

Table 7.2 lists the details of each data provider that may help you to choose the appropriate data provider for your application.

Data Provider	Description
.NET Framework data provider for SQL Server	<p>Recommended for:</p> <ul style="list-style-type: none"> <li>◆ Middle-tier applications using Microsoft SQL Server 7.0 or later.</li> <li>◆ Single-tier applications using Microsoft Data Engine (MSDE) or Microsoft SQL Server 7.0 or later.</li> <li>◆ SQL Server (SQLOLEDB) with the .NET Framework data provider for OLE DB.</li> </ul>
.NET Framework data provider for OLE DB	<p>Recommended for:</p> <ul style="list-style-type: none"> <li>◆ Middle-tier applications using Microsoft SQL Server 6.5 or earlier, or any OLE DB provider that supports the OLE DB interfaces, such as <code>IDataInitialize</code>, <code>IDBCreateSession</code>, and <code>ICommandText</code>.</li> <li>◆ Single-tier applications using Microsoft Access databases.</li> </ul>
.NET Framework data provider for ODBC	<p>Recommended for:</p> <ul style="list-style-type: none"> <li>◆ Middle-tier applications using ODBC data sources.</li> <li>◆ Single-tier applications using ODBC data sources.</li> </ul>
.NET Framework data provider for Oracle	<p>Recommended for:</p> <ul style="list-style-type: none"> <li>◆ Middle-tier applications using Oracle data sources.</li> <li>◆ Single-tier applications using Oracle data sources.</li> </ul>

Table 7.2: Data Provider

## Knowledge Check 2

1. Which of the following statements about .NET data providers are true?

(A)	Data provider helps in connecting to a database, performing operations and also retrieving results.
(B)	.NET Framework data provider for ODBC uses Oracle client connectivity software to enable data access.
(C)	ODBC Driver Manager allows you to specify the ODBC driver for your application.
(D)	.NET Framework data provider for SQL Server allows access to Microsoft SQL Server.
(E)	.NET Framework data provider for OLE DB enables data access from relational databases by using the connectivity software.

2. Can you match the namespaces against their corresponding descriptions?

Description		Namespace	
(A)	Contains the ODBC .NET data provider types.	(1)	System.Data.OleDb
(B)	Contains classes representing the ADO.NET architecture.	(2)	System.Data.SqlClient
(C)	Contains the OLE DB .NET data provider types.	(3)	System.Data.OracleClient
(D)	Contains the SQL Server .NET data provider types.	(4)	System.Data
(E)	Contains the Oracle .NET data provider.	(5)	System.Data.Odbc

### 7.3 Data Access Components

In this last lesson, **Data Access Components**, you will learn to:

- List the various data access components.
- Explain briefly datasets in ADO.NET.
- Describe the Connection object.
- Describe the Command object.
- Explain the DataAdapter object.
- Describe DataReader objects and their use.

### 7.3.1 Data Access Components

ADO.NET provides two components to access and manipulate data the one is .NET Framework data providers and the other via the `DataSet`.

The .NET Framework data providers are components which allow manipulating data, and giving read-only access to data. `DataSet` is designed for accessing data independent of any data source. Datasets are used to display and update data which are retrieved from a database in form of a table. The `DataSet` represents a complete set of data, including related tables, constraints, and relationships among the tables. ADO.NET consists of objects that allow you to establish connections and work with the database. The objects associated with ADO.NET technology are as follows:

- Command object
- Connection object
- DataAdapter
- DataReader
- DataSet
- DataProvider

Figure 7.1 displays the data access components.

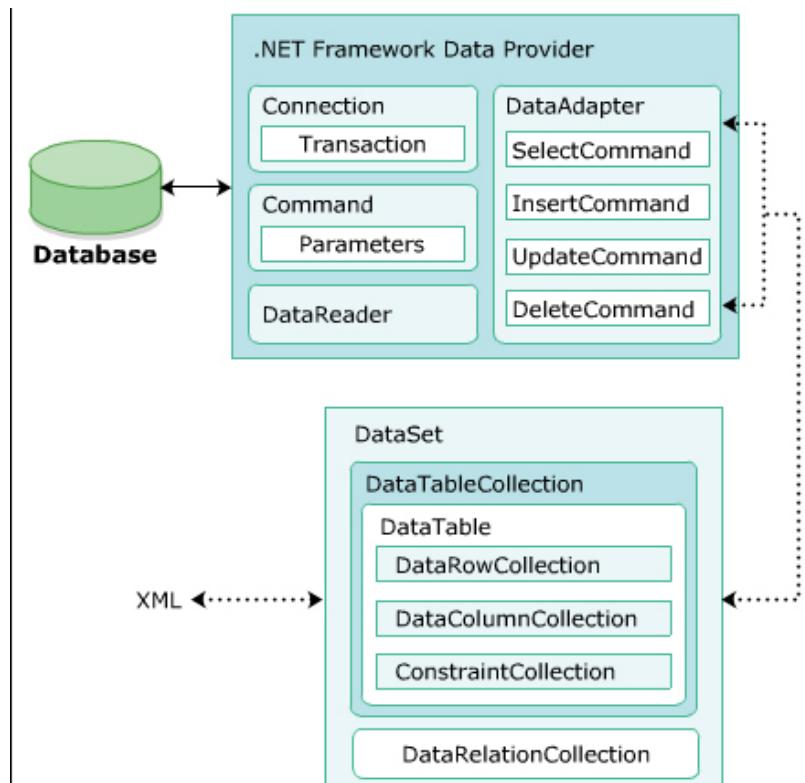


Figure 7.1: Data Access Components

### 7.3.2 DataSet

Consider a multi-national bank, which has many branch offices. Suppose the authorities of the branch office at Sydney want to retrieve some information and use it for their reference. Using datasets, they can connect to the head office, retrieve required information and save this information as a local copy. This allows the authorities to work independently on the retrieved part of the database.

Datasets are used to display and update data. It can be used for retrieving data from multiple sources. The retrieved data is stored as a local copy at the client system. Datasets can perform extensive processing on the data without having an open connection to the server which enables other clients to use the data source. Figure 7.2 displays the `DataSet`.

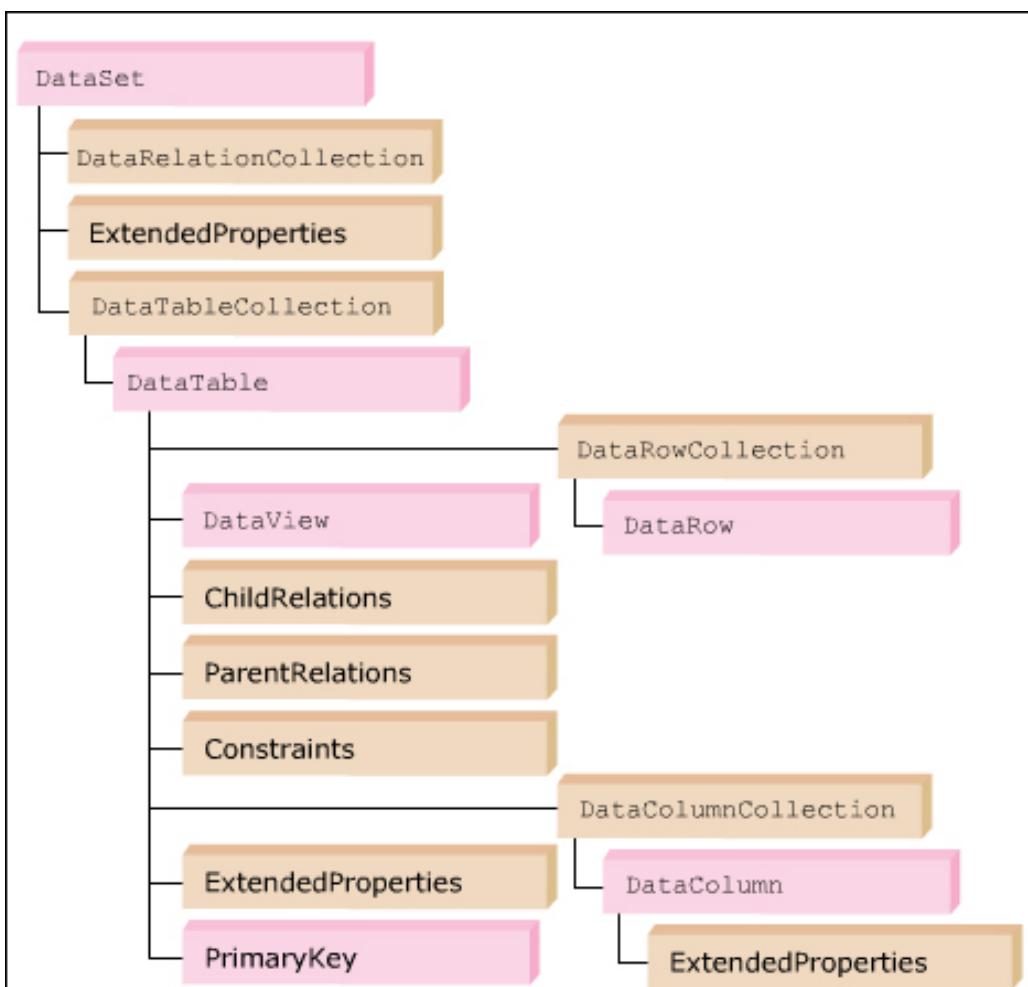


Figure 7.2: DataSet

**Note** - Datasets support the disconnected data access approach. This means the connection to the original database is established only while retrieving and updating data in the database. After that, the connection is automatically closed.

### 7.3.3 Connection Object

The `Connection` object allows you to create a connection between your application and the database. The `Connection` object stores information that is required to establish the connection. By establishing the connection, you can access and manipulate the data present in the database.

The `Connection` object is generally used when you want your application to access the database multiple times. The connection between the application and data source is established only for a specific time in which the data is retrieved or updated.

In ADO.NET, you can establish a connection to SQL Server and OLE DB data sources using the following connection classes:

#### → `SqlConnection` Class

The `SqlConnection` class is used to connect to an SQL Server database. The following is the syntax of `SqlConnection` class.

##### Syntax:

```
SqlConnection <connectionName> = new SqlConnection(<connectionString>);
```

where,

`<connectionString>`: Contains the database connection string.

The following code demonstrates how to use the `SqlConnection` class:

##### Code Snippet:

```
SqlConnection sqlconBank = new SqlConnection("Data
Source=localhost;Initial Catalog=Bank;Integrated Security=SSPI;");
sqlconBank.Open();
MessageBox.Show("Server Version : " + sqlconBank.ServerVersion);
MessageBox.Show("Database : " + sqlconBank.Database);
sqlconBank.Close();
```

The code creates an object of `SqlConnection` class to establish a connection to the specified SQL Server database. The `Data Source` is the name of server to connect to, `Initial Catalog` refers to the name of database, and `Integrated Security` option connects to the data source using Windows integrated security. `ServerVersion` and `Database` are properties of `SqlConnection` class which are used to get the server version and database name. The `Open()` and `Close()` methods of `SqlConnection` class are used to open and close a database connection. `SqlConnection` class is used to open a connection to a SQL Server database.

Table 7.3 lists the commonly used properties, methods, and events of the `SqlConnection` class.

Name	Description
<code>ConnectionString</code>	This property specifies or retrieves the string, which is used to open a SQL Server database.
<code>State</code>	This property specifies or retrieves the current state of the connection.
<code>Close</code>	This method closes the connection to the database.
<code>CreateCommand</code>	This method creates and returns a <code>SqlCommand</code> object, which is associated with <code>SqlConnection</code> .
<code>Open</code>	This method opens a connection to the database with the property settings mentioned by the <code>ConnectionString</code> .
<code>StateChange</code>	This event occurs when the state of the connection is changed.

Table 7.3: `SqlConnection` Class

#### → OleDbConnection Class

The `OleDbConnection` class is used to connect to relational databases using the OLE DB data provider.

The following is the syntax of `OleDbConnection` class.

#### Syntax:

```
OleDbConnection <connectionName> = new OleDbConnection(<connectionString>);
```

where,

`<connectionString>`: Contains the database connection string.

The following code demonstrates how to use the `OleDbConnection` class:

#### Code Snippet:

```
OleDbConnection oledbconBank = new OleDbConnection("Provider=Microsoft.Jet.OLEDB.4.0;
Data Source=c:\\Bank.mdb");
oledbconBank.Open();
MessageBox.Show("ServerVersion:" + oledbconBank.ServerVersion);
MessageBox.Show("DataSource: " + oledbconBank.DataSource);
oledbconBank.Close();
```

The code creates an object of `OleDbConnection` class to establish a connection to the specified Microsoft Access database. The `Provider` specifies which driver to use and `Data Source` is used to specify the location of the database. `ServerVersion` and `DataSource` are properties of `OleDbConnection` class which are used to get the server version and database name.

### 7.3.4 Command Object

The `Command` object enables you to execute a specific command against a data source. The `Command` object stores and executes SQL commands across a data connection. The `Command` object allows you to make a call to a stored procedure. It can be used to execute SQL statements such as `UPDATE`, `DELETE`, `INSERT`, or `SELECT`.

The `Command` object is specific to the data provider used in a connection. For executing commands on SQL Server the `System.Data.SqlClient.SqlCommand` is used and for executing commands on OLE DB data provider the `System.Data.OleDb.OleDbCommand` is used.

#### → Command Object Classes

The `SqlCommand` class specifies the type of operation that can be performed with the SQL Server database. The `OleDbCommand` class specifies any SQL statements or stored procedures that can be performed with any data source using the OLE DB provider. Table 7.4 lists the commonly used properties and methods of the `SqlCommand` class and `OleDbCommand` class.

Name	Description
<code>CommandText</code>	This property specifies or retrieves the commands to be executed at the data source.
<code>Connection</code>	This property specifies or retrieves the <code>SqlConnection</code> or <code>OleDbConnection</code> .
<code>CommandTimeout</code>	This method specifies or retrieves the wait time before ending an attempt to run a command and generating an error.
<code>CommandType</code>	This method specifies or retrieves a value, which indicates the way in which the <code>CommandText</code> property is interpreted.
<code>ExecuteNonQuery</code>	This method executes SQL statements against the connection across the data source and retrieves the number of rows affected.
<code>ExecuteReader</code>	This method sends the <code>CommandText</code> to the <code>Connection</code> and builds a data reader object.
<code>ExecuteScalar</code>	This method executes the database query and retrieves the first column of the first row in the result set.
<code>ExecuteXmlReader</code>	This method sends the <code>CommandText</code> to the <code>Connection</code> and builds an <code>XmlReader</code> object.

Table 7.4: `SqlCommand` Class

You can create an `SqlCommand` object by using the following syntax:

**Syntax:**

```
SqlCommand commandName = new SqlCommand();
```

where,

`SqlCommand` has several overloads, and you can specify the command as a string using the overloaded constructors.

The following code demonstrates how to use the `SqlCommand` class, assuming that a connection has already been created.

**Code Snippet:**

```
SqlCommand sqlcomBank = new SqlCommand();
sqlcomBank.Connection = sqlconBank;
sqlcomBank.CommandText = "CREATE TABLE ProductDetails(Model
varchar(50))";
sqlcomBank.ExecuteNonQuery();
```

The code creates an object of `SqlCommand` class. The `Connection` property is used to assign an object of `SqlConnection` class. The SQL statement is specified using the `CommandText` property. The `ExecuteNonQuery()` method executes the SQL statement and returns the number of rows that are affected.

### 7.3.5 DataAdapter Object

The `DataAdapter` object acts as a bridge between a dataset and a data source. This object updates the database to match it with the data in dataset. It uses the `Fill()` method to fill the dataset with the data from data source.

The `Update()` method is used to insert, update, or delete data from the data source as specified in the dataset. The different data adapters that can be used to connect to the database and dataset are as follows:

- ➔ `OleDbDataAdapter`
- ➔ `SqlDataAdapter`
- ➔ `OdbcDataAdapter`
- ➔ `OracleDataAdapter`

The following code demonstrates how to use the `SqlDataAdapter` class, assuming that a connection has been created.

**Code Snippet:**

```
SqlDataAdapter sqldaSchool = new SqlDataAdapter("SELECT * FROM Student",
sqlconSchool);
DataSet dsetSchoolInfo = new DataSet();
sqldaSchool.Fill(dsetSchoolInfo);
sqldaSchool.Update(dsetSchoolInfo);
```

The code creates an object of `SqlDataAdapter` class. The constructor of this object takes the SQL query and the connection object as parameters.

An object of the `DataSet` class is created for storing the details of student's retrieved using `SqlDataAdapter` object. The `Fill()` and `Update()` methods are used to fill and update the data set.

### 7.3.6 DataReader Object

The `DataReader` object reads a stream of data from the database. It reads one row at a time and moves forward one record at a time. Hence, it provides a forward-only, read-only stream of data.

The advantage of the `DataReader` object is that it increases the application performance by reading only one row at a time. However, the `DataReader` object requires an exclusive use of an open connection object for its whole life span.

The following code demonstrates how to use the `SqlDataReader` class.

#### Code Snippet:

```
SqlCommand sqlcomBank = new SqlCommand("SELECT * FROM Customer",
sqlconBank);
sqlconBank.Open();
SqlDataReader sqldreaderBankInfo = sqlcomBank.ExecuteReader();

// call read before accessing data
while (sqldreaderBankInfo.Read())
{
    MessageBox.Show("Name: " + sqldreaderBankInfo[1]);
}
// call close when finished reading.
sqldreaderBankInfo.Close();
```

This code creates an object of `SqlCommand` class, which specifies the command to be executed. The constructor of this object takes the SQL query and the connection object as parameters.

The `Open()` method opens the `SqlConnection`. The `ExecuteReader()` method executes the query and assigns the output to the object of `SqlDataReader` class. The `Read()` method fetches the next data from the database, which is displayed in message box and the `Close()` method closes the reader.

### Knowledge Check 3

- Can you match the terms against their corresponding descriptions?

Description		Term
(A)	Specifies or retrieves a connection across the data source.	(1) <code>DataAdapter</code> object
(B)	Requires exclusive use of an open Connection object.	(2) <code>Dataset</code> object
(C)	Used for accessing data stores of various types in a uniform manner.	(3) <code>DataReader</code> object
(D)	Acts as a link between a <code>DataSet</code> and a data source.	(4) <code>Connection</code>

	Description		Term
(E)	Enables to update or manipulate data.	(5)	OLE DB

2. Which one of the following codes will help you to create an SQL Server connection and execute SQL commands?

(A)	<pre>SqlConnection sqlconBank = new SqlConnection("Data Source=localhost;Initial Catalog=Bank;Integrated Security=SSPI;");  SqlCommand sqlcomBank = new SqlCommand(); sqlconBank.Open(); sqlcomBank.Connection = sqlconBank; sqlcomBank.CommandText = "DELETE FROM Customer WHERE AccNo=1"; sqlcomBank.ExecuteNonQuery();</pre>
(B)	<pre>SqlConnection sqlconBank = new SqlConnection("Data Source=localhost;Initial Catalog=Bank;Integrated Security=SSPI;");  SqlCommand sqlcomBank = new SqlCommand(); sqlconBank.Open(); sqlcomBank.Connection = sqlconBank; sqlcomBank.CommandText = "DELETE FROM Customer WHERE AccNo=1"; sqlcomBank.ExecuteNonQuery();</pre>
(C)	<pre>SqlConnection sqlconBank = new SqlConnection("Data Source=localhost;Initial Catalog=Bank;Integrated Security=SSPI;");  SqlCommand sqlcomBank = new SqlCommand(); sqlconBank.Open(); sqlcomBank.Connection = sqlconBank; sqlcomBank.CommandText = "DELETE FROM Customer WHERE AccNo = 1"; sqlcomBank.ExecuteNonQuery();</pre>
(D)	<pre>SqlConnection sqlconBank = new SqlConnection("Data Source=localhost;Initial Catalog=Bank;Integrated Security=SSPI;");  SqlCommand sqlcomBank = new SqlCommand(); sqlcomBank.Connection = sqlconBank; sqlcomBank.CommandText = "DELETE FROM Customer WHERE AccNo=1"; sqlcomBank.ExecuteNonQuery();</pre>



## Module Summary

In this module, **Introduction to ADO.NET**, you learnt about:

→ **ADO.NET Architecture**

ADO.NET is a data access technology that allows you to access data from various data sources. It supports disconnected data architecture, which means the connection to the database is established only when it is required.

→ **.NET Data Providers**

A data provider establishes and maintains connection to the database. The .NET Framework provides various data providers which are used for SQL Server, OLE DB, ODBC, and Oracle data sources.

→ **Data Access Components**

.NET Framework data providers and DataSet are components provided by ADO.NET for accessing data source and then, storing the retrieved records into tables. ADO.NET consists of various objects such as Connection, Command, DataAdapter, and DataReader.

# Module - 8

## Data Classes

Welcome to the Module, **Data Classes**.

A DataAdapter is used to connect a dataset to a data source. A dataset is a collection of data from a data source represented in the form of a table. DataSet objects store related tables from a data source and maintain data integrity. It can also contain multiple tables. You can also use DataTable objects to manipulate multiple tables simultaneously.

In this Module, you will learn about:

- ➔ DataAdapter Classes
- ➔ DataReader
- ➔ DataSets
- ➔ DataTables

## 8.1 Data Classes

In this first lesson, **DataAdapter Classes**, you will learn to:

- Explain the use of `DataAdapter` classes.
- Describe briefly the `SqlDataAdapter` class and its use.
- Describe briefly the `OleDbDataAdapter` class and its use.

### 8.1.1 DataAdapter Class

The `DataSet` object stores data from a data source. It retrieves data through data adapters. The data adapter serves as a link between a `DataSet` and a data source for retrieving and saving data.

The `DataAdapter` class consists of a set of SQL commands and a database `Connection` object. These commands and the `Connection` object are used to fill the `DataSet` with the records and update the data source. It forms the base class for all data adapters.

ADO.NET implements various data adapters. These data adapters are inherited from the `DbDataAdapter` class.

The `SqlDataAdapter` is used to connect to an SQL Server database, and the `OleDbDataAdapter` is used to connect to the OLE DB-supported data sources.

The following syntax creates a `SqlDataAdapter` object.

#### Syntax:

```
DataAdapter <objectName> = new SqlDataAdapter();
```

where,

`DataAdapter`: Is the built-in class used to create the `DataAdapter`.

The following code demonstrates a `DataAdapter` object named `daGoods`, for a SQL Server database.

#### Code Snippet:

```
DataAdapter daGoods = new SqlDataAdapter();
```

**Note** - The `DataAdapter` class exists in the `System.Data.Common` namespace.

#### → Properties, Methods, and Events

The different properties, methods, and events of `DataAdapter` class are used to specify whether the changes made to the dataset are accepted while updating the database.

Table 8.1 lists the most commonly used properties, methods, and events of `DataAdapter` class.

Name	Description
AcceptChangesDuringFill	This property specifies or retrieves a value that indicates whether the <code>AcceptChanges()</code> method is called after a row is added to the <code>DataTable</code> . The <code>AcceptChanges()</code> method assigns all changes made to the row.
AcceptChangesDuringUpdate	This property specifies or retrieves a value that indicates whether the <code>AcceptChanges()</code> method is called when the table is updated.
FillLoadOption	This property controls how a <code>DataTable</code> will handle query results that match a row that already exists in the <code>DataTable</code> .
TableMappings	This property retrieves a collection to map the source table to the <code>DataTable</code> .
Fill	This method uses the <code>DataSet</code> name to refresh rows in the <code>DataSet</code> to match those in the data source, and creates a <code>DataTable</code> .
GetFillParameters	This method retrieves the parameters set by the user while executing an SQL SELECT statement.
Update	This method calls the INSERT, UPDATE, or DELETE statements from a <code>DataTable</code> named 'Table' for each row which is inserted, updated, or deleted in the dataset.
FillError	This event takes place when an error occurs during a fill operation.

Table 8.1: DataAdapter Class Members

The following code demonstrates how to use the `DataAdapter` class.

#### Code Snippet:

```
DataAdapter daGoods = new SqlDataAdapter();
DataSet dsetGoods = new DataSet();
daGoods.FillLoadOption = LoadOption.PreserveChanges;
daGoods.AcceptChangesDuringFill = true;
daGoods.Fill(dsetGoods);
daGoods.Update(dsetGoods);

private void daGoods_FillError(object sender, FillErrorEventArgs e)
{
    MessageBox.Show("Error occurred during fill.", "Goods Details");
}
```

The code creates an object named, `daGoods` of `SqlDataAdapter` class. The object of the `DataSet` class is created for retrieving and saving data. The `FillLoadOption` property specifies how the data is filled in the `DataSet`. This is specified using the `PreserveChanges` value defined in the `LoadOption` enumeration.

The `PreserveChanges` value indicates that the current value in the columns will not change. The `AcceptChangesDuringFill` property accepts all changes that occur while the `Fill()` method is used. The `Fill()` and `Update()` methods are used to fill and update the `DataSet`. The `FillError` event is fired whenever an error occurs during the fill operation.

### 8.1.2 `SqlDataAdapter` Class

The `SqlDataAdapter` class helps you to store results of an SQL query in `DataSet` and `DataTable` objects. It is a set of data commands and a database connection used to add data in the `DataSet` and update the SQL Server database.

The `SqlDataAdapter` serves as a link between a `DataSet` and the SQL Server database for retrieving and saving data. While adding data in the `DataSet`, the `SqlDataAdapter` creates necessary tables and columns to save the retrieved data. It can create the schema of the `DataSet` that includes the primary key information. The `SqlDataAdapter` class is used along with its associated `SqlConnection` and `SqlCommand` class to connect to the SQL Server database.

The following code creates a `SqlDataAdapter` object named, `sqldaGoods`.

#### Code Snippet:

```
SqlDataAdapter sqldaGoods = new SqlDataAdapter();
```

**Note** - The `SqlDataAdapter` class exists in the `System.Data.SqlClient` namespace.

#### → Properties and Events

You can use the properties and events of `SqlDataAdapter` class to specify Transact-SQL statements. These properties and events can also be used to specify stored procedures to delete, insert, select, or update records from the `DataSet`.

Table 8.2 lists the most commonly used properties and events of the `SqlDataAdapter` class.

Name	Description
<code>DeleteCommand</code>	This property specifies or retrieves a Transact-SQL statement or stored procedure used to delete records from the <code>DataSet</code> .
<code>InsertCommand</code>	This property specifies or retrieves a Transact-SQL statement or stored procedure used to insert new records into the data source.
<code>SelectCommand</code>	This property specifies or retrieves a Transact-SQL statement or stored procedure used to select records in the data source.
<code>UpdateCommand</code>	This property specifies or retrieves a Transact-SQL statement or stored procedure used to update records in the data source.
<code>RowUpdated</code>	This event occurs when a row is updated in the SQL Server database.
<code>RowUpdating</code>	This event occurs while a row is being updated in the SQL Server database.

Table 8.2: `SqlDataAdapter` Class Members

The following code demonstrates how to connect to the database using the `SqlDataAdapter` class.

**Code Snippet:**

```
SqlDataAdapter sqldaAuto = new SqlDataAdapter();
SqlCommand sqlcomAuto = new SqlCommand("SELECT * FROM
AutomobileDetails", sqlconAuto);
sqldaAuto.AcceptChangesDuringFill = true;
sqldaAuto.SelectCommand = sqlcomAuto;
DataSet dsetAuto = new DataSet();
sqldaAuto.Fill(dsetAuto);
. . .

private void sqldaAuto_RowUpdated(object sender,
SqlRowEventArgs e)
{
    MessageBox.Show("Records affected: " + e.RecordsAffected);
}
```

The code creates an object of the `SqlDataAdapter` class. An object of the `SqlCommand` class is created that takes in the `SELECT` statement as the parameter. The `AcceptChangesDuringFill` property accepts all the changes that occur while the `Fill()` method is used. The `SelectCommand` property is assigned the `SqlCommand` object that contains the `SELECT` statement. An object of the `DataSet` class is created to display the retrieved data. The `RowUpdated` event is fired whenever the user updates a value in a row. A message box is used to state the number of records updated using the `RecordsAffected` property of the `SqlRowEventArgs` class.

### 8.1.3 OleDbDataAdapter Class

The `OleDbDataAdapter` class is a set of data commands and a database connection used to fill the `DataSet` and update data in the data source. It is used to connect to multiple databases such as SQL Server, Oracle, and Microsoft Access.

The `OleDbDataAdapter` serves as a link between a `DataSet` and data source for retrieving and saving data. It uses the `Fill()` method to load data from the data source into the `DataSet`. The changes made in the `DataSet` are applied to the data source by using the `Update()` method.

The following code creates an `OleDbDataAdapter` object named, `oledbdaProducts`.

**Code Snippet:**

```
OleDbDataAdapter oledbdaProducts = new OleDbDataAdapter();
```

**Note** - The `OleDbDataAdapter` class exists in the `System.Data.OleDb` namespace.

## → Properties and Events

The properties and events of the `OleDbDataAdapter` class can be used to specify Transact-SQL statements. They also allow you to specify stored procedures to delete, insert, update, or select records in the `DataSet`. Table 8.3 lists the most commonly used properties and events of the `OleDbDataAdapter` class.

Name	Description
<code>DeleteCommand</code>	This property specifies or retrieves a Transact-SQL statement or stored procedure to delete records from the <code>DataSet</code> .
<code>InsertCommand</code>	This property specifies or retrieves a Transact-SQL statement or stored procedure to insert new records into the data source.
<code>SelectCommand</code>	This property specifies or retrieves a Transact-SQL statement or stored procedure used to select records in the data source.
<code>UpdateCommand</code>	This property specifies or retrieves a Transact-SQL statement or stored procedure used to update records in the data source.
<code>RowUpdated</code>	This event occurs when a row is updated in the data source.
<code>RowUpdating</code>	This event occurs while a row is being updated in the data source.

Table 8.3: `OleDbDataAdapter` Class Members

The following code demonstrates how to connect to the database using the `SqlDataAdapter` class.

### Code Snippet:

```
SqlDataAdapter sqldaAuto = new SqlDataAdapter();
SqlCommand sqlcomAuto = new SqlCommand("SELECT * FROM
AutomobileDetails", sqlconAuto);
sqldaAuto.AcceptChangesDuringFill = true;
sqldaAuto.SelectCommand = sqlcomAuto;
DataSet dsetAuto = new DataSet();
sqldaAuto.Fill(dsetAuto);
.

.

private void sqldaAuto_RowUpdated(object sender,
SqlRowEventArgs e)
{
    MessageBox.Show("Records affected: " + e.RecordsAffected);
}
```

The code creates an object of the `SqlDataAdapter` class. An object of the `SqlCommand` class is created that takes in the `SELECT` statement as the parameter. The `AcceptChangesDuringFill` property accepts all the changes that occur while the `Fill()` method is used. The `SelectCommand` property is assigned the `SqlCommand` object that contains the `SELECT` statement. An object of the `DataSet` class is created to display the retrieved data. The `RowUpdated` event is fired whenever the user updates a value in a row.

A message box is used to state the number of records updated using the `RecordsAffected` property of the `SqlRowUpdatedEventArgs` class.

#### → OleDbDataAdapter Class

The `OleDbDataAdapter` class is a set of data commands and a database connection used to fill the `DataSet` and update data in the data source. It is used to connect to multiple databases such as SQL Server, Oracle, and Microsoft Access.

The `OleDbDataAdapter` serves as a link between a `DataSet` and data source for retrieving and saving data. It uses the `Fill()` method to load data from the data source into the `DataSet`. The changes made in the `DataSet` are applied to the data source by using the `Update()` method.

The following code creates an `OleDbDataAdapter` object named, `oledbdaProducts`.

#### Code Snippet:

```
OleDbDataAdapter oledbdaProducts = new OleDbDataAdapter();
```

**Note** - The `OleDbDataAdapter` class exists in the `System.Data.OleDb` namespace.

#### → Properties and Events

The properties and events of the `OleDbDataAdapter` class can be used to specify Transact-SQL statements. They also allow you to specify stored procedures to delete, insert, update, or select records in the `DataSet`. Table 8.4 lists the most commonly used properties and events of the `OleDbDataAdapter` class.

Name	Description
<code>DeleteCommand</code>	This property specifies or retrieves a Transact-SQL statement or stored procedure to delete records from the <code>DataSet</code> .
<code>InsertCommand</code>	This property specifies or retrieves a Transact-SQL statement or stored procedure to insert new records into the data source.
<code>SelectCommand</code>	This property specifies or retrieves a Transact-SQL statement or stored procedure used to select records in the data source.
<code>UpdateCommand</code>	This property specifies or retrieves a Transact-SQL statement or stored procedure used to update records in the data source.
<code>RowUpdated</code>	This event occurs when a row is updated in the data source.
<code>RowUpdating</code>	This event occurs while a row is being updated in the data source.

Table 8.4: `OleDbDataAdapter` Properties

The following code demonstrates how to connect to the database using the `OleDbDataAdapter` class.

**Code Snippet:**

```
OleDbDataAdapter oledbdaProducts = new OleDbDataAdapter();
OleDbCommand oedbcomProducts = new OleDbCommand("SELECT * FROM
Products", oedbconProducts);
oledbdaProducts.SelectCommand = oedbcomProducts;
oledbdaProducts.UpdateCommand = new OleDbCommand("UPDATE Products
SET ProductName='Mercedes BENZ' WHERE ProductID='P123'", 
oledbconProducts);
DataSet dsetProducts = new DataSet();
oledbdaProducts.Fill(dsetProducts);
. . .

private void oledbdaProducts_RowUpdated(object sender,
OleDbRowUpdatedEventArgs e)
{
    MessageBox.Show("Records affected: " + e.RecordsAffected);
}
```

The code creates an object of the `OleDbDataAdapter` class. An instance of the `OleDbCommand` class is created, which takes the `SELECT` statement and the connection string as parameters. The `SelectCommand` property is assigned the object of the `OleDbCommand` class. The `UpdateCommand` property is used to execute the update statement, which is specified as the parameter to the constructor of the `OleDbCommand` class. The `RowUpdated` event is fired whenever a row is updated in the data source. A message box is used to state the number of records updated using the `RecordsAffected` property of the `OleDbRowUpdatedEventArgs` class.

**Note** - The `OleDbDataAdapter` class exists in the `System.Data.OleDb` namespace.

## Knowledge Check 1

- Which of the following statements of `DataAdapter` classes are true?

(A)	The <code>DataAdapter</code> is used to link the <code>DataSet</code> to a data source.
(B)	The <code>SqlDataAdapter</code> is used to edit data in the SQL Server database and update the <code>DataSet</code> .
(C)	The <code>OleDbDataAdapter</code> class is used to connect to multiple databases such as SQL, Oracle, and Microsoft Access.
(D)	The <code>AcceptChangesDuringFill</code> property indicates how the <code>DataReader</code> object is used to fill the <code>DataSet</code> .
(E)	The <code>Fill()</code> method uses the dataset name to add in the <code>DataSet</code> to match those in the data source.

## 8.2 DataReader

In this second lesson, **DataReader**, you will learn to:

- Explain the working of `SqlDataReader` class.
- Explain the working of `OleDbDataReader` class.

### 8.2.1 SqlDataReader Class

The `SqlDataReader` class is used to read data from a SQL Server database in a sequential manner. It retrieves a forward-only and read-only stream of data when a connection to the database is established. It is used in the connected architecture of ADO.NET and it exists in the `System.Data.SqlClient` namespace.

For retrieving data from the database, you must first connect to SQL Server database using `SqlConnection`. Then, you must call the `ExecuteReader()` method of the `SqlCommand` class to retrieve records from the database. You can read the records by executing the `Read()` method.

The following code demonstrates how to create an object of the `SqlDataReader` class.

#### Code Snippet:

```
SqlDataReader sqldreaderSalary = sqlcomSalary.ExecuteReader();
```

The code creates an instance of the `SqlDataReader` class using the `ExecuteReader()` method. This method belongs to the `SqlCommand` class, whose return type is an object of the `SqlDataReader` class.

#### → Properties

The properties of the `SqlDataReader` class can be used to retrieve the number of columns and to fetch the value of a column. Table 8.5 lists the most commonly used properties of the `SqlDataReader` class.

Property	Description
<code>FieldCount</code>	Retrieves the total number of columns in the current row.
<code>HasRows</code>	Retrieves a value indicating whether the <code>SqlDataReader</code> contains one or more rows.
<code>IsClosed</code>	Retrieves a boolean value indicating whether the specified <code>SqlDataReader</code> instance is open or closed.
<code>Item</code>	Retrieves the column value in its native format.
<code>RecordsAffected</code>	Retrieves the total number of rows changed, inserted, or deleted by the Transact-SQL statements.

Table 8.5: `SqlDataReader` Properties

## → Methods

The methods of the `SqlDataReader` class are used to retrieve the value of a column as a string, 32-bit signed integer, or byte. Table 8.6 lists the most commonly used methods of the `SqlDataReader` class.

Method	Description
<code>Close</code>	Closes the <code>SqlDataReader</code> object.
<code>GetByte</code>	Retrieves the specified column value as a byte.
<code>GetChar</code>	Retrieves the specified column value as a single character.
<code>GetInt32</code>	Retrieves the specified column value as a 32-bit signed integer.
<code>GetName</code>	Retrieves the specified column name.
<code>GetSqlString</code>	Retrieves the specified column value as a <code>SqlString</code> .
<code>GetString</code>	Retrieves the specified column value in the string format.
<code>GetValue</code>	Retrieves the specified column value in its native format.
<code>Read</code>	Allows the <code>SqlDataReader</code> to read the next record.

**Table 8.6: SqlDataReader Methods**

The following code demonstrates how to retrieve the employee ID and name from the `Employees` table using the `SqlDataReader` class.

### Code Snippet:

```

. . .
SqlCommand sqlcomEmployees = new SqlCommand("SELECT * FROM Employees",
sqlconEmployees);
SqlDataReader sqldreaderEmployees = sqlcomEmployees.ExecuteReader();
int numberOffields = sqldreaderEmployees.FieldCount;
MessageBox.Show("Number of Fields: " + numberOffields);
if (sqldreaderEmployees.HasRows)
{
    sqldreaderEmployees.Read();
    txtEmployeeID.Text = sqldreaderEmployees.GetInt32(0).ToString();
    txtEmployeeName.Text = sqldreaderEmployees.GetString(1);
}
if (!sqldreaderEmployees.IsClosed)
{
    sqldreaderEmployees.Close();
}

```

The code creates an object of the `SqlCommand` class, which takes the `SELECT` statement as the parameter. An instance of the `SqlDataReader` class is created using the `ExecuteReader()` method of the `SqlCommand` class. The `FieldCount` property is used to count the total number of columns in the `Employees` table, which is then, displayed in the message box.

The `HasRows` property is used to check if there are any rows in the table. If rows exist, it will return `true` and will invoke the `Read()` method used to read the next record in the table. The `GetInt32()` method is used to convert the employee ID into integer, which is the first column in the table. The `ToString()` method is used to display the ID in the `TextBox` control. The `GetString()` method is used to display the employee name, which is the second column. The `IsClosed` property checks whether the `SqlDataReader` object is closed in the `if` statement. If not closed, the `Close()` method closes the `SqlDataReader` object.

### 8.2.2 OleDbDataReader Class

The `OleDbDataReader` class is used to read data from a data source sequentially. The data source can be MS Access, Sybase, and DB2. The `OleDbDataReader` class can read a single record at a time in the forward direction and in read-only mode.

To fetch records of a table using the `OleDbDataReader` object, you must first create the `OleDbConnection` object. This will allow you to establish a connection with the required data source. Now, invoke the `ExecuteReader()` method of the `OleDbCommand` class to retrieve the records from the required table. Finally, you can now navigate through the records by executing the `Read()` method and can manipulate them.

The following code demonstrates how to create an object of the `OleDbDataReader` class.

#### Code Snippet:

```
OleDbDataReader oledbreaderEmployee = oledbcomEmployee.ExecuteReader();
```

The code creates an instance of the `OleDbDataReader` class using the `ExecuteReader()` method.

#### → Properties

The properties of the `OleDbDataReader` class can be used to get the number of columns in a row, and a column value in its native format. Table 8.7 lists the most commonly used properties of the `OleDbDataReader` class.

Property	Description
<code>FieldCount</code>	Retrieves total number of columns in the current row.
<code>HasRows</code>	Retrieves a value indicating whether the <code>OleDbDataReader</code> contains a single row or multiple rows.
<code>IsClosed</code>	Indicates whether or not the <code>OleDbDataReader</code> is closed.
<code>Item</code>	Retrieves the column value in its native format.

Table 8.7: OleDbDataReader Properties

→ **Methods of OleDbDataReader Class**

You can use the methods of the `OleDbDataReader` class to retrieve the column values in different data types. Table 8.8 lists the most commonly used methods of the `OleDbDataReader` class.

Method	Description
<code>Close</code>	Closes the <code>OleDbDataReader</code> object.
<code>GetData</code>	Retrieves the <code>OleDbDataReader</code> object for the specified column ordinal.
<code>GetInt32</code>	Retrieves the specified column value as a 32-bit signed integer.
<code>GetName</code>	Retrieves the specified column name.
<code>GetOrdinal</code>	Retrieves the specific column ordinal.
<code>GetString</code>	Retrieves the column value in string format.
<code>GetValue</code>	Retrieves the specified column value in its native format.
<code>GetValues</code>	Retrieves all attribute columns in the active row.
<code>NextResult</code>	Moves the <code>OleDbDataReader</code> to the next result, while reading the results of batch SQL statements.
<code>Read</code>	Allows the <code>OleDbDataReader</code> to read the next record.

**Table 8.8: OleDbDataReader Methods**

The following code demonstrates how to retrieve the values from a table using the `OleDbDataReader` class.

**Code Snippet:**

```

OleDbCommand oledbcomSuppliers = new OleDbCommand("SELECT * FROM
Suppliers", oledbconSuppliers);
OleDbDataReader oledbdreaderSuppliers = oledbcomSuppliers.
ExecuteReader();

int numberOfFields = oledbdreaderSuppliers.FieldCount;
MessageBox.Show("Number of Fields: " + numberOfFields);
if (oledbdreaderSuppliers.HasRows)
{
    int supplierID = oledbdreaderSuppliers.GetOrdinal("SupplierID");
    int supplierName = oledbdreaderSuppliers.GetOrdinal("SupplierName");

    oledbdreaderSuppliers.Read();
    txtSupplierID.Text =
    oledbdreaderSuppliers.GetInt32(supplierID).ToString();

    txtSupplierName.Text =
    oledbdreaderSuppliers.GetString(supplierName);
}

oledbdreaderSuppliers.Close();

```

The code creates an object of the `OleDbCommand` class, which takes the `SELECT` statement and the connection object as parameters. An instance of the `OleDbDataReader` class is created using the `ExecuteReader()` method. The `FieldCount` property is used to count the number of columns in the table, which is then, displayed in the message box. The `HasRows` property is used to check if there are any rows in the table. If rows exist, the `GetOrdinal()` method is used to retrieve the column ordinal, which will retrieve the respective column index. The `Read()` method is invoked to move towards the next record to read the data. The supplier ID and its name is retrieved using the `GetInt32()` and `GetString()` methods. These methods take the integer variables as the parameter rather than taking the column names. The `Close()` method closes the `OleDbDataReader` object.

## Knowledge Check 2

- You want to connect to a table named `StudentDetails` in an SQL Server database to retrieve student information. Which one of the following codes will help you to achieve this?

(A)	<pre>SqlConnection sqlconStudentDetails = new SqlConnection("Data Source=MYSERVER\\SQLEXPRESS;Initial Catalog=HolyAngelSchool;Integr ated Security=SSPI");  sqlconStudentDetails.Open();  SqlCommand sqlcomStudentDetails = sqlconStudentDetails. CreateCommand();  sqlcomStudentDetails.Command = "SELECT * FROM StudentDetails";  SqlDataReader sqldreaderStudentDetails = sqlcomStudentDetails. ExecuteReader();  sqldreaderStudentDetails.Read();</pre>
(B)	<pre>SqlConnection sqlconStudentDetails = new SqlConnection("Data Source= MYSERVER\\SQLEXPRESS;Initial Catalog=HolyAngelSchool;Integ rated Security=SSPI");  sqlconStudentDetails.Open();  SqlCommand sqlcomStudentDetails = sqlconStudentDetails. CreateCommand();  sqlcomStudentDetails.CommandText = "SELECT * FROM StudentDetails";  SqlDataReader sqldreaderStudentDetails = sqlcomStudentDetails. ExecuteRead();  sqldreaderStudentDetails.Read();</pre>

(C)	<pre>SqlConnection sqlconStudentDetails = new SqlConnection("Data Source= MYSERVER\\SQLEXPRESS;Initial Catalog=HolyAngelSchool;Integ rated Security=SSPI");  sqlconStudentDetails.Open();  SqlCommand sqlcomStudentDetails = sqlconStudentDetails. CreateCommand();  sqlcomStudentDetails.CommandText = "SELECT * FROM StudentDetails";  SqlDataReader sqldreaderStudentDetails = sqlcomStudentDetails. ExecuteReader();  sqldreaderStudentDetails.ReadNext();</pre>
(D)	<pre>SqlConnection sqlconStudentDetails = new SqlConnection("Data Source= MYSERVER\\SQLEXPRESS;Initial Catalog=HolyAngelSchool;Integ rated Security=SSPI");  sqlconStudentDetails.Open();  SqlCommand sqlcomStudentDetails = sqlconStudentDetails. CreateCommand();  sqlcomStudentDetails.CommandText = "SELECT * FROM StudentDetails";  SqlDataReader sqldreaderStudentDetails = sqlcomStudentDetails. ExecuteReader();  sqldreaderStudentDetails.Read();\</pre>

2. You want to connect to a table named Products in an OLE DB supported database to retrieve the number of fields. Which one of the following code will help you to achieve this?

(A)	<pre>OleDbCommand oledbcomProducts = new OleDbCommand("SELECT * FROM Products");  OleDbDataReader oledbreaderProducts = oledbcomProducts. ExecuteReader();  int numberOfFields = oledbreaderProducts.FieldCount;  MessageBox.Show("Number of Fields: " + numberOfFields);</pre>
(B)	<pre>OleDbCommand oledbcomProducts = new OleDbCommand("SELECT * FROM Products");  OleDbDataReader oledbreaderProducts = oledbcomProducts. ExecuteReader();  int numberOfFields = oledbreaderProducts.FieldCount();  MessageBox.Show("Number of Fields: " + numberOfFields);</pre>

(C)	<pre>OleDbCommand oledbcomProducts = new OleDbCommand("SELECT * FROM Products", oledbconProducts); OleDbDataReader oledbreaderProducts = oledbcomProducts.ExecuteReader(); int numberOfFields = oledbreaderProducts.FieldCount; MessageBox.Show("Number of Fields: " + numberOfFields);</pre>
(D)	<pre>OleDbCommand oledbcomProducts = new OleDbCommand("SELECT * FROM Products", oledbconProducts); OleDbDataReader oledbreaderProducts = oledbcomProducts.ExecuteReader(); int numberOfFields = oledbreaderProducts.FieldCount(); MessageBox.Show("Number of Fields: " + numberOfFields);</pre>

## 8.3 DataSets

In this third lesson, **DataSets**, you will learn to:

- ➔ Describe datasets.
- ➔ Explain typed and untyped datasets.
- ➔ Identify the process of fetching XML data through datasets.
- ➔ Describe the process of accessing records.

### 8.3.1 DataSet

A **DataSet** object is a representation of database objects in the cached memory. The database objects include relational tables and constraints. A **DataSet** object indirectly interacts with the data source. It can only hold data retrieved from the data source. You can fetch the data in the **DataSet** using a **DataAdapter** and can modify the fetched data. There are various features of datasets. These are as follows:

#### ➔ Working with Disconnected Data

Once the data is fetched in the **DataSet**, the **DataSet** is disconnected from the database. You can then, make changes to the data in the **DataSet**. These changes can be saved back in the database by re-establishing the connection to the database.

#### → Working with Hierarchical Data

The `DataSet` objects store data in a hierarchical manner. This allows you to easily work with the related tables from a database.

#### → Caching Changes

The `DataSet` objects allow you to make changes to a row even while it is disconnected with the database. Therefore, `DataSet` objects contain multiple versions of each row in a cached memory. The latest rows are then, updated in the database using a `DataAdapter`, when the connection is established.

#### → Supporting XML Integration

You can retrieve data from an XML file into the `DataSet` and update XML data from `DataSet` to the XML file. You can also separate table, column, and constraint information and save it into an XML schema.

#### → Implementing Uniform Functionality

The `DataSet` object provides consistent functionalities irrespective of the type of data source and `DataAdapter` used.

The following syntax is used to create a `DataSet` object.

**Syntax :**

```
DataSet <objectName> = new DataSet();
```

#### → Properties

You can use the properties of the `DataSet` class to retrieve the name and collection of tables in a `DataSet`. Table 8.9 lists the most commonly used properties of the `DataSet` class.

Property	Description
<code>DataSetName</code>	Specifies or retrieves the current <code>DataSet</code> name.
<code>GetType</code>	Retrieves the type of the current instance.
<code>Locale</code>	Specifies or retrieves information used to compare strings in a table.
<code>Relations</code>	Retrieves the collection of relations that link tables and allow navigation from parent to child tables.
<code>Tables</code>	Retrieves the collection of relations, which associate tables and allow browsing from the parent to child tables.

Table 8.9: `DataSet` Properties

## → Methods

You can use the methods of the `DataSet` class to read and write XML data or schema from the `DataSet`. Table 8.10 lists the most commonly used methods of the `DataSet` class.

Method	Description
<code>AcceptChanges</code>	Commits all the changes made to the <code>DataSet</code> .
<code>Clear</code>	Removes all the rows of all tables in the <code>DataSet</code> .
<code>GetXml</code>	Retrieves the XML representation of the stored data.
<code>Merge</code>	Merges a particular <code>DataSet</code> , <code>DataTable</code> , or array of <code>DataRow</code> objects into the <code>DataSet</code> or <code>DataTable</code> .
<code>ReadXml</code>	Reads XML schema and data into the <code>DataSet</code> .
<code>RejectChanges</code>	Rejects all the changes made to the <code>DataSet</code> , since it was created or since the last time the <code>AcceptChanges ()</code> method was called.
<code>WriteXml</code>	Writes XML data or schema from the <code>DataSet</code> .

**Table 8.10: DataSet Methods**

The following code demonstrates how to fill the `DataSet` with the records of a table using the `DataSet` class.

### Code Snippet:

```
.
.
.
SqlDataAdapter sqldaStudents = new SqlDataAdapter("SELECT * FROM Students", sqlconStudents);
DataSet dsetStudents = new DataSet();
dsetStudents.DataSetName = "Students";
int totalRecords = sqldaStudents.Fill(dsetStudents, "Students");
MessageBox.Show("Total Records: " + totalRecords);
dsetStudents.AcceptChanges();
MessageBox.Show("Total Tables: " + dsetStudents.Tables.Count);
dsetStudents.Clear();
```

The code creates an object named, `dsetStudents` of the `DataSet` class. The `DataSetName` property sets the name of the dataset to `Students`. The total number of records fetched in the `DataSet` is displayed in the message box. The `AcceptChanges ()` method is used to accept the changes in the `DataSet`. The `Count` property is used to display the total number of tables in the `DataSet`. The `Clear ()` method is used to clear the `DataSet`.

### 8.3.2 Typed Datasets

`DataSet` objects can either be typed or untyped. A typed `DataSet` is a class derived from the `DataSet` class. The typed `DataSet` inherits all the properties, methods, and events of the `DataSet` class. Hence, it can be used with methods that take an instance of a `DataSet` class as a parameter. You can also use typed `DataSet` to access tables and columns by names, instead of using properties.

When the typed `DataSet` is created, three derived classes, namely `DataTable`, `DataRow`, and

`DataRowChangeEvent` are generated for each table in the `DataSet`. These classes have specific schema, properties, and methods for the associated tables. The typed `DataSet` objects can catch any type mismatch errors at compile-time.

The following code demonstrates how the table name is directly used by creating a typed `DataSet`.

#### Code Snippet:

```
...
SqlDataAdapter sqldaSuppliers = new SqlDataAdapter("SELECT * FROM Suppliers", sqlconSuppliers);
Suppliers dsetSuppliers = new Suppliers();
sqldaSuppliers.Fill(dsetSuppliers, "Suppliers");
txtSupplierID.Text = dsetSuppliers.Suppliers[0].SupplierID.ToString();
txtSupplierName.Text = dsetSuppliers.Suppliers[0].SupplierName;
```

An XSD file namely, `Suppliers.xsd` is already created for the `DataSet` by adding the file to the application. It is only with this schema that you can create a typed `DataSet`. An instance of a typed `DataSet` is created and is filled with the records of the `Suppliers` table. The supplier ID and name are retrieved by directly referring to the table name.

#### → Untyped Datasets

An untyped dataset does not have a built-in schema. In this dataset, the tables and columns are represented as a collection of data.

The untyped dataset is useful when working on data which does not have a static or predictable structure. You can add tables and columns to the untyped dataset by setting the properties at design time or by adding them at run time.

The following code demonstrates how the `Tables` property is used to access table records with an untyped dataset.

#### Code Snippet:

```
...
sqlDataAdapter sqldaSuppliers = new SqlDataAdapter("SELECT * FROM Suppliers", sqlconSuppliers);
DataSet dsetSuppliers = new DataSet();
sqldaSuppliers.Fill(dsetSuppliers, "Suppliers");

txtSupplierID.Text = dsetSuppliers.Tables["Suppliers"].Rows[0]
["SupplierID"].ToString();
txtSupplierName.Text = dsetSuppliers.Tables["Suppliers"].Rows[0]
["CompanyName"].ToString();
```

The code creates an untyped dataset and fills it with the records of the `Suppliers` table. The `Tables` property is used to retrieve the supplier ID and company name from the `Suppliers` table. The table name cannot be used directly to access the records with an untyped dataset.

### 8.3.3 Fetching XML Data Using Datasets

DataSet objects can fetch data from an XML file. To fetch XML data using DataSet, you must first read the XML file into a DataSet object. This is done by using ReadXml() method. The ReadXml() method contains the path of the XML file. This method reads the XML data including the schema of the XML file.

You can make changes to the XML data in the DataSet and update the changes to the XML source file using the WriteXml() method.

You can create an XML representation of data fetched from a database into the DataSet. If the XML representation includes the schema, it is written using the XML Schema definition language (XSD). The schema contains the tables, relations, and constraint definitions.

The following code demonstrates how to read data from a XML file.

#### Code Snippet:

```
DataSet dsetRead = new DataSet();
dsetRead.ReadXml("authors.xml");
```

The code creates an object named, dsetRead of the DataSet class. The ReadXml() method reads data from the XML file.

### 8.3.4 Navigating Records

DataSet objects treat a table as a collection and allow you to access rows using the index of the collection. The easiest way to navigate through records is to bind the data source to the DataSet using the BindingSource class. You can use the built-in methods of the BindingSource class, such as MoveNext, MoveLast, MovePrevious, and MoveFirst to navigate to a particular record in the DataSet.

The following code demonstrates how to retrieve the product names from the Products table using the Dataset class.

#### Code Snippet:

```
SqlDataAdapter sqldaProducts = new SqlDataAdapter("SELECT * FROM Products",
sqlconProducts);
DataSet dsetProducts = new DataSet();
sqldaProducts.Fill(dsetProducts,"Products");
int rowNumber = 0;

while (rowNumber < dsetProducts.Tables[0].Rows.Count)
{
    cboProducts.Items.Add(dsetProducts.Tables["Products"].
    Rows[rowNumber]["ProductName"].ToString());
    rowNumber++;
}
```

The code fills the `DataSet` with the records of the `Products` table. In the `while` loop, the product names are fetched and added to the `ComboBox` control.

### Knowledge Check 3

- Which of the following statements regarding datasets are true?

(A)	The typed <code>DataSet</code> is inherited from the <code>DataSet</code> class.
(B)	The <code>ReadXml()</code> method is used to update the XML file.
(C)	The <code>MoveNext()</code> method of the <code>BindingSource</code> class is used to navigate to a record in the <code>DataSet</code> .
(D)	The untyped dataset is used on data that do not have static or predictable structure.
(E)	The typed <code>DataSet</code> is used to catch any wrong values fetched from a table.

- Can you match the properties and methods of the `DataSet` class against their corresponding descriptions?

Description		Properties and Methods
(A)	This property specifies or retrieves the locale information used to compare strings in a table.	(1) <code>ReadXml</code>
(B)	This property specifies or retrieves the <code>DataSet</code> name.	(2) <code>Clear</code>
(C)	This method removes rows in all the tables.	(3) <code>GetType</code>
(D)	This method retrieves the type of the current instance.	(4) <code>Locale</code>
(E)	This method retrieves XML schema and data into the <code>DataSet</code> .	(5) <code>DataSetName</code>

### 8.4 DataTables

In this last lesson, **DataTables**, you will learn to:

- Explain `DataTable` class and its use.
- Describe the `DataTableCollection` class.
- Define a relation and describe the process of setting relation between tables.
- Describe the  `DataColumn` class.
- Describe the  `DataRow` class.
- Describe the  `DataTableReader` class.

### 8.4.1 DataTable Class

A `DataTable` class represents a table in the `DataSet`. It is used to store data from a data source. It contains a set of rows and columns to organize data in a tabular format. It is widely used in the disconnected environment. A `DataTable` object can be implemented for any database. It can store the results of SQL queries, which can be later accessed or manipulated using the `DataRow` and `DataColumn` objects.

The `DataTable` objects are conditionally case-sensitive. For example, if one `DataTable` is named as `newtable` and another is named `Newtable`, then, the string casing must match with the casing of the required table name.

The `DataTable` contains constraint objects that ensure the data integrity in the related tables.

The `DataTable` class contains various properties, methods, and events that allow interaction with the database. It helps to handle data in a better and an efficient way.

#### → Properties

The properties of the `DataTable` class are used to retrieve the rows and columns of a table. Table 8.11 lists the most commonly used properties of the `DataTable` class.

Property	Description
<code>ChildRelations</code>	Retrieves a collection of child relations for the current <code>DataTable</code> .
<code>Columns</code>	Retrieves the collection of columns of the table.
<code>DataSet</code>	Retrieves the <code>DataSet</code> to which the table belongs.
<code>IsInitialized</code>	Retrieves a value indicating whether or not the <code>DataTable</code> is initialized.
<code>ParentRelations</code>	Retrieves a collection of parent relations for the current <code>DataTable</code> .
<code>Rows</code>	Retrieves the collection of rows of the table.
<code>TableName</code>	Specifies or retrieves the name of the <code>DataTable</code> .

Table 8.11: `DataTable` Properties

#### → Methods

The methods of the `DataTable` class are used to add new rows and to retrieve XML schema into the `DataTable`. Table 8.12 lists the most commonly used methods of the `DataTable` class.

Method	Description
<code>AcceptChanges</code>	Assigns all the changes made to the table, since the last time <code>AcceptChanges</code> was invoked.
<code>Clear</code>	Removes all the data from the <code>DataTable</code> .
<code>Load</code>	Fills a <code>DataTable</code> with the records from a data source by using the specified <code>IDataReader</code> . The requested data is combined with the existing rows, if the <code>DataTable</code> has some rows.
<code>NewRow</code>	Creates a new row with the schema of the table.

Method	Description
ReadXml	Reads XML schema and data into the DataTable.
Select	Retrieves an array of DataRow objects.
WriteXML	Writes the data in the DataTable in the XML format.
ToString	Retrieves the name of table and displays it in the string format.

Table 8.12: DataTable Methods

## → Events

Table 8.13 lists the most commonly used events of the DataTable class.

Event	Description
ColumnChanged	Occurs after a value in the specified DataColumn is changed in a DataRow.
RowChanged	Occurs after a DataRow is changed.
RowDeleted	Occurs after a row is deleted.
TableNewRow	Occurs when a new DataRow is inserted.

Table 8.13: DataTable Events

The following code demonstrates how to retrieve the order ID from the first row of the Orders table using the DataTable class.

### Code Snippet:

```
...
SqlDataAdapter sqldaOrders = new SqlDataAdapter("SELECT * FROM Orders", sqlconOrders);
DataTable dtableOrders = new DataTable();
dtableOrders.TableName = "Orders";
dtableOrders.AcceptChanges();
sqldaOrders.Fill(dtableOrders);
txtEmployeeID.Text = dtableOrders.Rows[0]["OrderID"].ToString();
```

The code creates an object of the DataTable class and fills it with the records of the Orders table. The TableName property sets the name of the table. The AcceptChanges() method applies the changes to the table, if any. The Rows property is used to retrieve the order ID appearing in the first row.

### 8.4.2 DataTableCollection Class

The DataTableCollection class is a collection of tables for a DataSet. It enables you to access the collection of DataTable objects and navigate through the tables in the DataSet.

You can use the properties, methods and events of the DataTableCollection class to manage multiple tables for the DataSet.

Table 8.14 lists the most commonly used properties, methods, and events of the `DataTableCollection` class.

Name	Description
Count	This property retrieves the total number of elements in a collection.
Item	This property retrieves a <code>DataTable</code> object from the collection.
Add	This method inserts a <code>DataTable</code> object to the collection.
Clear	This method removes all the <code>DataTable</code> objects from the collection.
Remove	This method removes a specified <code>DataTable</code> object from the collection.
CollectionChanged	This event occurs after adding or removing objects in the collection.

Table 8.14: `DataTableCollection` Members

The following code demonstrates how to add tables to a `DataTableCollection` object.

#### Code Snippet:

```
SqlDataAdapter sqldaOrders = new SqlDataAdapter("SELECT * FROM Orders",
sqlconOrders);
DataSet dsetOrders = new DataSet();
sqldaOrders.Fill(dsetOrders, "Orders");

sqldaOrders = new SqlDataAdapter("SELECT * FROM [Customers]",
sqlconOrders);
sqldaOrders.Fill(dsetOrders, "Customers");
DataTableCollection dtabcollOrders = dsetOrders.Tables;

foreach (DataTable dtabc in dtabcollOrders)
{
    cboProducts.Items.Add(dtabc.TableName);
}

DataTable dtabcSuppliers = new DataTable("Suppliers");
sqldaOrders = new SqlDataAdapter("SELECT * FROM Suppliers", sqlconOrders);
sqldaOrders.Fill(dtabcSuppliers);
dtabcollOrders.Add(dtabcSuppliers);

private void dtabcollOrders_CollectionChanged(object sender,
CollectionChangeEventArgs e)
{
    MessageBox.Show("The data table collection is changed");
}
```

The code fills the dataset with two tables namely, `Orders` and `Customers`. An object of the `DataTableCollection` class is created. The tables in the `DataSet` are assigned to this object using the `Tables` property of the `DataSet` class. The `for` loop is used to iterate through each table and the table name is displayed in the `ComboBox` control. An object of the `DataTable` class is created that refers to the `Suppliers` table. This table is added to the `DataTableCollection` object using the `Add()` method.

This triggers the `CollectionChanged` event, which inserts a table to the `DataTableCollection` object. A message box appears that displays a message about the changed collection.

### 8.4.3 Setting Relation between Tables

A parent or child relationship between two or more tables in the `DataSet` is represented by the `DataRelation` object. The `DataRelation` object allows navigation between  `DataTables` within the `DataSet`. It provides related records that are currently being worked upon.

The `DataRelation` object maintains referential integrity by enforcing foreign-key constraint. The `DataRelation` object also, enforces a unique constraint to ensure that there are no duplicate entries in a column of a table. It is also useful for performing cascade update and delete operations.

#### → Adding Relation between Tables

The `DataRelation` class is used to relate two tables through  `DataColumn` objects. For adding relation between tables, you must specify the common column that exists in both the tables. Then, the `Add()` method is used to add the relation between the two tables. The properties of the column can be used to fetch the foreign key constraints and unique key constraints. Table 8.15 lists the most commonly used properties of the `DataRelation` class.

Property	Description
<code>ChildColumns</code>	Retrieves the child <code> DataColumn</code> objects of the current relation.
<code>ChildKeyConstraint</code>	Retrieves the <code> ForeignKeyConstraint</code> of the relation.
<code>ChildTable</code>	Retrieves the child table of the relation.
<code>DataSet</code>	Retrieves the <code> DataSet</code> of the relation.
<code>ParentColumns</code>	Retrieves an array of <code> DataColumn</code> objects, which are the parent columns of the relation.
<code>ParentKeyConstraint</code>	Retrieves the <code> UniqueConstraint</code> , which assures that the values in the parent column are unique.
<code>ParentTable</code>	Retrieves the parent <code> DataTable</code> of the relation.
<code>RelationName</code>	Specifies or retrieves the name for retrieving <code> DataRelation</code> from the <code> DataRelationCollection</code> .

Table 8.15: `DataRelation` Properties

**Note** - The `DataRelation` object, by default, adds a unique constraint to the parent table and a foreign key constraint to the child table.

The following code demonstrates how to add a relation to the  `DataSet` object.

**Code Snippet:**

```
...
SqlDataAdapter sqldaInvoice = new SqlDataAdapter("SELECT * FROM
Invoice",
```

```

sqlconInvoice);
DataSet dsetInvoice = new DataSet();
sqldaInvoice.Fill(dsetInvoice,"Invoice");
sqldaInvoice = new SqlDataAdapter("SELECT * FROM ProductDetails",
sqlconInvoice);
sqlda Invoice.Fill(dsetInvoice, "ProductDetails");

DataRelation drelationInvoice = new DataRelation("InvoiceDetails",
dsetInvoice.Tables["Invoice"].Columns["InvoiceNo"],
dsetInvoice.Tables["ProductDetails"].Columns["InvoiceNo"]);
dsetInvoice.Relations.Add(drelationInvoice);

```

The code fills the `DataSet` with the records of the `ProductDetails` table. An instance of the `DataRelation` class is created namely, `drelationInvoice`. The constructor of the class takes the table name and the common columns of the two tables as the parameters. Then, the `Relations` property of the `DataSet` class is used to add the relation to the `DataSet`.

#### 8.4.4 DataColumn Class

The `DataColumn` class is used for creating a schema for a particular column. It is the basic element for `DataTable`. The schema is used to define the data type and constraints. You can add `DataColumn` objects to the `DataColumnCollection` to create the schema of the `DataTable`.

The properties of the `DataColumn` class allow you to retrieve data type of the column and to set the maximum length of the column. Table 8.16 lists the most commonly used properties of the `DataColumn` class.

Property	Description
Caption	Specifies or retrieves the column caption.
ColumnName	Specifies or retrieves the name of the column in the <code>DataColumnCollection</code> .
DataType	Specifies or retrieves the type of data stored in the column.
DefaultValue	Specifies or retrieves the default value of a column, while adding new rows.
MaxLength	Specifies or retrieves the maximum length of a text column.
Table	Retrieves a <code>DataTable</code> to which the column belongs.
Unique	Specifies or retrieves a value indicating whether the values in each row of the column should be unique.

Table 8.16: `DataColumn` Properties

The following code demonstrates how a new column is added to the `Products` table using the `DataColumn` class.

##### Code Snippet:

```

//Code for database connection
...
DataTable dtableProducts = new DataTable("Products");
DataColumn dcolPrice = new DataColumn();

```

```
dcolPrice.DataType = Type.GetType("System.Decimal");
dcolPrice.Caption = "Price";
dcolPrice.ColumnName = "Price";
dcolPrice.DefaultValue = 25;

datatableProducts.Columns.Add(dcolPrice);
```

The code creates an object of `DataColumn` class. The data type is set to `Decimal` using the `DataType` property. The `Caption` property is used to set the column name as `Price`, which will be displayed as the column header. The `DefaultValue` property sets the default value in the column as `25`. The new column is now added to the `Products` table using the `Add()` method.

#### 8.4.5 `DataRow` Class

The `DataRow` class represents a row in a `DataTable`. The object of the class is a basic element of the `DataTable`. The class is used to insert, update, and delete values in the `DataTable`. You can use the properties and methods of the `DataRow` class to specify or retrieve data in a row and to delete a row. Table 8.17 lists the most commonly used properties and methods of a `DataRow` class.

Name	Description
Item	This property specifies or retrieves data stored in a specified column.
RowError	This property specifies or retrieves the user-defined error description for a row.
RowState	This property retrieves the current row state in relation with the <code>DataRowCollection</code> . The <code>DataRowCollection</code> class represents an array of rows in the <code>DataTable</code> . The values of this property are defined in the <code>DataRowState</code> enumeration.
Table	This property retrieves the <code>DataTable</code> for which the current row has a schema.
AcceptChanges ()	This method makes all the changes made to the current row, since the last time the method was called.
Delete ()	This method deletes the <code>DataRow</code> .
SetAdded ()	This method manipulates the <code>RowState</code> of a <code>DataRow</code> to be added.

Table 8.17: `DataRow` Class Members

The following code demonstrates how a new row is added to the `DataTable` object table using the `DataRow` class.

##### Code Snippet:

```
//Code for database connection
...
DataTable dtableEmployee = new DataTable("Employees");
DataColumn dcolEmployeeID = new DataColumn();
dcolEmployeeID.DataType = Type.GetType("System.String");
dcolEmployeeID.Caption = "EmployeeID";
```

```

dcoleEmployeeID.ColumnName = "EmployeeID";
dtableEmployee.Columns.Add(dcoleEmployeeID);
DataRow drowEmployee = dtableEmployee.NewRow();
drowEmployee["EmployeeID"] = "E001";
drowEmployee.RowError = "An error has taken place while dealing with a
row";
dtableEmployee.Rows.Add(drowEmployee);

```

The code adds a new column to the Employees table namely, EmployeeID. An instance of the DataRow class is created using the `NewRow()` method of the DataTable class. The value in the EmployeeID column is set to E001. The RowError property is assigned a custom string to be displayed if any error occurs while handling rows of the table. Finally, the value of the employee ID is added to the table.

#### 8.4.6 DataTableReader Class

The `DataTableReader` class is used to retrieve data from one or more `DataTable` objects. It retrieves data in forward-only and read-only mode. It is a new class introduced in .NET Framework 2.0. It allows you to iterate rows in the cache memory.

You can modify the table in the `DataSet` only when the `DataTableReader` is active. When the data is modified, the `DataTableReader` automatically maintains its position in the `DataSet`.

When you create a `DataTableReader` object from a `DataTable`, the `DataTableReader` object contains the same data in the same order as in the `DataTable`. It can contain multiple tables in the same sequence as present in the `DataTableCollection`. The `DataTableReader` object contains only the latest version of each row in the table. The deleted rows are taken into consideration.

##### → Properties

The properties of the `DataTableReader` class can be used to get the total number of columns of a row and to obtain a column value. Table 8.18 lists the most commonly used properties of the `DataTableReader` class.

Property	Description
FieldCount	Retrieves total number of columns in the active row.
IsClosed	Retrieves a value indicating whether the <code>DataTableReader</code> object is closed.
Item	Retrieves the value of a particular column in its native format.
RecordsAffected	Retrieves the number of rows added, updated, or removed by executing the SQL statement.

Table 8.18: `DataTableReader` Properties

## → Methods

The methods of the `DataTableReader` class can be used identify the data type of a column and to fetch the next record.

Table 8.19 lists the most commonly used methods of the `DataTableReader` class.

Method	Description
<code>Close</code>	Closes the current <code>DataTableReader</code> object.
<code>GetData</code>	Retrieves a <code>DbDataReader</code> object for the specified column ordinal.
<code>GetDataTypeName</code>	Retrieves a string, which represents the data type of a particular column.
<code>GetName</code>	Retrieves the value of a particular column in the string format.
<code>GetString</code>	Retrieves the value of a given column in the string format.
<code>GetValue</code>	Retrieves the value of a given column in its native format.
<code>NextResult</code>	Moves the <code>DataTableReader</code> object to the next result set, if any more result sets exist.
<code>Read</code>	Moves the <code>DataTableReader</code> object to the next record.

**Table 8.19: DataTableReader Methods**

The following code demonstrates how to retrieve all the customer names from the `Customers` table using the `DataTableReader` class.

### Code Snippet:

```
SqlDataAdapter sqldaCustomers = new SqlDataAdapter("SELECT * FROM Customers", sqlconCustomers);
DataTable dtableCustomers = new DataTable("Customers");
sqldaCustomers.Fill(dtableCustomers);

DataTableReader dtblrdrCustomers = new DataTableReader(dtableCustomers);
while (dtblrdrCustomers.Read())
{
    cboCustomers.Items.Add(dtblrdrCustomers["CustomerName"]);
}
dtblrdrCustomers.Close();
```

The code fills the `DataTable` object with the records of the `Customers` table. An instance of the `DataTableReader` class is created which takes the `DataTable` object as the parameter. The `Read()` method is used to iterate through each record in the table using the `while` loop. In the loop, the customer names are retrieved from the table and are added to the `ComboBox` control. The `Close()` method is used to close the `DataTableReader` object.

## Knowledge Check 4

1. Which of the following statements of the `DataTable`, `DataRelation`,  `DataColumn`, `DataRow`, and `DataTableReader` classes are true and which statements are false?

(A)	The <code>DataTable</code> object is used as a case-sensitive object if the casing of two tables with same name differs.
(B)	The <code>DataRow</code> and <code> DataColumn</code> classes are used as the primary components of the <code>DataTable</code> class.
(C)	The <code>DataRelation</code> objects are used to enter duplicate values in a column of a table.
(D)	The <code>DataRelation</code> objects are used to enforce referential integrity.
(E)	The <code>DataTableReader</code> object is used to return the records of a table in editable mode.

2. Can you match the properties and methods of the `DataTable` class against their corresponding descriptions?

Description		Property, Method, and Event
(A)	This property specifies or retrieves the table name.	(1) RowDeleted
(B)	This property retrieves the <code>DataSet</code> belonging to the table.	(2) TableNewRow
(C)	This event occurs on deletion of a row in the table.	(3) Clear()
(D)	This event occurs on inserting a row in the table.	(4) TableName
(E)	This method removes all the data from the table.	(5) DataSet



In this module, **Data Classes**, you learnt about:

→ **DataAdapter Classes**

Data can be inserted and updated in the database using DataAdapter. ADO.NET implements various types of data adapters, which are inherited from the DbDataAdapter class. The SqlDataAdapter class is used to interact with the SQL database. The OleDbDataAdapter class is used to interact with multiple databases such as Sybase and Oracle.

→ **DataReader**

The SqlDataReader object reads data from a SQL Server database. The OleDbDataReader reads data from a data source such as Microsoft Access, Sybase, and DB2.

→ **DataSets**

DataSet objects are a collection of tables in the cached memory. They support the disconnected architecture and support multiple tables for working simultaneously. DataSet objects can either be typed or untyped. DataSet objects can display data from an XML file and can store data in an XML file in XML format.

→  **DataTables**

A DataTable is a single table in the DataSet. The DataColumn and the DataRow classes form the basic elements for creating a DataTable schema. The DataRelation objects are used to interact with tables having the parent-child relationship.

# Module - 9

## DataView and DataGridView Controls

Welcome to the Module, **DataView and DataGridView Controls**.

The DataGrid and DataGridView controls are used to fetch and display records from a table in the database. The DataView class allows sorting and filtering of data, while DataGrid control allows inserting, updating, and deleting records. The DataGridView control is used to work with different types of data sources from which the data is fetched. It is used to customize the appearance and behavior of the rows, columns, and cells.

In this Module, you will learn about:

- DataView Class
- DataGrid Control
- DataGridView Control
- DataGrid and DataGridView - Difference

## 9.1 DataView Class

In this first lesson, **DataView Class**, you will learn to:

- Explain the `DataView` class and its importance.
- Explain briefly the `DataViewManager` class.

### 9.1.1 Purpose

Consider a scenario where the accountant of an IT firm is using the Employee Payroll application. The accountant wants to know the names of employees who are drawing a salary of greater than 1000 dollars. The retrieved records must be sorted in the ascending order of the employee names. This will help the accountant to know a list of employees who are eligible to get a bonus more than 100 dollars. Therefore, the application needs to be modified to provide this functionality.

To fulfill this task, the developer should use the `DataView` class to filter and sort data of the employees.

### 9.1.2 DataView Class

The `DataView` class is used to sort, filter, search, modify, and navigate through the records of a table. This is done by taking the `DataTable` object as the parameter in the constructor of the class. The `DataTable` object is linked to the required table in the database. This technique of linking the object to the table in the database is known as data binding.

The `DataView` class can fetch and modify specific rows from the `DataTable`. Thus, it acts as a subset of the `DataTable`. Therefore, you can work with two different versions of the same table simultaneously.

The `DataView` class exists in `System.Data` namespace.

The following syntax is used to initialize the `DataView` class.

#### Syntax:

```
DataView <objectName> = new DataView(<DataTable>);
```

where,

`DataView`: Is the built-in class used to initialize the `DataView` class.

`DataTable`: Is the object of the `DataTable` class.

The following code creates an object of the `DataView` class named, `dvwPhoto`.

#### Code Snippet:

```
DataView dvwPhoto = new DataView(dtablePhoto);
```

The constructor of the `DataView` class takes an object of the `DataTable` class namely, `dtablePhoto`.

**Note** - The `DataView` class defines two more constructors, `DataView()` and `DataView(DataTable, String, String, DataViewRowState)`. The second constructor takes the `DataTable` object, the column value for filtering, the column name for sorting, and the value defined in the `DataViewRowState` enumeration. This enumeration displays different rows such as changed and original rows.

## → Properties

The properties of the `DataView` class are used to specify whether the user can perform any edit or delete operations. They are also used to retrieve the number of records in a table. Table 9.1 lists the most commonly used properties of the `DataView` class.

Property	Description
<code>AllowDelete</code>	Specifies or retrieves a value indicating whether the user can delete any data.
<code>AllowEdit</code>	Specifies or retrieves a value indicating whether the user can modify any data.
<code>AllowNew</code>	Specifies or retrieves a value indicating whether new rows can be added to the table using the <code>AddNew()</code> method.
<code>Count</code>	Retrieves a value indicating the number of records available in the <code>DataView</code> after applying the <code>RowFilter</code> and <code>RowStateFilter</code> properties.
<code>Item</code>	Retrieves a row of data from a specified table.
<code>RowFilter</code>	Specifies or retrieves the expression, which will be used to filter the rows.
<code>RowStateFilter</code>	Specifies or retrieves the row state filter used in the <code>DataView</code> . The property uses the values defined in the <code>DataViewRowState</code> enumeration.
<code>Sort</code>	Specifies or retrieves the sort columns and sort order for the <code>DataView</code> .
<code>Table</code>	Specifies or retrieves the source table of the control.

Table 9.1: `DataView` Properties

The following code demonstrates how to retrieve the total number of rows after the `RowStateFilter` property of the `DataView` class is applied.

### Code Snippet:

```
// Code to fill the DataTable with the Products table
...
DataView dvwPhoto = new DataView(dtablePhoto);
dvwPhoto.AllowDelete = true;
dvwPhoto.RowStateFilter = DataViewRowState.Unchanged;
int rows = dvwPhoto.Count;
```

The code creates an instance of the `DataView` class that refers to the table in the `DataTable` object. The `AllowDelete` property is set to `true`, which will allow any delete operations on the rows of the table. The `RowStateFilter` property is used to set the row state filter to `Unchanged` value using the `DataViewRowState` enumeration.

This means all those rows, which are not changed will be retrieved. The `Count` property is used to count the rows displayed in the control after the `RowStateFilter` property is applied.

## → Methods

The methods of `DataView` class are used to add rows, find rows, and create a new `DataTable`, which is based on rows existing in the `DataView`. Table 9.2 lists the most commonly used methods of the `DataView` class.

Method	Description
<code>AddNew</code>	Adds a new row to the <code>DataView</code> .
<code>Close</code>	Closes the <code>DataView</code> .
<code>Delete</code>	Removes a particular row at the given index.
<code>Dispose</code>	Disposes the resources, except the memory, which is used by the <code>DataView</code> object.
<code>Find</code>	Searches a row in the <code>DataView</code> using the given sort key value.
<code>Open</code>	Opens a <code>DataView</code> .
<code>ToTable</code>	Creates and returns a new <code>DataTable</code> , which is based on rows present in the current <code>DataView</code> .

Table 9.2: `DataView` Methods

The following code demonstrates how to find a row based on a value and delete the same row using the `DataView` class.

### Code Snippet:

```
DataGridView dvwPhoto = new DataGridView(dtablePhoto);
dvwPhoto.Sort = "FirstName";
int rowToDelete = dvwPhoto.Find("Tommy");
dvwPhoto.Delete(rowToDelete);
```

The code uses the `Find()` method to search for the value `Tommy`. The row index is saved in the `rowToDelete` variable. This variable is passed as a parameter to the `Delete()` method, which will delete the row.

## → Events

The events of the `DataView` class are used to handle the changes to the underlying list of data. Table 9.3 lists the most commonly used events of the `DataView` class.

Event	Description
<code>Initialized</code>	Occurs once the <code>DataView</code> has been initialized.
<code>ListChanged</code>	Occurs after any change is made to the list managed by the <code>DataView</code> .

Table 9.3: `DataView` Events

The following code demonstrates how to use the `ListChanged` event of the `DataView` class.

**Code Snippet:**

```
SqlDataAdapter sqldaCustomers = new SqlDataAdapter("SELECT * FROM Customers", sqlconCustomers);
DataTable dtableCustomers = new DataTable("Customers");
dtableCustomers.AcceptChanges();
sqldaCustomers.Fill(dtableCustomers);
DataView dvwUSACustomers = new DataView(dtableCustomers);
dvwUSACustomers.RowFilter = "Country='USA'";

dvwUSACustomers.AllowDelete = true;
dvwUSACustomers.Delete(4);
sqldaCustomers.Update(dtableCustomers);

// Code to display the records
. . .

private void dvwUSACustomers_ListChanged(object sender,
ListChangedEventArgs e)
{
    MessageBox.Show("The list has been changed.");
}
```

The code fills the `DataTable` object with the records of the `Customers` class. An instance of the `DataView` class is created, which takes the `DataTable` object parameter. The `RowFilter` property is used to display the records of customers who are from USA. The `AllowDelete` property is set to `true`, which means that the user is allowed to delete rows. The `Delete()` method is invoked to delete the fifth row from the table. This change also takes place in the table of the database by using the `Update()` method of the `SqlDataAdapter` class. When the row is deleted, the `ListChanged` event is raised. When this event is raised, a message box is displayed stating that the list is changed.

### 9.1.3 *DataViewManager Class*

The `DataViewManager` class is used to specify the settings for each table in the data set. It is useful when you want to bind a `DataSet` to more than one table and display them in the `DataView`. The `DataViewManager` class ensures that accurate and required data is retrieved from each table.

The class is used to specify different view settings for different tables in the data set. This is done by using the various properties, methods, and events defined in the class. Table 9.4 lists the most commonly used properties, methods, and events of the `DataViewManager` class.

Name	Description
DataSet	This property specifies or retrieves the name of the <code>DataSet</code> , which is to be used with the <code>DataViewManager</code> .

Name	Description
DataViewSettings	This property retrieves the DataViewSettingCollection for each DataTable in the DataSet.
CreateDataGridView	This method creates a DataView for the specified DataTable.
ListChanged	This event occurs after the addition or deletion of a row from the DataView.

Table 9.4: DataView Class Members

The following code demonstrates how to sort the records of the Products table using the DataViewManager class.

#### Code Snippet:

```
...
SqlDataAdapter sqldaProducts = new SqlDataAdapter("SELECT * FROM Products",
sqlcon);
DataSet dsetProducts = new DataSet("Products");
sqldaProducts.Fill(dsetProducts, "Products");

DataViewManager dvwmgrProducts = new DataViewManager();
dvwmgrProducts.DataSet = dsetProducts;
dvwmgrProducts.DataViewSettings["Products"].Sort = "ProductName";
// Display product details.
. . .
```

The code fills the DataSet with the records of the Products table. An object of the DataViewManager class is created namely, dvwmgrProducts. The DataSet property is used to specify the data set for the DataViewManager. The DataViewSettings property is used to sort the records according to the ProductName column.

#### Knowledge Check 1

1. You want to use a DataView to filter data from a DataTable. Which one of the following codes will help you to achieve this?

(A)	<pre>DataTable dvwOrders = new DataView(); dvwOrders.Table = dsetOrders.Tables(0); dvwOrders.RowFilter = "CustomerID = 'LABD'"; dvwOrders.RowStateFilter = DataViewRowState.CurrentRows;</pre>
-----	--

(B)	<pre>DataGridView dvwOrders = new DataGridView();  dvwOrders.Table = dsetOrders.Tables[0];  dvwOrders.RowFilter = "CustomerID = 'LABD'";  dvwOrders.RowStateFilter = DataViewRowState.CurrentRows;</pre>
(C)	<pre>DataGridView dvwOrders = new DataGridView();  dvwOrders.Table = dsetOrders.Tables[0];  dvwOrders.Filter = "CustomerID = 'LABD'";  dvwOrders.StateFilter = DataViewRowState.CurrentRows;</pre>
(D)	<pre>DataGridView dvwOrders = new DataGridView();  dvwOrders.Table = dsetOrders.Tables[0];  dvwOrders.RowFilter = "CustomerID = 'LABD'";  dvwOrders.RowStateFilter = DataViewRowState.Current;</pre>

2. Which of these statements about **DataView** and **DataViewManager** classes are false?

(A)	The <b>DataView</b> class provides view settings for multiple tables in the <b>DataSet</b> .
(B)	The <b>DataViewManager</b> class ensures that the information from each table is fetched correctly.
(C)	The <b>DataView</b> class supports searching and navigating records of a table.
(D)	The <b>DataViewManager</b> class allows you to filter and sort rows of a table.
(E)	The <b>ListChanged</b> event of the <b>DataViewManager</b> class occurs when a row is deleted from the <b>DataView</b> .

## 9.2 DataGrid Control

In this second lesson, **DataGrid Control**, you will learn to:

- Identify the use of **DataGrid** control and explain it.
- List and explain the properties, methods, and events of the **DataGrid** control.

### 9.2.1 Purpose

Consider a scenario where the HR manager of a firm is updating some employee details using the Employee form of the Payroll application. The form provides text boxes to update the employee details

such as name, age, address, and e-mail. At any given point of time, the manager can view and edit details of only a single employee. This is because the `TextBox` control can display only one record at a time.

The HR manager finds this way of updating details as time consuming and tedious. To overcome this problem, the developer is asked to implement some efficient method of updating employee information.

To do so, the developer can use the `DataGridView` control. This control has the capability of displaying the entire `Employee` table or the required records from the database in a tabular format. This makes it easy for the manager to view, edit, and update details.

### 9.2.2 *DataGrid Control*

The `DataGrid` control displays data in a tabular format. It acts as an interface for `DataSet` objects and is used to display the data of the required table from the database. The control is automatically populated by the table present in the `DataSet`, when the control is bound to the `DataSet`. It is also used to insert, update, and delete data from the table in the database. The control can display data of a single table, multiple tables, and related tables. Figure 9.1 displays the `DataGrid` control.

	OrderID	CustomerID	OrderDate	RequiredDate
▶	0101	ALPA	09/20/2007	09/30/2007
	0102	TOKY	09/21/2007	09/25/2007
	0103	PLPQ	09/22/2007	09/28/2007
	0104	TYUR	09/22/2007	10/02/2007
	0110	OMOF	09/23/2007	09/30/2007
	0100	BDHR	09/15/2007	09/20/2007
*	0120	LABD	09/15/2007	09/30/2007
*				

Figure 9.1: DataGrid Control

The following syntax is used to create a `DataGrid` object.

#### Syntax:

```
DataGrid <objName> = new DataGrid();
```

where,

`DataGrid`: Is the class used to create the `DataGrid` control.

The following code creates the `DataGrid` control, namely `dgridNewLife`.

#### Code Snippet:

```
DataGrid dgridNewLife = new DataGrid();
```

## → Properties

The `DataGridView` class is used to create the `DataGridView` control and it exists in the `System.Windows.Forms` namespace.

The class provides various properties to specify and retrieve the data source and the value within a cell of the control. Table 9.5 lists the most commonly used properties of the `DataGridView` class.

Property	Description
<code>CurrentCell</code>	Specifies or retrieves the currently focused cell.
<code>CurrentRowIndex</code>	Specifies or retrieves the index of the currently focused row.
<code>DataBindings</code>	Retrieves the data bindings for the control and is inherited from the <code>Control</code> class.
<code>DataMember</code>	Specifies or retrieves the specified list in the data source, which needs to be displayed as a table.
<code>DataSource</code>	Specifies or retrieves the data source for which the control will display data.
<code>Item</code>	Specifies or retrieves the value of a given cell.

Table 9.5: `DataGridView` Properties

## → Methods

The `DataGridView` class provides various methods used to edit the data in the table and navigate through tables. Table 9.6 lists the most commonly used methods of `DataGridView` class.

Method	Description
<code>BeginEdit</code>	Enables you to edit the data in the <code>DataGridView</code> control.
<code>Collapse</code>	Collapses child relations for the given row or for all the rows.
<code>IsExpanded</code>	Retrieves a value indicating whether the node of the given row is in expanded or collapsed state.
<code>isSelected</code>	Retrieves a value indicating whether the given row is selected.
<code>NavigateBack</code>	Navigates back to the previously displayed table in the grid.
<code>NavigateTo</code>	Navigates to the table indicated by the row and the relation name.
<code>Select</code>	Selects the given row.
<code>Unselect</code>	Unselects the given row.

Table 9.6: `DataGridView` Methods

## → Events

The `DataGridView` class provides various events used to track user generated events. Table 9.7 lists the most commonly used events of the `DataGridView` class.

Event	Description
BackButtonClick	Occurs when the back button on the child table is clicked.
Scroll	Occurs when the user scrolls the control.
ShowParentDetailsButtonClick	Occurs when the ShowParentDetails button is clicked.
TextChanged	Occurs when the value of the Text property is changed.

Table 9.7: DataGridView Events

The following code demonstrates how to fill the `DataGridView` control with the records of the `Orders` table.

#### Code Snippet:

```
...
SqlDataAdapter sqldaOrders = new SqlDataAdapter("SELECT * FROM Orders", sqlconOrders);
DataSet dsetOrders = new DataSet("Orders");
sqldaOrders.Fill(dsetOrders, "Orders");
DataGrid dgridOrders = new DataGrid();
this.Controls.Add(dgridOrders);
dgridOrders.DataSource = dsetOrders;
dgridOrders.Dock = DockStyle.Fill;
dgridOrders.DataMember = "Orders";
```

The code fills the `DataSet` with the records of the `Orders` table. An instance of the `DataGrid` class namely, `dgridOrders` is created. The `DataSource` property is set to the `DataSet` object, which will be the data source of the `DataGrid`. The `Dock` property is used to fit the control within the size of the form. The `DataMember` property is set to the table, `Orders`. Finally, the control displays the records of the `Orders` table.

## Knowledge Check 2

- Can you match the properties of the `DataGrid` class against their corresponding descriptions?

Description		Property
(A)	Represents the index of the focused row.	(1) DataMember
(B)	Represents the data source for the control.	(2) CurrentRowIndex
(C)	Indicates the list in the data source, which is to be displayed in the control.	(3) DataSource
(D)	Represents the currently focused cell.	(4) Item
(E)	Indicates a particular cell value.	(5) CurrentCell

2. Can you match the methods of the `DataGridView` class against their corresponding descriptions?

Description		Method
(A)	Navigates to the table by relation name.	(1) <code>BeginEdit</code>
(B)	Indicates the expanded or collapsed state of the row node.	(2) <code>NavigateBack</code>
(C)	Puts the control in the editable state.	(3) <code>IsExpanded</code>
(D)	Navigates back to the previous table displayed in the control.	(4) <code>NavigateTo</code>
(E)	Collapses child relations for all the rows.	(5) <code>Collapse</code>

### 9.3 DataGridView Control

In this third lesson, **DataGridView Control**, you will learn to:

- Describe the `DataGridView` control.
- List and explain the properties, methods, and events of the `DataGridView` control.

#### 9.3.1 DataGridView Control

The `DataGridView` control is a flexible control used to display data from a table in a tabular format from different types of data sources. You can define custom operations using this control. This is possible with its following features:

- Provides different column styles such as columns with `TextBox` and `CheckBox`, which can be displayed in the `DataGridView` control.
- Supports multiple data sources to work with different databases. You can use the control, even if it is not bound to any data source.
- Implements the virtual mode feature by allowing the control to use the cache memory. This feature enhances the performance while managing large number of records because of cache and enables you to perform customized operations.
- Uses the `BindingSource` component to simplify the process of data binding. This is because the component provides all the services required for data binding.

The following syntax is used to create the `DataGridView` object.

##### Syntax:

```
DataGridView <objectName> = new DataGridView();
```

where,

`DataGridView`: Is the built-in class used to create the `DataGridView` control.

The following code creates an object of `DataGridView` class named, `dgvwDepartment`.

**Code Snippet:**

```
DataGridView dgvwDepartment = new DataGridView();
```

**Note** - If the `DataGridView` control is not bound to a data source, this mode of working is referred to as the unbound mode. This state is usually used to manage small amount of data. Here, you must populate the control by using the `DataGridViewRowCollection.Add()` method.

The `DataGridView` class is new in .NET Framework 2.0. The `DataGridView` control is useful in the following scenarios:

- Displaying small amount of data using the unbound mode.
- Viewing and updating data in tables of a database using ADO.NET objects such as `DataTable`.
- Handling large amount of data using virtual mode for optimum performance.
- Customizing the rows, columns, cells, and headers to provide tool tips and shortcut menus.

→ **Properties**

The properties of the `DataGridView` class are used to specify or retrieve the table name, rows, and columns. Table 9.8 lists the most commonly used properties of the `DataGridView` class.

Property	Description
<code>AutoGenerateColumns</code>	Specifies or retrieves a value that indicates whether columns are created automatically after setting the <code>DataSource</code> or <code>DataMember</code> properties.
<code>Columns</code>	Retrieves a collection that contains all the columns of the control.
<code>DataMember</code>	Specifies or retrieves the list or table name whose data is displayed in the control.
<code>DataSource</code>	Specifies or retrieves the data source for which the data is displayed in the control.
<code>Item</code>	Specifies or retrieves the cell, which is located at the specified row and column intersection.
<code>Rows</code>	Retrieves a collection that contains all the rows of the control.
<code>SelectedColumns</code>	Retrieves the columns, which are selected by the user.
<code>SelectedRows</code>	Retrieves the rows, which are selected by the user.

**Table 9.8: DataGridView Properties**

The following code demonstrates how to add rows and columns to the `DataGridView` control using the `DataGridView` class.

**Code Snippet:**

```

DataGridView dgvwDepartment = new DataGridView();
this.Controls.Add(dgvwDepartment);
dgvwDepartment.DataSource = dsetDepartment.Tables[1];
dgvwDepartment.AutoGenerateColumns = true;
dgvwDepartment.Columns.Add("dcolName", "Name");
dgvwDepartment.Columns.Add("dcolAge", "Age");

dgvwDepartment.Rows.Add("Harry");
dgvwDepartment.Rows[0].Cells[1].Value = "20";

```

The code uses the `DataSource` property of the `DataGridView` class, which is used to specify the second table in the `DataSet` as the data source. The `AutoGenerateColumns` property is set to `true`. This will automatically generate columns in the control as the `DataSource` property is already set. The `Columns` property is used to add columns namely, `Name` and `Age`. It takes the  `DataColumn` object as the first parameter and the column name as the second parameter. The `Rows` property is used to add a row, whose first column is given the value as `Harry`. Now, the `Value` property is used to enter the value `20` in the second column of the first row.

## → Methods

The methods of the `DataGridView` class can be used to select and sort cells. They can also be used to edit and retrieve the cells that fulfill the filter condition. Table 9.9 lists the most commonly used methods of the `DataView` class.

Method	Description
<code>ClearSelection</code>	Cancels the selection of the currently selected cells.
<code>CommitEdit</code>	Makes changes in the current cell to the data cache without ending the edit mode.
<code>DisplayedColumnCount</code>	Retrieves the number of columns that are displayed to the user.
<code>DisplayedRowCount</code>	Retrieves the number of rows displayed to the user.
<code>GetCellCount</code>	Retrieves the number of cells satisfying a specified filter.
<code>NotifyCurrentCellDirty</code>	Allows the <code>DataGridView</code> to be notified about the current cell, which has not committed the changes.
<code>SelectAll</code>	Selects all the cells in the <code>DataGridView</code> .
<code>Sort</code>	Sorts the data in the <code>DataGridView</code> control.

Table 9.9: `DataView` Methods

The following code demonstrates how to sort the records of a table in the descending order by using the methods of the `DataGridView` class.

#### Code Snippet:

```
DataGridView dgvwDepartment = new DataGridView();
this.Controls.Add(dgvwDepartment);
int cellCount = dgvwDepartment.GetCellCount(DataGridViewElementStates.
Selected);
dgvwDepartment.SelectAll();
dgvwDepartment.Sort(dgvwDepartment.Columns[0], ListSortDirection.
Descending);
```

The `GetCellCount()` method is used to count the number of cells selected by the user. This is done using the `Selected` value of the `DataGridViewElementStates` enumeration. The `SelectAll()` method is used select all the cells in the control. The `Sort()` method is used to sort the data based on the values in the first column and in descending order. This is done using the `Columns` property and the `ListSortDirection` enumeration.

#### → Events

The events of the `DataGridView` control can be used to handle the changes made to the `DataMember` and `DataSource` properties. Table 9.10 lists the most commonly used events of the `DataGridView` class.

Event	Description
CellClick	Occurs when any part of a cell has been clicked.
CellValueChanged	Occurs when the value of a cell has been changed.
DataMemberChanged	Occurs when the value of the <code>DataMember</code> property is changed.
DataSourceChanged	Occurs when the value of the <code>DataSource</code> property is changed.
RowValidated	Occurs when the process of validating a row is complete.
RowValidating	Occurs during the process of validating a row.

Table 9.10: DataGridView Events

The following code demonstrates how to select the entire row when the user clicks a particular cell of the row.

**Code Snippet:**

```
DataGridView dgvwCustomers = new DataGridView();
this.Controls.Add(dgvwDepartment);
SqlDataAdapter sqldaCustomers = new SqlDataAdapter("SELECT * FROM
Customers", sqlconCustomers);
DataSet dsetCustomers = new DataSet("Customers");
sqldaCustomers.Fill(dsetCustomers, "Customers");

dgvwCustomers.Dock = DockStyle.Fill;
dgvwCustomers.DataSource = dsetCustomers;
dgvwCustomers.DataMember = "Customers";

private void dgvwCustomers_CellClick(object sender,
DataGridViewCellEventArgs e)
{
    dgvwCustomers.Rows[e.RowIndex].Selected = true;
}
```

The code creates an instance of the `DataGridView` class and fills it with the records of the `Customer` table. When the user clicks any cell within the table displayed in the control, the `CellClick` event is raised. When this event is raised, the row in which the clicked cell exists is completely selected.

This is done using the `RowIndex` property of the `DataGridViewCellEventArgs` class and the `Selected` property of the control. The `RowIndex` property retrieves the index of the row, in which the clicked cell exists.

### Knowledge Check 3

- Can you match the properties, methods, and events of the `DataGridView` class against its corresponding descriptions?

Description		Property, Method, and Event	
(A)	This property notifies the control about the cell, which has uncommitted changes.	(1)	AutoGenerateColumns
(B)	This event occurs after the value of a cell changes.	(2)	CellClick
(C)	This property indicates whether columns can be generated automatically.	(3)	NotifyCurrentCellDirty
(D)	This event occurs when the user clicks the cell.	(4)	CommitEdit
(E)	This method makes the changes in the current cell to the data cache.	(5)	CellValueChanged

2. You want to add two columns namely, CustomerID and Name in the DataGridView control. Which one of the following codes will help you to achieve this?

(A)	<pre>DataGridView dgvwCustomers = new DataGridView(); dgvwCustomers.Columns.Add("dcolID", "CustomerID"); dgvwCustomers.Columns.Add("dcolName", "Name"); this.Controls.Add(dgvwCustomers);</pre>
(B)	<pre>DataGridView dgvwCustomers = new DataGridView(); dgvwCustomers.Columns.AddNew("dcolID", "CustomerID"); dgvwCustomers.Columns.AddNew("dcolName", "Name"); this.Controls.Add(dgvwCustomers);</pre>
(C)	<pre>DataGridView dgvwCustomers = new DataGridView(); dgvwCustomers.Columns.AddColumn("dcolID", "CustomerID"); dgvwCustomers.Columns.AddColumn("dcolName", "Name"); this.Controls.Add(dgvwCustomers);</pre>
(D)	<pre>DataGridView dgvwCustomers = new DataGridView(); dgvwCustomers.Column.Add("dcolID", "CustomerID"); dgvwCustomers.Column.Add("dcolName", "Name"); this.Controls.Add(dgvwCustomers);</pre>

## 9.4 DataGrid and DataGridView Controls

In this fourth lesson, **DataGrid and DataGridView Controls**, you will learn to:

- Compare the DataGrid and DataGridView controls.

### 9.4.1 Comparison

The DataGridView control extends the functionality of the DataGrid control by providing various new features. Table 9.11 lists the differences between the two controls.

DataGridView Control	DataGrid Control
Provides more built-in column types.	Provides restricted number of column types.
Displays data from bound and unbound data source.	Displays data only from an external data source.
Supports multiple ways of displaying and formatting data. For example, you can set an appearance for the cells having similar kind of data.	Supports limited formatting of data.

DataGridView Control	DataGrid Control
Provides multiple options for changing the appearance of cells, rows, columns, and headers. For example, you can change their size; provide tool tips, and shortcut menus.	Supports limited formatting of cells, rows, columns, and headers.
Does not support hierarchical display of data from tables.	Supports hierarchical display of data from two tables, which are related.

Table 9.11: Differences

## Knowledge Check 4

1. Which of the following statements about the DataGrid and DataGridView controls are false?

(A)	The DataGridView control allows you to specify custom column types.
(B)	The DataGrid control displays data only from an external data source.
(C)	The DataGridView control allows you to set the same appearance for the cells having similar kind of data.
(D)	The DataGrid control allows you to set shortcut menus for cells.
(E)	The DataGridView control allows you to display hierarchical information of two tables.

## Module Summary

In this module, **DataView and DataGridView Controls**, you learnt about:

### → **DataView Class**

The DataView class allows you to search, sort, and filter the records of a table. The data members of the class can be used to fetch the required records from the DataTable object. The DataViewManager object is used along with the DataView class to specify settings for handling multiple tables.

### → **DataGridView Control**

The DataGridView control is used to insert, update, delete, and sort the records of a table in the database. This is done by attaching the control to the table using the method of data binding. The DataSource property of the DataGridView class binds the control to the data source.

### → **DataGridView Control**

The DataGridView control is used to display data from tables using different types of data sources. The control uses the BindingSource component to ease the process of data binding. It allows the user to implement virtual mode, which is used to maximize the user-defined operations on huge data.

### → **DataGridView and DataGridView - Difference**

The DataGridView control provides various column styles to customize the look of the control. The BindingSource component is used along with the control to enhance the process of data binding. These features are not provided by the DataGridView control. However, the DataGridView control can display data in a hierarchical manner.

# Module - 10

## Data Binding

Welcome to the Module, **Data Binding**.

Data binding is the technique used to fetch and update the records through controls. You can implement data binding to fetch and update a single record or multiple records at a time. The BindingSource component in Windows Forms is used to optimize the process of data binding. Simple binding is the process of binding a property of a control to a single value in the data source. Complex binding is the process of binding a control to more than one element in the data source.

In this Module, you will learn about:

- Data Binding
- Types of Binding
- BindingSource Component
- Sorting and Filtering

## 10.1 Data Binding

In this first lesson, **Data Binding**, you will learn to:

- Explain the process of data binding.
- Describe the terms data providers and data consumers.
- Explain `BindingContext` and `CurrencyManager` classes.

### 10.1.1 Data Binding

Consider a scenario where an application for student examination system is being created by a developer. This application is a desktop application, which will allow the user to search the percentage obtained in an exam based on the roll no and exam date. To do this, the developer must fetch the percentage from the database and display it using an appropriate control. This can be done by linking the suitable control to the appropriate field of the table in the database. In Windows Forms programming, this linking between the control and the table in the database is referred to as data binding.

Data binding is one of the most powerful features of .NET and Windows Forms. It helps in linking a control to a data source such as tables in the database. This is useful to view, insert, update, search, and delete records in the table.

Data binding allows you to create a link between the controls of the form and the database. Once the controls are linked to the database, you can retrieve data from the database. You can fetch records either from a single table or multiple tables. These records are generally fetched in the `DataSet` or `DataTable`, and controls are generally bound to these components.

There are certain advantages and disadvantages of the data binding technique. Some of the advantages of data binding are as follows:

- Data binding binds data with minimum lines of code and thus, makes the execution process faster.
- The developer has the flexibility to modify the auto-generated code on data binding to meet the requirements.

The disadvantages of data binding are as follows:

- Difficult to debug applications because of the extra binding layer that exists.
- It does not provide 100% flexibility to include customized functionalities.

### 10.1.2 Data Providers and Data Consumers

Consider an example of an electricity provider, which acts as an organization medium through which you receive electricity. Here, you are the consumer of the electricity. Similarly, in the data binding technique, a data provider is an object or component, which makes the data available to the consumers. Some of the data provider objects are arrays, `DataSet`, `DataTable`, and `DataView`.

A data consumer is an object, which uses the data made available by the provider. The role of data consumers is to display the data with the purpose of viewing, modifying, or navigating the available data. Some of the data consumers are text boxes, labels, and grid controls.

**Note** - In .NET, for using data binding on controls having more than one item, the provider should implement the `IList` interface, which represents an index-based collection of items.

### 10.1.3 BindingContext and CurrencyManager Classes

Windows Form contains at least one `BindingContext` object. This object is an instance of the `BindingContext` class, which helps in binding to multiple data sources. For any control that inherits from `Control` class, the `BindingContext` object manages a collection of `BindingManagerBase` objects. These objects are instances of the `BindingManagerBase` class, which is an abstract class. It helps in managing all objects bound to the same data source.

All data sources, which are bound to Windows Forms controls, will be linked with a currency manager. The `CurrencyManager` class is used to implement currency manager and is used to point to an item in the data source. It has a one-to-one relationship with the data source and is used to navigate through records. The class is inherited from the `BindingManagerBase` class and is used to synchronize records and refresh the record values.

**Note** - All controls that are derived from `System.Windows.Forms.Control` have the `BindingContext` property.

The `BindingContext` class defines various properties, methods, and events used to specify and retrieve items and bound objects. Table 10.1 lists the commonly used properties, and methods of `BindingContext` class.

Name	Description
<code>IsReadOnly</code>	This property retrieves a value, which indicates whether the collection is read-only.
<code>Item</code>	This property retrieves an object of the <code>BindingManagerBase</code> class.
<code>Add</code>	This method adds an object of the <code>BindingManagerBase</code> class, which is associated with the specified data source.
<code>Contains</code>	This method retrieves a value that indicates whether an object of the <code>BindingContext</code> class contains an object of the specified <code>BindingManagerBase</code> class.
<code>Remove</code>	This method removes an object of the <code>BindingManagerBase</code> class, which is associated with the specified data source.

Name	Description
UpdateBinding	This method links an object of the Binding class with a new object of the BindingContext class.

Table 10.1: BindingContext Class Members

The following code creates a TextBox control and binds this control to the SQL database table.

#### Code Snippet:

```
TextBox txtPID = new TextBox();
this.Controls.Add(txtPID);
SqlConnection sqlconProjects = new SqlConnection("Data Source=OLIVE\\SQLEXPRESS;Initial Catalog=Alpine;Integrated Security=SSPI");
SqlDataAdapter sqldaMedi = new SqlDataAdapter("SELECT * FROM Projects",
sqlconProjects);
DataSet dsetMedi = new DataSet();
sqldaMedi.Fill(dsetMedi, "Projects");
txtPID.DataBindings.Add("Text", dsetMedi, "Projects.PID");
```

The code creates a TextBox control and an instance of the SqlConnection class namely, sqlconProjects. An instance of the SqlDataAdapter class is created and it passes the SQL command and SqlConnection object as parameters. A dataset is created and the Fill() method fills the dataset with the records of the Projects table. The DataBindings property binds the PID column of the Projects table to the TextBox control.

#### → CurrencyManager Class

Consider an example where two text boxes refer to two different fields of the same table called **Students**. The CurrencyManager class ensures that the correct values are displayed in the respective text boxes by using the various properties, methods, and events.

- Properties

The properties of the CurrencyManager class allow you to count the number of items and retrieve the current item. Table 10.2 lists the commonly used properties of the CurrencyManager class.

Property	Description
Bindings	Retrieves the collection of bindings that are being managed.
Count	Retrieves the number of items in the list.
Current	Retrieves the current item in the list.
List	Retrieves the list for CurrencyManager.
Position	Specifies or retrieves the current position within the list.

Table 10.2: CurrencyManager Properties

- Methods

The methods of `CurrencyManager` class allow you to add, remove items based on the specified index position, or resume the data binding process. Table 10.3 lists the most commonly used methods of the `CurrencyManager` class.

Method	Description
AddNew	Adds a new item to the current list.
Refresh	Forces a data-bound list to be repopulated.
RemoveAt	Deletes the item at the mentioned index.
ResumeBinding	Resumes data binding.

Table 10.3: `CurrencyManager` Methods

- Events

The events of the `CurrencyManager` class allow you to handle the events, which occur when an item in the list is changed. Table 10.4 lists the most commonly used events of the `CurrencyManager` class.

Event	Description
ItemChanged	Occurs when the current item has been changed.
ListChanged	Occurs when an item in the list is changed or the list is changed.

Table 10.4: `CurrencyManager` Events

The following code creates a `CurrencyManager` object and sets its position to the first record.

**Code Snippet:**

```
TextBox txtModel = new TextBox();
this.Controls.Add(txtModel);
SqlConnection sqlconProducts = new SqlConnection("Data Source=OLIVE\\
SQLEXPRESS;Initial Catalog=AutomobileInventory;Integrated
Security=SSPI");
SqlDataAdapter sqldaProducts = new SqlDataAdapter("SELECT * FROM
ProductDetails", sqlconProducts);
DataSet dsetProducts = new DataSet();
sqldaProducts.Fill(dsetProducts, "ProductDetails");
txtModel.DataBindings.Add("Text", dsetProducts, "ProductDetails.
Model");
CurrencyManager cmgrProducts = (CurrencyManager)this.
BindingContext[dsetProducts];
```

The code creates a `TextBox` control and an instance of the `SqlConnection` class namely, `sqlconProducts`. An instance of the `SqlDataAdapter` class is created and it passes the SQL command and the `SqlConnection` object as parameters. A dataset is created and the `Fill()` method fills the dataset with the records of the `ProductDetails` table. The `DataBindings` property binds the `Model` column of the `ProductDetails` table to the `TextBox` control.

An instance of the `CurrencyManager` class namely, `cmgrProducts` is created by binding the dataset using the binding context of the form.

## Knowledge Check 1

1. Can you match the properties, methods, and events of the `CurrencyManager` class against their corresponding descriptions?

Description		Property, Method, and Event	
(A)	This method resumes data binding.	(1)	ItemChanged
(B)	This property identifies the current position in the list.	(2)	List
(C)	This event occurs when the current item in the list is changed.	(3)	Current
(D)	This property identifies the bounded list.	(4)	Position
(E)	This property identifies the current item.	(5)	ResumeBinding

2. Which of these statements about data binding are false?

(A)	Data binding allows you to fetch multiple tables simultaneously.
(B)	Examples of data providers include arrays and data tables.
(C)	Data consumers fetch the data from the database.
(D)	The <code>Item</code> property of the <code>BindingContext</code> class retrieves the reference to the <code>BindingManager</code> class.
(E)	The <code>Count</code> property of the <code>CurrencyManager</code> class counts the total number of bounded lists.

## 10.2 Types of Binding

In this second lesson, **Types of Binding**, you will learn to:

- List and explain the types of binding.
- Describe simple data binding.
- Describe complex data binding.

### 10.2.1 Types of Binding

Data binding facilitates displaying and modifying information from a data source. It helps in reducing the amount of code to be written for retrieving data. Binding helps in retrieving data from any data source as well as any structure, which contains data.

You can bind controls to the data source using two types of binding:

- Simple
- Complex

In simple binding, a property of a control is bound to a single value in the data source. In complex data binding, a control is associated with one or more data values of the data source. Complex binding can only be done on controls, which can display more than one value at a time.

### 10.2.2 Simple Data Binding

Consider a scenario where the administrator of an IT firm wants to know the password of an employee based on the login ID. The developer has provided two text boxes for login ID and password. Here, the developer has to bind the text box for password to the respective password column of the table. This way of binding a control to a single value in the table is simple data binding.

Simple data binding is the ability to bind a control to a single value in a data source. This type of binding is basically used for controls such as `TextBox` or `Label` because they only display a single value at a time. Figure 10.1 displays the Simple Data Binding.

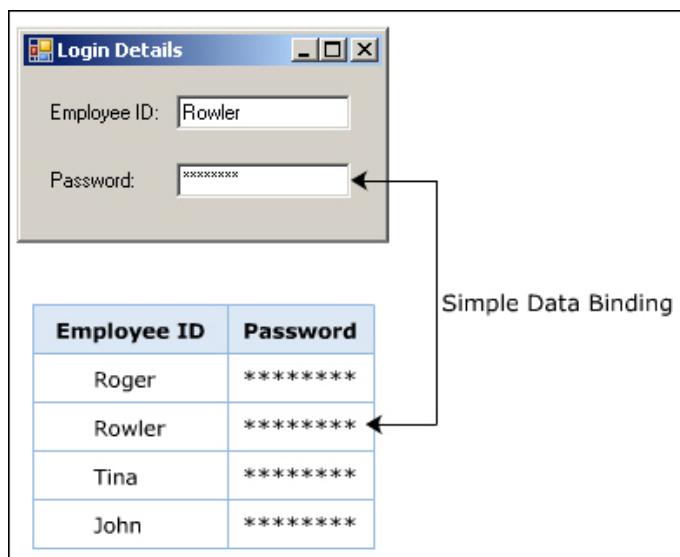


Figure 10.1: Simple Data Binding

The following code creates a data table and binds it to the `TextBox` control.

#### Code Snippet:

```
...
SqlDataAdapter sqldaLogin = new SqlDataAdapter("SELECT * FROM Login",
sqlconLogin);
DataSet dsetLogin = new DataSet();
sqldaProducts.Fill(dsetLogin, "Login");
```

```
DataTable dtableLogin = dsetLogin.Tables["Login"];
TextBox txtPassword = new TextBox();
this.Controls.Add(txtPassword);
txtPassword.DataBindings.Add("Text", dtableLogin, "Login");
```

The code creates an instance of the `SqlDataAdapter` class. The constructor takes two parameters, an SQL statement and an instance of the `SqlConnection` class, `sqlconLogin`. A dataset is created and the `Fill()` method fills the dataset with the records of the `Login` table. The `Login` table from the dataset is assigned to the `DataTable` object. The `Textbox` control is created and the `DataBindings` property binds the `Login` table to the `Textbox`.

#### → Binding Class

The `Binding` class helps to bind the property value of an object to the property value of a control. For example, you can bind the `Name` field of the `Employee` table to the `Text` property of the `TextBox` control, to retrieve the name of an employee. Table 10.5 lists the commonly used properties, methods, and events of the `Binding` class.

Name	Description
BindableComponent	This property retrieves the control to which the binding is associated.
BindingManagerBase	This property retrieves the <code>BindingManagerBase</code> for the current binding. The <code>BindingManagerBase</code> class is used to manage all binding objects.
Control	This property retrieves the control to which the binding belongs to.
DataSource	This property retrieves the data source for the current binding.
ReadValue	This method specifies the control property to the value that is read from the data source.
WriteValue	This method writes the current value to the data source after reading it from the control property.
BindingComplete	This event occurs when a binding operation is complete.

Table 10.5: Binding Class Members

The following code creates a data grid and fills the control with the table records by using the `DataSource` property.

#### Code Snippet:

```
...
DataGridView dgridTracks = new DataGridView();
TextBox txtTitle = new TextBox();
this.Controls.Add(dgridTracks);
this.Controls.Add(txtTitle);
SqlDataAdapter sqldaTracks = new SqlDataAdapter("SELECT * FROM
Tracks", sqlconTracks);
DataSet dsetTracks = new DataSet();
sqldaTracks.Fill(dsetTracks, "Tracks");
```

```
Binding bndTracks = new Binding("Text", dsetTracks, "Tracks.Title");
txtTitle.DataBindings.Add(bndTracks);
dgridTracks.DataSource = bndTracks.DataSource;
bndTracks.ReadValue();
```

The code creates a data grid, text box and an instance of the `SqlDataAdapter` class. The constructor of the `SqlDataAdapter` takes two parameters, an SQL statement and an instance of the `SqlConnection` class, `sqlconTracks`. A dataset is created and the `Fill()` method fills the dataset with the records of the `Tracks` table. An instance of the `Binding` class is created and its constructor takes three parameters. The `DataBindings` property binds the `Binding` object to a `TextBox`. This binds the `Title` column of the `Tracks` table to `Textbox`. The `DataSource` property of the grid is set to the `DataSource` property of the `Binding` object. This displays the records of the table in a grid. The `ReadValue()` method reads values from the `Binding` object and sets it to the `TextBox` control.

### 10.2.3 Complex Data Binding

The `DataGridView` control in Windows Forms supports complex data binding. It provides a powerful and flexible way to display data in a tabular format.

Complex data binding allows you to bind the control to multiple data elements. By using complex data binding, a `DataGridView` can be bound to an entire `DataSet`. Some of the controls that support complex binding are `DataGrid`, `ComboBox`, `ListBox`, and `ErrorProvider` controls. Figure 10.2 displays the Complex data binding.

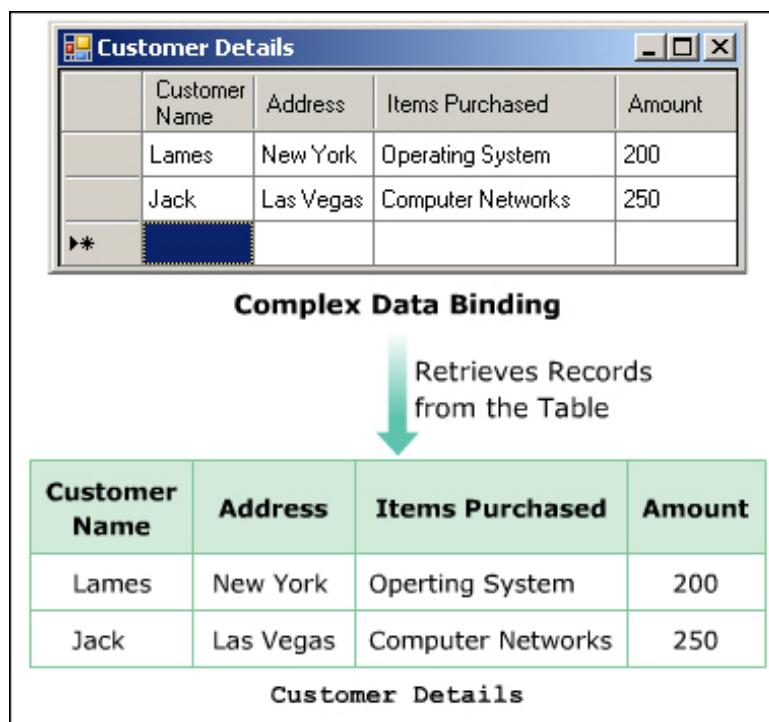


Figure 10.2: Complex Data Binding

The following code creates a data grid and fills the control with the records of the table by using the `DataMember` property.

**Code Snippet:**

```
SqlConnection sqlconBirdFly = new SqlConnection("Data Source=OLIVE\\SQLEXPRESS;Initial Catalog=BirdFly;Integrated Security=SSPI");
DataGridView dgridFly = new DataGridView();
this.Controls.Add(dgridFly);
SqlDataAdapter sqldaFly = new SqlDataAdapter("SELECT * FROM Pilot",
sqlconBirdFly);
DataSet dsetFly = new DataSet();
sqldaFly.Fill(dsetFly, "Pilot");
dgridFly.DataSource = dsetFly;
dgridFly.DataMember = "Pilot";
```

The code creates an SQL connection, `DataGridView` control, and an instance of the `SqlDataAdapter` class. A dataset is created and the `Fill()` method fills the dataset with the records of the `Pilot` table. The `DataSource` property of the grid is set to the `DataSet` object. The `DataMember` property of the grid is set to the `Pilot` table. This displays the records of the `Pilot` table in a grid.

**Note** - Complex data binding is not supported by all controls as not all controls will be able to display multi-item data.

## Knowledge Check 2

1. Which of the following statements about data binding are false?

(A)	Data binding helps in retrieving and updating records of a table.
(B)	Simple data binding supports binding to multiple records.
(C)	Complex data binding supports binding to a single element.
(D)	The <code>TextBox</code> control allows only simple data binding.
(E)	The <code>ListBox</code> control supports complex data binding.

2. You want to bind the label to the `Designation` column of the `EmployeeDetails` table. Which one of the following codes will help you to achieve this?

(A)	<pre>Label lblDesignation = new Label(); SqlConnection sqlconFastCarrier = new SqlConnection("Data Source=OLIVE\\SQLEXPRESS;Initial Catalog=FastCarrier;Integrated Security=SSPI"); SqlDataAdapter sqldaFastCarrier = new SqlDataAdapter("SELECT * FROM EmployeeDetails", sqlconFastCarrier); DataSet dsetFastCarrier = new DataSet(); sqldaFastCarrier.Fill(dsetFastCarrier, "EmployeeDetails"); lblDesignation.DataBindings.Add("Text", dsetFastCarrier, "EmployeeDetails.Designation");</pre>
-----	--

(B)	<pre>Label lblDesignation = new Label(); SqlConnection sqlconFastCarrier = new SqlConnection("Data Source=OLIVE\\SQLEXPRESS;Initial Catalog=FastCarrier;Integrated Security=SSPI"); SqlDataAdapter sqldaFastCarrier = new SqlDataAdapter("SELECT * FROM EmployeeDetails", sqlconFastCarrier); DataSet dsetFastCarrier = new DataSet(EmployeeDetails); sqldaFastCarrier.Fill(dsetFastCarrier); lblDesignation.DataBindings.Add("Text", dsetFastCarrier, "EmployeeDetails.Designation");</pre>
(C)	<pre>Label lblDesignation = new Label(); SqlConnection sqlconFastCarrier = new SqlConnection("Data Source=OLIVE\\SQLEXPRESS;Initial Catalog=FastCarrier;Integrated Security=SSPI"); SqlDataAdapter sqldaFastCarrier = new SqlDataAdapter("SELECT * FROM EmployeeDetails"); DataSet dsetFastCarrier = new DataSet(); sqldaFastCarrier.Fill(dsetFastCarrier, "EmployeeDetails"); lblDesignation.DataBindings.Add("Text", dsetFastCarrier, "EmployeeDetails.Designation");</pre>
(D)	<pre>Label lblDesignation = new Label(); SqlConnection sqlconFastCarrier = new SqlConnection("Data Source=OLIVE\\SQLEXPRESS;Initial Catalog=FastCarrier;Integrated Security=SSPI"); SqlDataAdapter sqldaFastCarrier = new SqlDataAdapter("SELECT * FROM EmployeeDetails", sqlconFastCarrier); DataSet dsetFastCarrier = new DataSet(); sqldaFastCarrier.Fill(dsetFastCarrier, "EmployeeDetails"); lblDesignation.DataBindings.Add(dsetFastCarrier, "EmployeeDetails. Designation");</pre>

### 10.3 BindingSource Component

In this third lesson, **BindingSource Component**, you will learn to:

- Explain **BindingSource component** and its use.
- List and explain the properties and methods of **BindingSource class**.
- List the steps of binding a data source at design time.

### 10.3.1 BindingSource Component

The `BindingSource` component is a new component introduced in Visual Studio 2005 to simplify the process of data binding. It acts as a data source for controls by adding data to it and finally updating the database. It also acts as a medium to provide binding and currency management functions.

It is not mandatory to bind a component to the data source before adding records to the component. The component can be used to implement both simple and complex data binding. The various uses of the component are as follows:

- It acts as a strongly typed data source.
- It helps in binding the controls and data indirectly.
- It supports data navigation, data manipulation, and sorting and filtering of data.

**Note** - Binding the controls with data indirectly is achieved by first binding the `BindingSource` component to the data source and then, binding the controls to the `BindingSource` component.

#### → Properties

The `BindingSource` class is used to implement the `BindingSource` component. The class is new in .NET Framework 2.0 and it exists in the `System.Windows.Forms` namespace. The class defines various properties used to identify the data source and position of an item in the list. Table 10.6 lists the commonly used properties of the `BindingSource` class.

Property	Description
Count	Retrieves the total number of items present in the list.
CurrencyManager	Retrieves the currency manager that is linked with the component.
Current	Retrieves the current item in the list.
DataMember	Specifies or retrieves a particular list in the data source to which the component is bound.
DataSource	Specifies or retrieves the data source to which the component is bound.
Item	Specifies or retrieves the list element at a particular index.
List	Retrieves the list, to which the component is bound.
Position	Specifies or retrieves the index of the current item in the list.

Table 10.6: `BindingSource` Properties

#### → Methods

The `BindingSource` class defines various methods that allow you to add, remove, or move items contained in the list.

Table 10.7 lists the commonly used methods of the `BindingSource` class.

Method	Description
<code>Add</code>	Adds an existing item to the internal list.
<code>AddNew</code>	Adds a new item to the list.
<code>Clear</code>	Removes all the items present in the list.
<code>Insert</code>	Inserts an item into the list at a given index.
<code>MoveFirst</code>	Moves to the first item in the list.
<code>MoveLast</code>	Moves to the last item in the list.
<code>MoveNext</code>	Moves to the next item in the list.
<code>MovePrevious</code>	Moves to the previous item in the list.
<code>RemoveAt</code>	Removes an item from the specified index position.

Table 10.7: `BindingSource` Methods

## → Events

The `BindingSource` class provides events that are generated when any item in the list is modified.

Table 10.8 lists the commonly used events of the `BindingSource` class.

Event	Description
<code>BindingComplete</code>	Occurs when all the clients are bound to the component.
<code>CurrentChanged</code>	Occurs when the currently bound item is changed.
<code>DataMemberChanged</code>	Occurs when the value of the <code>DataMember</code> property is changed.
<code>DataSourceChanged</code>	Occurs when the value of <code>DataSource</code> property is changed.
<code>ListChanged</code>	Occurs when the list or the items within the list changes.

Table 10.8: `BindingSource` Events

The following code creates a `Binding` object and checks that the cursor is present in the first record.

### Code Snippet:

```
//Code to write database connection and data adapter
.....
.....
ListBox lstBooks = new ListBox();
this.Controls.Add(lstBooks);
BindingSource bsrcBookStore = new BindingSource();
bsrcBookStore.DataSource = dsetBookStore.Tables["BookMaster"];
lstBooks.DataSource = bsrcBookStore;
lstBooks.DisplayMember = "Title";
if (bsrcBookStore.CurrencyManager.Position != 0)
{
    bsrcBookStore.MoveFirst();
}
```

The code creates a list box control and an instance of the `BindingSource` class. The `DataSource` property of the `Binding` object is set to the `BookMaster` table present in the `DataSet` object, `dsetBookStore`. The `DataSource` property of list box is set to `BindingSource` object. The `DisplayMember` property of the list box is set to `Title`. This indicates that the list box will contain records from the `Title` column. If the position of the cursor is not on the first record, then, the `MoveFirst()` method sets the cursor position to the first record.

### 10.3.2 Binding a Data Source

You can bind a control to a data source at design time or at run time. In the design time, the **Properties** window is used to bind the control. The steps to bind the control at design time are given in the image.

### Knowledge Check 3

1. Which of the following statements about the `BindingSource` component are false?

(A)	The <code>BindingSource</code> component allows adding records without first linking the component to the data source.
(B)	The <code>Current</code> property indicates the current position of an item.
(C)	The <code>DataMember</code> property specifies a list in the data source to which the component is bound.
(D)	The <code>CurrentChanged</code> event invokes the <code>BindingComplete</code> event.
(E)	The <code>DataBindings</code> property allows binding the control to a data source.

2. You want to display a combo box that displays the marks of the table by using the `BindingSource` class. Which one of the following codes will help you to achieve this?

(A)	<pre>SqlConnection sqlconHolyAngel = new SqlConnection("Data Source=OLIVE\\SQLEXPRESS;Initial Catalog=HolyAngelSchool;Integrat ed Security=SSPI"); SqlDataAdapter sqldaHolyAngel = new SqlDataAdapter("SELECT * FROM StudentDetails", sqlconHolyAngel); DataSet dsetHolyAngel = new DataSet(); sqldaHolyAngel.Fill(dsetHolyAngel, "StudentDetails"); ComboBox cboMarks = new ComboBox(); BindingSource bsrcHolyAngel = new BindingSource(); bsrcHolyAngel.DataSource = dsetHolyAngel.Tables{"StudentDetails"}; cboMarks.DataSource = bsrcHolyAngel; cboMarks.DisplayMember = "Marks";</pre>
-----	---

(B)	<pre>SqlConnection sqlconHolyAngel = new SqlConnection("Data Source=OLIVE\\SQLEXPRESS;Initial Catalog=HolyAngelSchool;Integrated Security=SSPI"); SqlDataAdapter sqldaHolyAngel = new SqlDataAdapter("SELECT * FROM StudentDetails", sqlconHolyAngel); DataSet dsetHolyAngel = new DataSet(); sqldaHolyAngel.Fill(dsetHolyAngel, "StudentDetails"); ComboBox cboMarks = new ComboBox(); BindingSource bsrcHolyAngel = new BindingSource(); bsrcHolyAngel.DataSource = dsetHolyAngel.Tables("StudentDetails"); cboMarks.DataSource = bsrcHolyAngel; cboMarks.DisplayMember = "Marks";</pre>
(C)	<pre>SqlConnection sqlconHolyAngel = new SqlConnection("Data Source=OLIVE\\SQLEXPRESS;Initial Catalog=HolyAngelSchool;Integrated Security=SSPI"); SqlDataAdapter sqldaHolyAngel = new SqlDataAdapter("SELECT * FROM StudentDetails", sqlconHolyAngel); DataSet dsetHolyAngel = new DataSet(); sqldaHolyAngel.Fill(dsetHolyAngel, "StudentDetails"); ComboBox cboMarks = new ComboBox(); BindingSource bsrcHolyAngel = new BindingSource(); bsrcHolyAngel.DataSource = dsetHolyAngel.Tables["StudentDetails"]; cboMarks.DataSource = bsrcHolyAngel; cboMarks.DisplayMember = "Marks";</pre>
(D)	<pre>SqlConnection sqlconHolyAngel = new SqlConnection("Data Source=OLIVE\\SQLEXPRESS;Initial Catalog=HolyAngelSchool;Integrated Security=SSPI"); SqlDataAdapter sqldaHolyAngel = new SqlDataAdapter("SELECT * FROM StudentDetails",); DataSet dsetHolyAngel = new DataSet(); sqldaHolyAngel.Fill(dsetHolyAngel, "StudentDetails"); ComboBox cboMarks = new ComboBox(); BindingSource bsrcHolyAngel = new BindingSource(); bsrcHolyAngel.DataSource = dsetHolyAngel.Tables("StudentDetails"); cboMarks.DataSource = bsrcHolyAngel; cboMarks.DisplayMember = "Marks";</pre>

## 10.4 Sorting and Filtering

In this last lesson, **Sorting and Filtering**, you will learn to:

- Explain briefly the **Sort** property of the **BindingSource** class.
- Explain briefly the **Filter** property of the **BindingSource** class.

### 10.4.1 Sort Property

The `Sort` property of `BindingSource` class is used to retrieve or specify the names of the column used for sorting, and also, the order in which data is sorted. This property is new in .NET Framework 2.0. The `Sort` property is a case-sensitive string that specifies the names of the columns used to sort the rows and also specifies the direction of sorting. To support sorting, a list must implement the `IBindingList` or `IBindingListView` interfaces.

By default, the columns are sorted in ascending order. You can sort multiple columns by separating them by commas, such as 'Name, Age DESC'.

The following code uses the `Sort` property to sort the records of the table.

**Code Snippet:**

```
.....
SqlDataAdapter sqldaNorthwind =
new SqlDataAdapter("Select * from Customers", sqlconNorthwind);
DataTable dtableCustomer= new DataTable();
sqldaNorthwind.Fill(dtableCustomer);
BindingSource bsrcCustomer = new BindingSource();
bsrcCustomer.DataSource = dtableCustomer;
bsrcCustomer.Sort = "Country DESC";
DataGridView dgvwCustomer = new DataGridView();
this.Controls.Add(dgvwCustomer);
dgvwCustomer.DataSource = bsrcCustomer;
```

The code creates an instance of the `SqlDataAdapter` class, and the `DataTable` class. The `Fill()` method fills the `DataTable` object with the records of the `Customers` table. The `DataSource` property of the `BindingSource` object is set to the `DataTable` object. The `Sort` property sorts the `Country` column of the `Customer` table in descending order. The `DataGridView` control is created and the `DataSource` property of this control is set to the `BindingSource` object. This displays the records of the `BindingSource` object in a grid view.

### 10.4.2 Filter Property

The `Filter` property of the `BindingSource` class is used to specify or retrieve the expression, which is used to filter the records. This property is new in .NET Framework version 2.0. The `Filter` property is mostly used in complex data binding. It is only supported by the list that implements the `IBindingListView` interface.

When the `Filter` property is not a null reference, the `BindingSource` passes the property to the list. The value of the `Filter` property will remain intact even if the data source changes.

Figure 10.3 displays the Filter property.

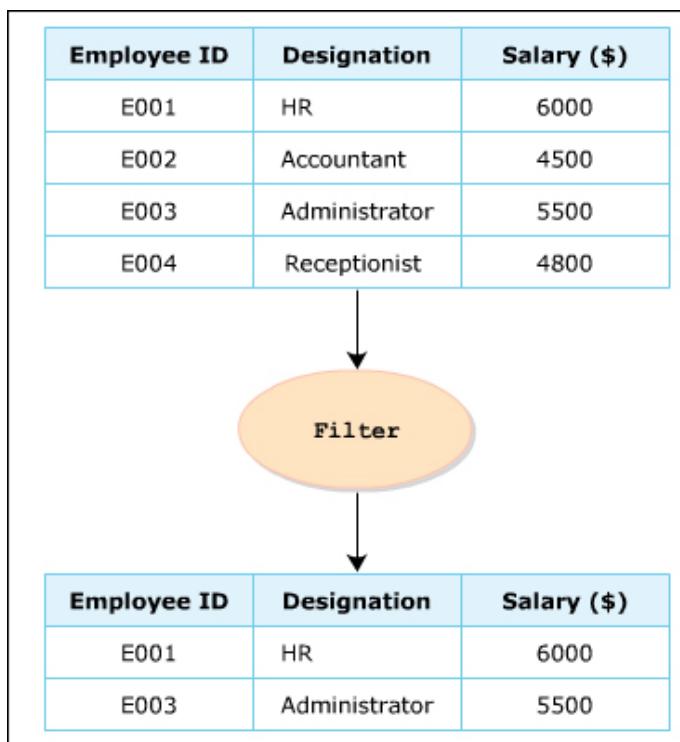


Figure 10.3: Filter Property

You can create a filter value by specifying the column name followed by an operator, and then, the value to be filtered. This value should be inside quotation marks such as 'Address = California'.

The following code uses the `Filter` property to filter the records of the table.

#### Code Snippet:

```
.....  
SqlDataAdapter sqldaNorthwind =  
new SqlDataAdapter("Select * from Customers", sqlconNorthwind);  
DataTable dtableCustomer= new DataTable();  
sqldaNorthwind.Fill(dtableCustomer);  
BindingSource bsrcCustomer = new BindingSource();  
bsrcCustomer.DataSource = dtableCustomer;  
bsrcCustomer.Sort = "Country DESC";  
DataGridView dgvwCustomer = new DataGridView();  
this.Controls.Add(dgvwCustomer);  
dgvwCustomer.DataSource = bsrcCustomer;
```

The code creates an instance of the `SqlDataAdapter` class, and the `DataTable` class. The `Fill()` method fills the `DataTable` object with the records of the `Customers` table. The `DataSource` property of the `BindingSource` object is set to the `DataTable` object. The `Filter` property retrieves the record, which contains the value `USA` in the `Country` column. The `DataGridView` control is created and the `DataSource` property of this control is set to the `BindingSource` object. This displays the filtered records of the `BindingSource` object in a grid view.

## Knowledge Check 4

1. You want to display the models of cars in ascending order that have the price greater than 450000. Which one of the following codes will help you to achieve this?

(A)	<pre>DataGridView dgvwModel = new DataGridView(); BindingSource bsrcUnitedMotors = new BindingSource(); bsrcUnitedMotors.DataSource = dsetUnitedMotors.Tables("CarModel"); bsrcUnitedMotors.Sort = "Model ASC"; bsrcUnitedMotors.Filter = "Price &gt; 45000"; dgvwModel = bsrcUnitedMotors;</pre>
(B)	<pre>DataGridView dgvwModel = new DataGridView(); BindingSource bsrcUnitedMotors = new BindingSource(); bsrcUnitedMotors.DataSource = dsetUnitedMotors.Tables("CarModel"); bsrcUnitedMotors.Sort(Model ASC); bsrcUnitedMotors.Filter(Price &gt; 45000); bsrcUnitedMotors = dgvwModel.DataSource;</pre>
(C)	<pre>DataGridView dgvwModel = new DataGridView(); BindingSource bsrcUnitedMotors = new BindingSource(); bsrcUnitedMotors.DataSource = dsetUnitedMotors.Tables("CarModel"); bsrcUnitedMotors.Sort[Model ASC]; bsrcUnitedMotors.Filter[Price &gt; 45000]; dgvwModel.DataSource = bsrcUnitedMotors;</pre>
(D)	<pre>DataGridView dgvwModel = new DataGridView(); BindingSource bsrcUnitedMotors = new BindingSource(); bsrcUnitedMotors.DataSource = dsetUnitedMotors.Tables["CarModel"]; bsrcUnitedMotors.Sort = "Model ASC"; bsrcUnitedMotors.Filter = "Price &gt; 45000"; dgvwModel.DataSource = bsrcUnitedMotors;</pre>

## Module Summary

In this module, **Data Binding**, you learnt about:

### → **Data Binding**

Data binding is used to link a control to the table in the database. It facilitates inserting, updating, and deleting records in a table of the database through the controls on the form. The `BindingContext` and the `CurrencyManager` classes are used to manage binding objects.

### → **Types of Binding**

There are two types of binding, simple data binding and complex data binding. Simple data binding allows you to bind a control to a single value in a data source. Complex data binding allows you to bind a control to multiple data elements in a data source. The `Binding` class is used to implement data binding.

### → **BindingSource Component**

The `BindingSource` component is a new component introduced to optimize the data management tasks implemented through data binding. The `BindingSource` class is used to implement the component. This class allows you to work with the items within the list. You can bind a control to the data source either at design or at run time.

### → **Sorting and Filtering**

The `BindingSource` class provides the `Sort` property that allows you to specify the order in which the data will be sorted. The `Filter` property of the `BindingSource` class filters the records based on the filter expression. A filter value can be created by specifying the column name followed by an operator and then, the value to be filtered.



are just a  
*click away*

To chat with a

**tutor**

**Login to [www.onlinevarsity.com](http://www.onlinevarsity.com)**

# Module - 11

## Working with GDI+

Welcome to the Module, **Working with GDI+**.

Windows Forms allows you to include graphics to make your application more attractive. GDI+ is a technology used in Windows Forms for drawing and displaying images. In GDI+, the Graphics class is used to draw different shapes. If your applications contain complex painting operations, the flicker effect can be reduced by using double-buffered graphics. You can either manage or render the double-buffered graphics manually or use the in-built support provided by .NET Framework.

In this Module, you will learn about:

- Introduction to Graphics with GDI+
- The Graphics Class
- Introduction to Graphics Objects
- Understanding the Font Class
- Working with Advanced GDI+

## 11.1 Introduction to Graphics with GDI+

In this first lesson, **Introduction to Graphics with GDI+**, you will learn to:

- Explain the role of graphics in Windows Forms application development and define GDI+.
- Describe device contexts and Graphics object.

### 11.1.1 Role of Graphics in Windows Forms

Windows Forms allows you to include forms and controls to your application to create a more effective and attractive user interface. Sometimes you might feel the need for graphics in your application. For example, when you want to develop a game using Windows Forms, the game must consist of various lines or arcs.

Graphics allow you to paint the user interface, or render colors, images, and objects. You can also use graphics for designing software models, blueprint of a building, and displaying the different shopping items.

Windows Forms allows you to implement graphics by using GDI+, which is an advanced version of Graphics Device Interface (GDI).

#### → Advantages of GDI+

Microsoft Windows GDI+ is a graphical device interface that allows you to create, edit, and render graphics on a form. GDI+ allows you to draw text, shapes, and curves using various built-in classes. It is widely used in creating simple to complex images. Some of the advantages of GDI+ are as follows:

- GDI+ allows you to create and render graphics quickly and efficiently.
- GDI+ allows you to display information on a computer screen or printer, irrespective of the details of a particular display device.

#### → GDI+ Classes

GDI+ provides a basic functionality for including graphics in the Windows Forms application. GDI+ exists in `System.Drawing.dll` assembly. Some of the namespaces applicable for GDI+ classes are as follows:

- `System.Drawing`
- `System.Drawing.Drawing2D`
- `System.Drawing.Printing`
- `System.Drawing.Text`

These namespaces provide classes that allow you to draw shapes and fill them with colorful and attractive text. Using GDI+, you can add images without using picture controls.

## → Device Context

The device context is a data structure that contains information about different graphic objects such as pen, brush, and so on. The device context allows you to draw images by defining various attributes of the graphics that are displayed on the output device such as a display or a printer. Thus, device contexts allow you to draw graphics irrespective of the output device.

### 11.1.2 Different Graphics Object

You can create graphical images using different Graphics objects. The different graphic objects include a pen, brush, palette, bitmap, region, and path. A pen is used to draw lines, a brush for painting and filling images, palette is used for defining a set of available colors. You can copy and scroll parts of the display screen using a bitmap, a region allows you to perform various operations such as clipping. Similarly, a path allows you to perform painting and drawing operations. Figure 11.1 displays the Graphics object.



Figure 11.1: Graphics Object

The system automatically stores the default graphics objects when an application creates a device context.

### Knowledge Check 1

- Which of the following statements about GDI+ and device context are true?

(A)	GDI+ is used to only draw images and does not allow text insertion.
(B)	GDI+ is used directly on Web Forms.
(C)	The device context is used to supply information about different graphic objects.
(D)	A device context is used to draw graphics by considering the display device.
(E)	Graphics objects are used to perform clipping operations.

## 11.2 Graphics Class

In this second lesson, The **Graphics Class**, you will learn to:

- List the steps to create a Graphics object.
- List and explain the methods of `Graphics` class.
- State the working of Graphics objects.

### 11.2.1 Graphics Object

The `Graphics` object allows you to draw various graphics in the output device. The `Graphics` object is associated with the drawing surface of the control. These objects belong to the `Graphics` class, which is a sealed class. The class provides all the functionalities to enable GDI+ drawing, editing, and rendering. You can create a `Graphics` object in three different ways. These ways are as follows:

#### → Using the Paint Event

You can create a `Graphics` object by using the `Paint` event of the `PaintEventArgs` class to retrieve the reference to the `Graphics` object.

The following code uses the `Paint` event of the `frmGraphics` form to create an object of the `Graphics` class.

#### Code Snippet:

```
private void frmGraphics_Paint(object sender, PaintEventArgs e)
{
    Graphics graphics = e.Graphics;
    graphics.FillRectangle(Brushes.Blue, ClientRectangle);
}
```

#### → Using the CreateGraphics() Method

You can create a `Graphics` object by using the `CreateGraphics()` method. This `CreateGraphics()` method belongs to the `Control` class.

The following code uses the `CreateGraphics()` method of the `Control` class to create an object of the `Graphics` class.

#### Code Snippet:

```
Graphics graphics = this.CreateGraphics();
graphics.DrawString("Welcome", this.Font, Brushes.Black, new
PointF(20, 20));
```

## → Using an Object that is Derived from the Image Class

You can create a `Graphics` object by using any instance that inherits the `Image` class. For this purpose, the `FromImage()` method of the `Graphics` class is used.

The following code uses the `Image` class to create an object of the `Graphics` class.

### Code Snippet:

```
Image imgMD = Image.FromFile("MD.jpg");
Graphics graphics = Graphics.FromImage(imgMD);
graphics.FillRectangle(Brushes.Green, 10, 10, 50, 50);
```

The code retrieves the specified .jpg image file by invoking the `FromFile()` method. The method takes the image file name as the parameter and returns a reference to the `Image` class. The `Image` class reference is passed to the `FromImage()` method, which returns the reference to the `Graphics` class.

## → Properties

The `Graphics` class encapsulates the graphic device interface. This class is associated with a particular device context. Table 11.1 lists the most commonly used properties of `Graphics` class.

Property	Description
<code>Clip</code>	Specifies or retrieves a region that limits the drawing surface.
<code>DpiX</code>	Retrieves the horizontal resolution of the current <code>Graphics</code> .
<code>DpiY</code>	Retrieves the vertical resolution of the current <code>Graphics</code> .
<code>IsClipEmpty</code>	Retrieves a value that indicates whether the clipping area is empty.
<code>IsVisibleClipEmpty</code>	Retrieves a value that indicates whether the visible clipping area is empty.
<code>PageUnit</code>	Specifies or retrieves the unit measure used for page coordinates.

Table 11.1: Graphic Properties

The following code draws a string text at the specified location.

### Code Snippet:

```
Graphics graphics = this.CreateGraphics();
graphics.Clip=new Region(new Rectangle(0,0,200,200));
graphics.PageUnit = GraphicsUnit.Pixel;
graphics.DrawString("Welcome to GDI+ Programming", this.Font, Brushes.
DarkMagenta, new PointF(15, 15));
```

The code creates an object of `Graphics` class by invoking the `CreateGraphics()` method. The `Clip` property is used to set the region of the drawing surface by invoking the constructor of the `Rectangle` class. The constructor takes four parameters namely, the x coordinate, the y coordinate, width, and height of the rectangular region. The `PageUnit` property is used to get the page coordinates using the `Pixel` property of the `GraphicsUnit` class. The `DrawString()` method draws the text `Welcome to GDI+ Programming` at the position 15, 15 using the brush

DarkMagenta.

## → Methods

The `Graphics` class methods allow you to draw graphics on the display screen. Table 11.2 lists the most commonly used methods of the `Graphics` class.

Method	Description
Clear	Clears the entire drawing surface and fills it with the specified background color.
DrawArc	Draws an arc that represents a portion of an ellipse.
DrawBezier	Draws a Bezier spline that is defined by four <code>Point</code> structures. The <code>Point</code> structure represents a pair of x and y co-ordinates of a point in a two-dimensional plane.
DrawLine	Draws a line that connects the specified two points.
DrawRectangle	Draws a rectangle with the specified height and width.
DrawString	Draws the specified text string at the specified location.
FillRectangle	Fills the interior part of the rectangle.
FillRegion	Fills the interior part of the region.
FromImage	Creates a new <code>Graphics</code> from the specified image.

Table 11.2: Graphics Methods

The following code clears the drawing area and draws an arc and a filled rectangle.

### Code Snippet:

```
Graphics graphics = this.CreateGraphics();
graphics.Clear(Color.White);
graphics.DrawArc(new Pen(Color.Black), 10, 10, 200, 200, 45, 60);
graphics.FillRectangle(new SolidBrush(Color.Black), 25, 30, 100, 100);
```

The code uses the `Clear()` method to clear the drawing area by setting the background color of the display area as White. The `DrawArc()` method draws an arc in black color using the `Pen` instance. It takes in the coordinates, width, height, the start angle position, and the sweep angle position as parameters. The `FillRectangle()` method is used to fill the shape with the black color using a solid brush. It takes in the coordinates, width, and height as the parameters to fill the particular region with the black color.

**Note** - Spline is a curve created by connecting two or more points. It also represents the mathematical equation to produce a curve.

### 11.2.2 Working with Graphics Objects

The different methods of the `Graphics` class are used for drawing graphics. Every method of this class can be represented in several forms.

One of them is listed as follows:

→ **DrawRectangle(Pen, Rectangle)**

The `DrawRectangle(Pen, Rectangle)` method allows you to draw a rectangle by specifying its basic elements such as width, height, and upper-left corner using the argument `Rectangle`. The argument `Pen` specifies the style and color of the rectangle.

→ **DrawEllipse(Pen, Rectangle)**

The `DrawEllipse(Pen, Rectangle)` method allows you to draw an ellipse by specifying its boundaries using the argument `Rectangle`. The argument `Pen` specifies the style and color of the ellipse.

→ **DrawLine(Pen, Point, Point)**

The `DrawLine(Pen, Point, Point)` method allows you to draw a line by specifying the two points to connect a line. The argument `Pen` specifies the style and color of the line. The following code draws a rectangle with the specified dimensions.

**Code Snippet:**

```
Graphics graphics = this.CreateGraphics();
Pen pen = new Pen(Color.Blue);
graphics.DrawRectangle(pen, 30, 30, 100, 100);
```

The following code draws an ellipse with the specified dimensions.

**Code Snippet:**

```
Graphics graphics = this.CreateGraphics();
Pen pen = new Pen(Color.Blue);
graphics.DrawEllipse(pen, 30, 150, 100, 50);
```

The following code draws a line at the specified location.

**Code Snippet:**

```
Graphics graphics = this.CreateGraphics();
Pen pen = new Pen(Color.Blue);
graphics.DrawLine(pen, 150, 50, 220, 150);
```

## Knowledge Check 2

- Which of the following statements about Graphics objects are true?

(A)	The Graphics object is created by obtaining a reference to a graphic object from the PaintEventArgs method.
(B)	The Graphics object is created by invoking the CreateGraphics() method.

(C)	The Graphics object is created by using an instance that inherits from the <code>Image</code> class.
(D)	The <code>DrawRectangle(Pen, Point, Point)</code> method is used to draw a rectangle.
(E)	The <code>DrawEllipse(Pen, Rectangle)</code> method is used to draw an ellipse by specifying its boundaries using the argument, <code>Rectangle</code> .

2. Can you match the properties and methods of `Graphics` class against their corresponding descriptions?

Description		Property and Method
(A)	This method creates a <code>Graphics</code> object from the specified image.	(1) <code>DrawCurve</code>
(B)	This method draws an arc that represents a portion of an ellipse.	(2) <code>PageUnit</code>
(C)	This method draws a cardinal spline that is defined by an array of points.	(3) <code>Clip</code>
(D)	This property specifies or retrieves the page coordinates.	(4) <code>FromImage</code>
(E)	This property specifies or retrieves a region that limits the drawing surface.	(5) <code>DrawArc</code>

### 11.3 Introduction to Graphics Objects

In this third lesson, **Introduction to Graphics Objects**, you will learn to:

- Explain the `Color` structure and its role in GDI+.
- Describe the `Pen` class.
- Describe the `Brush` class and its derived classes.

#### 11.3.1 Color Structure

The `Color` structure is a data structure provided by the GDI+ library to provide various colors. It is one of the basic structures for improving the visual display of an image by providing different colors.

The structure defines the alpha, red, green, and blue colors, which are the four primary components of the structure. You can specify different colors using these color combinations. Different colors can be created by a specific mixture of the red, green, and blue colors. GDI+ has added the alpha component to increase the complexity and transparency of the colors.

The `Color` structure acts as a parameter while creating brushes and pens. The brushes and pens can be made semi-transparent by using the alpha component.

**Note** - The `Color` structure includes a collection of shared as well as static properties that can be used to create additional colors. Thus, there is no need for searching for color definition.

Figure 11.2 displays the Color structure.

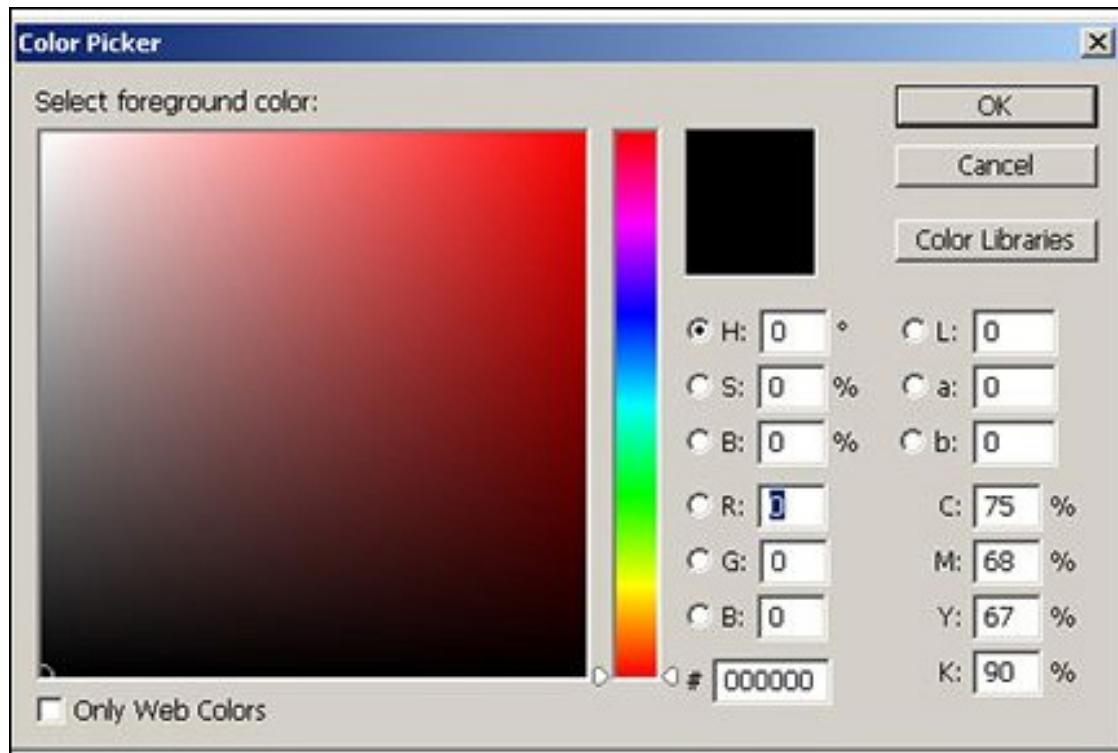


Figure 11.2: Color Structure

#### → Properties

The `Color` structure exists in `System.Drawing` namespace. The properties of the `Color` structure allow you to retrieve various system-defined colors. Table 11.3 lists the most commonly used properties of `Color` structure.

Property	Description
A	Retrieves the alpha component value that specifies the transparency of the color.
B	Retrieves the blue component value.
Blue	Retrieves the system-defined blue color.
G	Retrieves the green component value.
Green	Retrieves the system-defined green color.
Name	Retrieves the name of the currently used color.
R	Retrieves the red component value.
Red	Retrieves the system-defined red color.

Table 11.3: Color Properties

The following code creates a pen and draws two lines with two different colors.

#### Code Snippet:

```
Graphics graphics = this.CreateGraphics();
Color color = Color.DarkRed;
Pen pen = new Pen(Color.FromArgb(color.R, 50, 50));
graphics.DrawLine(pen, 20, 20, 100, 100);
pen = new Pen(Color.Blue);
graphics.DrawLine(pen, 120, 50, 100, 100);
```

The code creates an instance of the `Graphics` class. It creates an instance of the `Color` structure, which represents the `DarkRed` color. An instance of the `Pen` class is created and `FromArgb()` method retrieves the red color from the specified `DarkRed` color. The `DrawLine()` method draws a line at the specified location using the pen with the red color. Another instance of the `Pen` class is created to draw a line with the blue color.

#### → Methods

The methods of `Color` structure specify the different color combinations to create a `Color` structure.

Table 11.4 lists the most commonly used methods of `Color` structure.

Method	Description
<code>FromArgb</code>	Creates a <code>Color</code> structure consisting of four 8-bit alpha, red, green, and blue (ARGB) values.
<code>FromName</code>	Creates a <code>Color</code> structure from the mentioned name of a predefined color.
<code>ToArgb</code>	Retrieves the 32-bit alpha, red, green, and blue values.
<code>ToString</code>	Converts the <code>Color</code> structure into a human-readable string.

Table 11.4: Color Methods

The following code draws an arc using the specified color and a filled rectangle using the brush.

#### Code Snippet:

```
Graphics graphics = this.CreateGraphics();
Pen penArc = new Pen(Color.FromArgb(25, 121, 200));
SolidBrush solidRectangle = new SolidBrush(Color.FromName("Green"));
graphics.DrawArc(penArc, 120, 10, 100, 100, 0, 90);
graphics.FillRectangle(solidRectangle, 25, 30, 120, 130);
```

The code creates an instance of the `Graphics` class. It also creates an instance of the `Pen` class namely, `penArc`. The `FromArgb()` method retrieves the color with the specified RGB values. The `DrawArc()` method draws an arc in the specified location using the pen. An instance of `SolidBrush` is created and the rectangle is filled with green color using the brush.

### 11.3.2 Pen Class

The `Pen` class is used to represent a pen object. You can specify the width and style for various shapes using the `Pen` object. Even though you cannot inherit the `Pen` class, you can instantiate it to create objects. The `Pen` class is a sealed class and it exists in `System.Drawing` namespace. The line drawn by an object of the `Pen` class can be filled using variety of styles, which includes solid colors and textures. Table 11.5 lists the most commonly used properties and methods of `Pen` class.

Name	Description
Alignment	This property specifies or retrieves the alignment of the pen.
Brush	This property specifies or retrieves the <code>Brush</code> , which identifies the attributes of the pen.
Color	This property specifies or retrieves the color of the pen.
LineJoin	This property specifies or retrieves the style of joining the ends of two consecutive lines, which are drawn using the pen.
Transform	This property specifies or retrieves the geometric transformation for the pen.
Clone	This method creates an exact copy of the current pen.
Dispose	This method releases all the system resources used by the pen.

Table 11.5: Pen Class Members

The following code draws a line in gold color using the pen.

#### Code Snippet:

```
Graphics graphics = this.CreateGraphics();
Pen penLine = new Pen(Color.Gold);
penLine.Alignment = System.Drawing.Drawing2D.PenAlignment.Left;
penLine.LineJoin = System.Drawing.Drawing2D.LineJoin.Round;
Pen penLineClone = (Pen)penLine.Clone();
graphics.DrawLine(penLine, 200, 150, 20, 30);
graphics.DrawLine(penLineClone, 300, 200, 20, 30);
```

The code creates instances of the `Graphics` and the `Pen` classes. The constructor of the `Pen` class takes the `Gold` value of the `Color` structure. This means that whatever you draw with the pen will appear in gold color. The `Alignment` property is used to specify the position of the pen to the left. This is done by using the `PenAlignment` enumeration. The `LineJoin` property is used to specify the style of joining two lines as `Round`. This is done using the `LineJoin` enumeration. The `Clone()` method is used to create an exact copy of the current pen in use. The method returns an object, which is typecasted as the `Pen` object. The `DrawLine()` method draws two lines at the specified locations using the pen and the cloned pen, respectively.

### 11.3.3 Brush Class

Graphical shapes such as rectangles, polygons, ellipses, and quadrilaterals can be filled with colors by using the `Brush` class objects. Since this is an abstract base class, it cannot be instantiated. However, you can instantiate one of its derived classes to use the `Brush` class. GDI+ provides various types of brushes namely, solid, hatch, texture, and gradient.

The derived classes of the `Brush` class are as follows:

→ **SolidBrush**

The `SolidBrush` class allows you to fill an area with a single solid color. In order to create an object of the `SolidBrush` class, the constructor needs to be called and a `Color` structure should be passed as its parameter.

→ **HatchBrush**

The `HatchBrush` class allows you to implement a rectangular brush with a hatch style among the 54 unique styles, a foreground color and a background color. The constructor of the `HatchBrush` class accepts `HatchStyle` as its initial parameter and color as the subsequent parameter.

→ **TextureBrush**

The `TextureBrush` class allows you to use an image as a brush to fill the interior of a shape. It is mostly used to fill a `Graphics` object with images in a pattern, such as tile.

→ **LinearGradientBrush**

The `LinearGradientBrush` class is used to characterize a brush to paint using a linear gradient. The gradients are defined along a line by specifying the width of a rectangle or by mentioning any two points. This class encapsulates both two-color and custom multicolor gradients.

→ **PathGradientBrush**

The `PathGradientBrush` class allows you to fill a graphics path with a gradient. The `PathGradientBrush` class represents a blend of colors, which starts from the center of a graphics path and ends at the outside boundary of the path.

The following code uses a `HatchBrush` to fill a pie at the specified section.

**Code Snippet:**

```
Graphics graphics = this.CreateGraphics();
HatchBrush hatchBrush = new HatchBrush(HatchStyle.ZigZag, Color.
BurlyWood);
graphics.FillPie(hatchBrush, ClientRectangle, 0, 90);
```

The code creates an instance of the `HatchBrush` class to represent a rectangular brush with the `ZigZag` style. The style represents horizontal lines with zigzags and it is defined in the `HatchStyle` enumeration. The `FillPie()` method fills the specified section of the pie using the brush.

The methods of `Brush` class are used to clone a brush or specify a reference to a brush object while it is being used in a derived class.

Table 11.6 lists the most commonly used methods of Brush class.

Method	Description
Clone	Creates an exact copy of the brush when this method is overridden.
Dispose	Releases all system resources used by the Brush.
SetNativeBrush	Specifies a reference to a GDI+ brush object when it is used in a derived class.

Table 11.6: Brush Methods

The following code demonstrates how to draw an ellipse using the cloned SolidBrush object.

#### Code Snippet:

```
Graphics graphics = this.CreateGraphics();
Brush brush = new SolidBrush(Color.SteelBlue);
graphics.FillEllipse(brush, 70, 10, 50, 50);
Brush brushClone = (Brush)brush.Clone();
graphics.FillEllipse(brushClone, 10, 10, 50, 50);
brush.Dispose();
```

The code creates an instance of the Brush class by initializing the constructor of the SolidBrush class. The constructor of the SolidBrush class sets the color of the brush to SteelBlue. The FillEllipse() method fills the ellipse at the specified location with the steel blue color using the Brush. The Clone() method is used to create an exact copy of the current Brush object. The method returns an object, which is typecasted as the Brush object.

This cloned brush is used to fill the other ellipse at the specified location. The Dispose() method is invoked to free all system resources associated with the brush.

### Knowledge Check 3

1. Which of these statements about Color structure, Pen class, and Brush class are true?

(A)	The Color structure supports creation of transparent colors using the alpha value.
(B)	The Color structure exists in System.Drawing.Color namespace.
(C)	The Argb property creates a Color structure from the particular name of a predefined color.
(D)	The Transform property of the Pen class identifies the geometric transformation for the pen.
(E)	The SetNativeBrush() method of the Brush class retrieves the reference of the GDI+ brush.

2. You want to draw an arc in blue color using a pen. In addition, you want to position the pen towards the left. Which one of the following codes will help you to achieve this?

(A)	Graphics graphics = this.CreateGraphics(); Pen penLine = new Pen(Color.Blue); penLine.Alignment = System.Drawing.Drawing2D.PenAlignment.Left; graphics.DrawArc(penLine, new Rectangle(10, 10, 100, 100), 10, 200);
(B)	Graphics graphics = CreateGraphics(); Pen penLine = new Pen(Color.Blue); penLine.Alignment = System.Drawing.PenAlignment.Left; graphics.DrawArc(penLine, new Rectangle(10, 10, 100, 100), 10, 200);
(C)	Graphics graphics = this.CreateGraphics(); Pen penLine = (Color.Blue); Alignment = System.Drawing.PenAlignment.Left; graphics.DrawArc(penLine, new Rectangle(10, 10, 100, 100), 10, 200);
(D)	Graphics graphics = CreateGraphics(); Pen penLine = new Pen(Color.Blue); Alignment = System.Drawing.Drawing2D.PenAlignment.Left; graphics.DrawArc(penLine, new Rectangle(10, 10, 100, 100), 10, 200);

## 11.4 Understanding the Font Class

In this fourth lesson, **Understanding the Font Class**, you will learn to:

- Explain the `Font` class and its use.
- Explain the `FontFamily` class and its use.

### 11.4.1 Font Class

The `Font` class provides the font face, size, and style attributes to format text. This is a sealed class that is defined in the `System.Drawing` namespace.

Windows provides two types of fonts namely, GDI fonts and device fonts. GDI fonts are located as files on your system, normally in the `Fonts` folder under system root. They are used to format text. All fonts have their own file. For example, Arial, Arial Bold, and Arial Italic are all different fonts in the Arial Font family and are represented by different files. Device fonts are used for output devices such as monitors or printers.

Figure 11.3 displays the Fonts.

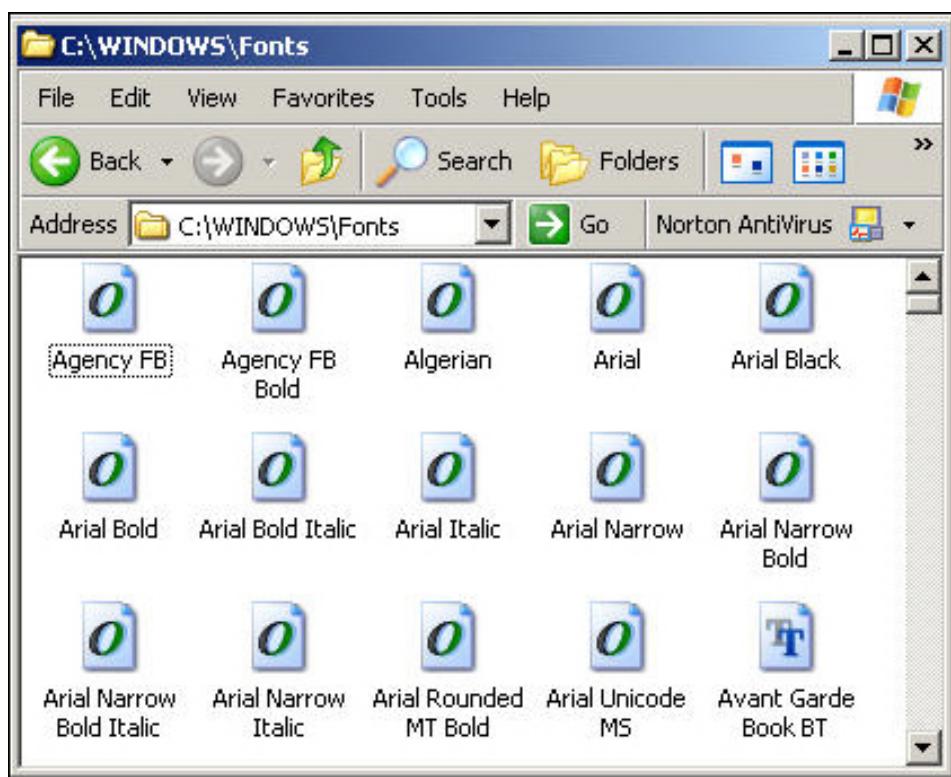


Figure 11.3: Fonts

#### → Properties and Methods

The properties of the `Font` class are used to specify the font family and font styles. Table 11.7 lists the commonly used properties and methods of `Font` class.

Name	Description
<code>Bold</code>	This property retrieves a value, which indicates whether the font is bold.
<code>FontFamily</code>	This property retrieves <code>FontFamily</code> , which is associated with the font.
<code>Italic</code>	This property retrieves a value, which indicates whether the font is italic.
<code>Name</code>	This property retrieves the current font face name.
<code>Strikeout</code>	This property retrieves a value, which indicates whether the font has a horizontal line through it.
<code>Style</code>	This property retrieves the style information for the current font.
<code>Underline</code>	This property retrieves a value that indicates whether the current font is underlined.
<code>GetHeight</code>	This method returns the line spacing of the current font.
<code>ToString</code>	This method returns a string using the current font.

Table 11.7: Font Class Members

The following code demonstrates how to draw a string with `Font` and `Brush` objects.

#### Code Snippet:

```
Font font = new Font(SystemFonts.DefaultFont, FontStyle.Regular);
if (!this.Font.Bold)
    font = new Font(this.Font.FontFamily, 20, FontStyle.Bold);
Graphics graphics = this.CreateGraphics();
graphics.DrawString("Font Demo", font, Brushes.Red, new PointF(30,
30));
```

The code creates an instance of the `Font` class namely, `font`. The constructor of the class takes two parameters namely, `SystemFonts.DefaultFont` and the font style, which is set to `Regular` using the `FontStyle` enumeration. If the font style specified is not `Bold`, then, the style is set to `Bold` using the `FontStyle` enumeration. A `Graphics` object is created and the `DrawString()` method draws the string text on the form with the specified font.

### 11.4.2 *FontFamily Class*

The `FontFamily` class is used to define various font faces. These font faces provide you with different styles having a similar basic design and certain variations in styles. The class exists in `System.Drawing` namespace and it cannot be inherited.

#### → Properties

The properties of the `FontFamily` class allow you to retrieve an array of the `FontFamily` objects or to retrieve the name of the current `FontFamily`. Table 11.8 lists the commonly used properties of `FontFamily` class.

Property	Description
Families	Retrieves an array of the <code>FontFamily</code> objects related to the current graphics context.
GenericMonospace	Retrieves a generic monospace <code>FontFamily</code> .
GenericSansSerif	Retrieves a generic sans serif <code>FontFamily</code> object.
GenericSerif	Retrieves a generic serif <code>FontFamily</code> .
Name	Retrieves the name of the current <code>FontFamily</code> .

Table 11.8: `FontFamily` Properties

#### → Methods

The methods of `FontFamily` class help in getting the name of `FontFamily` in a particular language.

Table 11.9 lists the most commonly used methods of `FontFamily` class.

Method	Description
GetFamilies	Retrieves an array of all <code>FontFamily</code> objects, which are present in the given graphics context.
GetLineSpacing	Retrieves the spacing of line in design units of the <code>FontFamily</code> of a given style. This line spacing is the vertical distance between the base lines of two consecutive lines of text.
GetName	Retrieves the name of the current <code>FontFamily</code> in a given language.
ToString	Converts the current <code>FontFamily</code> into a human-readable string.

**Table 11.9: `FontFamily` Methods**

The following code creates a `TextBox` control and applies font to the text in the control.

#### Code Snippet:

```
TextBox txtFile = new TextBox();
this.Controls.Add(txtFile);
FontFamily fontFamily = new FontFamily("Verdana");
txtFile.Font = new Font(fontFamily.Name, 12);
if (fontFamily.GetLineSpacing(FontStyle.Regular) > 2000)
{
    txtFile.Font = new Font("Courier New", 15);
}
```

The code creates a `TextBox` control and an instance of the `FontFamily` class namely, `fontFamily`. The constructor of `FontFamily` class takes the `Verdana` font as the parameter. The constructor of the `Font` class is invoked to apply the `Verdana` font and font size as 12 to the text in the text box. The `GetLineSpacing()` method is used to retrieve the line spacing of the text, whose style is `Regular`. If this spacing is more than 2000 pixels, the font in the control is set to `Courier New` and size to 15.

#### Knowledge Check 4

- Can you match the properties and methods of `FontFamily` class against their corresponding descriptions?

Description		Property and Method
(A)	This property returns an array, which contains all the <code>FontFamily</code> objects.	(1) GetFamilies
(B)	This property retrieves the name of the current <code>FontFamily</code> .	(2) GetName
(C)	This method retrieves an array of objects of <code>FontFamily</code> class.	(3) ToString
(D)	This method returns the name of <code>FontFamily</code> in a given language.	(4) Name

Description		Property and Method	
(E)	This method converts the <code>FontFamily</code> object into a string representation.	(5)	Families

2. You want to apply the font, Arial-15 with style as `StrikeOut` to the text 'This is richtextbox' inside the `RichTextBox` control. Which one of the following codes will help you to achieve this?

(A)	<pre>RichTextBox rtbExp = new RichTextBox(); FontFamily family = new FontFamily(); Font font = new Font(family, Strikeout); rtbExp.Text = "This is richtextbox"; rtbExp.Font = font;</pre>
(B)	<pre>RichTextBox rtbExp = new RichTextBox(); FontFamily family = new FontFamily(); Font font = new Font(family, 15, Strikeout); rtbExp.Text = "This is richtextbox"; font = rtbExp.Font;</pre>
(C)	<pre>RichTextBox rtbExp = new RichTextBox(); FontFamily family = new FontFamily("Arial"); Font font = new Font(family, 15, FontStyle.Strikeout); rtbExp.Text = "This is richtextbox"; rtbExp.Font = font;</pre>
(D)	<pre>RichTextBox rtbExp = new RichTextBox(); FontFamily family = new FontFamily("Arial"); Font font = new Font(family, 15, FontStyle.Strikeout); Text = "This is richtextbox"; Font = font;</pre>

## 11.5 Working with Advanced GDI+

In this last lesson, **Working with Advanced GDI+**, you will learn to:

- Identify the process of displaying images.
- Explain the process of printing GDI+ objects.
- Describe double buffering in GDI+.

### 11.5.1 Displaying Images

GDI+ can be used to display images on the form. This is done by creating an object of an `Image` class (such as `Bitmap`) and creating a `Graphics` object that refers to the drawing surface. The `DrawImage()` method of the `Graphics` class should be invoked. The `Bitmap` class is derived from the `Image` class and it exists in `System.Drawing` namespace. The `Image` class is an abstract class used to provide functionalities for the `Bitmap` and `Metafile` classes.

The `DrawImage()` is an overloaded method that takes many parameters. The different parameters that can be specified with this method are as follows:

#### → **DrawImage(Image, Point)**

The `DrawImage(Image, Point)` method takes the parameter `Image` to draw the image. The `Point` parameter specifies the location of the upper-left corner of the image.

#### → **DrawImage(Image, float, float)**

The `DrawImage(Image, float, float)` method takes the parameter `Image` to draw the image. The second and the third parameters represent the x and y co-ordinates of the upper-left corner of the image respectively.

#### → **DrawImage(Image, int, int, int, int)**

The `DrawImage(Image, int, int, int, int)` method takes the parameter `Image` to draw the image. The second and the third parameters represent the x and y co-ordinates of the upper-left corner of the image respectively. The fourth and the fifth parameters represent the width and height respectively.

The following code displays the image using the `DrawImage()` method.

#### **Code Snippet:**

```
protected override void OnPaint(PaintEventArgs e)
{
    int x, y;
    Image image = Image.FromFile("sample.jpg");
    Graphics graphics = e.Graphics;
    x = 20;
    y = 20;
    graphics.DrawImage(image, x, y);
}
```

The code overrides the `OnPaint()` method. The `FromFile()` method retrieves the specified .jpg image file. The method takes the image file name as the parameter. The `DrawImage()` method draws the specified image at the specified x- and y-coordinates.

### 11.5.2 Printing GDI+ Objects

The .NET Framework provides various printing features that use the existing GDI+ classes. These classes are defined in `System.Drawing.Printing` namespace. This allows you to provide print-related services to your application. In order to do so, a new instance of the `PrintDocument` class should be created. You should then, set appropriate properties that describe what to print and finally invoke the `Print()` method.

The `PrintController` class, when implemented in a derived class, lets the printer know about how the document is to be printed.

The following code demonstrates how to use the `PrintDocument` class for printing.

**Code Snippet:**

```
Label lblPrintStatus = new Label();
PrintDocument printDocument = new PrintDocument();
printDocument.DocumentName = "printSalesReport";
printDocument.Print();
...
private void printDocument_PrintPage(object sender, PrintPageEventArgs e)
{
    lblPrintStatus.Text = "Printing..";
}
```

The code creates a `Label` control and an instance of the `PrintDocument` class, namely `printDocument`. The `DocumentName` property is used to display the name of the document in the printer queue. The `Print()` method is invoked to print the document. This method, in turn, raises the `PrintPage` event. When the printing is in process, the label displays the text `Printing..`.

#### → Purpose

One of the most common problems with complex images is that they flicker or their appearance is not clear. This happens due to complex combinations of shapes, text, and painting being done to the images. Another reason for flickering is the frequent redrawing of images on the screen.

To overcome the problem of flickering and to make the appearance of the images appealing, the .NET Framework provides the **Double Buffering** technique. In this technique, instead of drawing the image on the screen, all the drawing and painting operations to create the image is first performed in a buffer memory from the source location. The source location is the area where the image is created programmatically. Then, from the buffer, the image is drawn on the target screen. This switching between the buffer memory and screen reduces the flickering and improves the appearance of the image.

### 11.5.3 Double Buffering

Double buffering is used to lessen the rendering time of the image on the screen. At times, it becomes time consuming to create multiple painting operations even by using the most suitable graphic tools. Under such circumstances, double buffering is used.

The double buffering technique reduces the flicker and helps in providing faster and smoother drawings. The double buffering technique also helps in improving the appearance and rendering of graphics.

By default, double buffering is activated for Windows Forms. However, to implement double buffering, you can set the `DoubleBuffered` property of the `Control` class to `true`. Alternatively, you can also, invoke the `SetStyle()` method of the `Control` class to set the `OptimizedDoubleBuffer` value to `true`. The `OptimizedDoubleBuffer` value is defined in the `ControlStyles` enumeration to enable double buffering.

The following code creates a `PictureBox` control and uses the `DoubleBuffered` property.

**Code Snippet:**

```
PictureBox picImage = new PictureBox();
this.Controls.Add(picImage);
this.DoubleBuffered = true;
picImage.Image = Image.FromFile("alley.jpg");
```

The code creates an instance of the `PictureBox` class. The `DoubleBuffered` property is `true`, which reduces the flicker effect of an image. The `FromFile()` method retrieves the specified `.jpg` image file. The method takes the image file name as the parameter and returns a reference to the `Image` class. The `Image` property is used to specify the image, which is to be displayed by the `PictureBox` control.

**Note** - The `SetStyle()` method of `Control` class is only recommended for custom controls.

## Knowledge Check 5

- Which of these statements about displaying, printing, and buffering GDI+ objects are true?

(A)	The <code>DrawImage()</code> method of <code>Graphics</code> class is used to draw the image into the buffer.
(B)	The <code>Bitmap</code> class is inherited from the <code>Image</code> class.
(C)	The <code>PrintDocuments</code> class is used to print the GDI+ objects.
(D)	Double buffering is used to render the images faster on the screen.
(E)	The <code>DoubleBuffer</code> property is used to enable double buffering.

2. You want to fill a rectangle with black color and print it using the `PrintDocument` class. Which one of the following codes will help you to achieve this?

(A)	<pre>PrintDocument printDocument = new PrintDocument(); printDocument.DocumentName = "GraphicsObjects"; printDocument.PrintPage += new PrintPageEventHandler(printDocument_PrintPage); printDocument.Print(); private void printDocument_PrintPage(object sender, PrintPageEventArgs e) {     e.Graphics.FillRectangle(Brushes.Black, ClientRectangle); }</pre>
(B)	<pre>PrintDocument printDocument = new PrintDocument(); printDocument.DocumentName = "GraphicsObjects"; printDocument.PrintPage = new PrintPageEventHandler(printDocument_PrintPage); printDocument.Print(); private void printDocument_PrintPage(object sender, PrintPageEventArgs e) {     Graphics.FillRectangle(Brushes.Black, ClientRectangle); }</pre>
(C)	<pre>PrintDocument printDocument = new PrintDocument(); printDocument.DocumentName = "GraphicsObjects"; printDocument.PrintPage = new PrintPageEventHandler(printDocument_PrintPage); printDocument.Print(); private void printDocument_PrintPage(object sender, PrintPageEventArgs e) {     Graphics.FillRectangle(Brushes, ClientRectangle); }</pre>

(D)

```
PrintDocument printDocument = new PrintDocument();
printDocument.DocumentName = "GraphicsObjects";
printDocument.PrintPage += new PrintPageEventHandler(printDocument);
printDocument.Print();
private void PrintPage(object sender, PrintPageEventArgs e)
{
    e.Graphics.FillRectangle(Brushes.Black, ClientRectangle);
}
```



## Module Summary

In this module, **Working with GDI+**, you learnt about:

→ **Introduction to Graphics with GDI+**

GDI+ allows you to draw, paint, and render simple to complex images on a form. It helps in fast and efficient rendering of graphics and supports any display device. It works in coordination with the device context that gives information about the pens and brushes in use.

→ **Graphics Class**

The Graphics class is an abstract class used to draw shapes on the screen. You can create a Graphics object by using the Paint() event, CreateGraphics() method, or from any object that is derived from the Image class.

→ **Introduction to Graphics Objects**

The Color structure provides various colors by using the alpha, red, green, and blue color values. The Pen class allows you to draw various shapes with the specified dimensions. You can also fill the created shapes using variety of styles, which includes solid colors and textures. The Brush class is used to represent a brush to fill the shapes with colors.

→ **Understanding the Font Class**

The Font class is used to format the text by applying different font, font styles, and sizes. The FontFamily class is used to specify the font family, which can be applied to the text.

→ **Working with Advanced GDI+**

The DrawImage() method is used to display images on the screen. The PrintDocument class is used to print the documents from Windows Forms. The SetStyle() method and the DoubleBuffered property of the Control class are used to enable double buffering.

# Module - 12

## Custom Controls

Welcome to the Module, **Custom Controls**.

Custom controls are created by the user so that they can be reused across multiple applications. You can create these controls when the Windows Forms existing controls do not cater to your application requirements. Custom controls can be composite, extended, or custom depending on how they are created. They can be created either by inheriting the Control class or UserControl class, or any of the Windows Forms existing controls.

In this Module, you will learn about:

- ➔ Introduction to Custom Controls
- ➔ Creating Custom Controls
- ➔ Using Custom Controls
- ➔ Creating and Using Composite Controls

## 12.1 Introduction to Custom Controls

In this first lesson, **Introduction to Custom Controls**, you will learn to:

- Explain the need for custom controls.
- List the types of custom controls.

### 12.1.1 Need for Custom Controls

Windows Forms provides a rich set of controls such as `Textbox`, `Button`, and so on. These controls cater to most of your application requirements. Sometimes, your application might demand certain functionalities that these existing controls do not fulfill. This is where the need for custom controls arises.

Custom controls are user-defined controls that can be reused across various applications. They save the time of the developer to create the same kind of user interface multiple times. Custom controls can be used to perform various validations in the control. A user login page is an example of custom control where you can reuse the control across multiple applications.

### 12.1.2 Features

The advantage of using custom controls is that once it is created, it can be reused across applications. There are certain features of custom controls, some of which are as follows:

#### → User-defined Validation

User-defined validation allows you to create validation forms that can be reused across multiple applications. It allows you to define the type of validation the different controls will perform, such as the text box allowing special characters.

#### → Customized Design

Customized design allows you to design the user interface of the custom control. This involves where and how the controls are placed and designed.

### 12.1.3 Types of Custom Controls

Windows Forms provides different types of custom controls. These controls can be reused across multiple applications. The different types of custom controls are as follows:

#### → Composite

A composite control is created by inheriting the `UserControl` class. The composite control contains a set of Windows Forms existing controls. You can create a composite control when you want to combine the functionalities of several Windows Forms existing controls.

→ **Extended**

An extended control is derived from any Windows Forms controls such as a button. This allows you to inherit all the functionalities of the Windows Forms control and then, extend its functionality according to your requirements.

→ **Custom**

A custom control is created by inheriting the `Control` class. The custom control provides greater flexibility than the composite and extended controls. You can create a custom control if you want to implement certain functionalities that the standard controls do not provide.

**Note** - Composite controls are controls that are enclosed in a common container. The contained controls are called constituent controls.

Constituent controls are controls that form a user control. The constituent controls are generally declared as `private`. This prevents the users from directly accessing them.

## Knowledge Check 1

- Which of the following statements about custom controls and their types are false?

(A)	Custom controls allow you to create reusable controls.
(B)	User-defined validation allows you to define the type of validation the user can perform.
(C)	A composite control allows you to create a control by inheriting from any standard control.
(D)	Customized design allows you to specify the appearance of standard controls on the custom control.
(E)	An extended control allows you to create a control by inheriting from the <code>Control</code> class.

## 12.2 Creating Custom Controls

In this second lesson, **Creating Custom Controls**, you will learn to:

- Describe how to create and use simple custom controls.
- List and explain how to add properties to custom controls.
- Explain how to associate events with custom controls.

### 12.2.1 Create and Use Simple Custom Controls

Simple custom controls can be created in three different ways. You can create a simple custom control by inheriting any of the Windows Forms existing controls, or the `Control` class, or the `UserControl` class. When you create a simple custom control by inheriting any of the Windows Forms existing controls such as `TextBox`, all the functionalities of that control are inherited by default.

The `Control` class is the base class for all controls and it provides the basic functionality for displaying information to the user. The `UserControl` class provides a control that can hold other controls. The `Control` and `UserControl` classes exist in `System.Windows.Forms` namespace.

A rounded button or a `TextBox` with rounded edges are some examples of simple custom controls.

The following code demonstrates how to create a custom control by inheriting the Windows Forms existing control, `TextBox`.

#### Code Snippet:

```
public partial class NumericTextBox : TextBox
{
    protected override void OnKeyPress(KeyEventEventArgs e)
    {
        base.OnKeyPress(e);
        if (!char.IsDigit(e.KeyChar) && e.KeyChar != 8 && e.KeyChar != 13)
        {
            MessageBox.Show("Please enter a numeric value.", "NumericTextBox");
            e.Handled = true;
        }
        else
        {
            e.Handled = false;
        }
    }
}
```

The code creates a custom control named, `NumericTextBox`, by inheriting the Windows Forms existing control, `TextBox`. The class `NumericTextBox` overrides the method `OnKeyPress()`. The `base` keyword invokes the base class method `OnKeyPress()`. If the user enters characters in the text box, then, a message box is displayed requesting him to enter numeric values.

A simple custom control can be created by deriving the Windows Forms controls, or `Control` class, or `UserControl` class. The steps to create a custom control are listed as follows:

#### → Define a Class

You need to define a class that is derived from the `Control` class of `System.Windows.Forms` namespace.

## → Define Properties

You need not explicitly define properties for the control since the control inherits all the properties of the `Control` class. Generally most custom controls do define some additional properties to include more functionality.

## → Override OnPaint() Method

You need to override the protected `OnPaint()` method that is inherited from the `Control` class. This method should be overridden to define the logic for the custom control to redraw by itself. If you fail to override the `OnPaint()` method, then, your control will not be able to redraw by itself.

## → Provide Attributes

You need to provide attributes to your custom control to appropriately display the control at design time. By defining attributes, you can align the text on the control or display the property values in a logical category.

## → Provide Resources

You can provide resources, such as bitmap, to your control by using the `/res` option. This step is optional. You can retrieve the resources at runtime by using the methods of the `ResourceManager` class.

## → Compile and Deploy

You need to save your control with `.cs` extension. You can compile and deploy the code into an assembly from the **Build Solution** option of the **Build** menu in Visual Studio IDE.

The following code creates a simple custom control by inheriting the `Control` class.

### Code Snippet:

```
public partial class CustomUserButton : Control
{
    private Color backgroundColor = Color.Purple;
    public Color BackgroundColor
    {
        get
        {
            return backgroundColor;
        }
        set
        {
            backgroundColor = value;
        }
    }
    protected override void OnPaint(PaintEventArgs e)
```

```

    {
        this.BackColor = backgroundColor;
    }
}

```

The code creates a `CustomUserButton` class that is derived from `Control` class. A property named `BackgroundColor` is created. The `set` accessor assigns the value to the `backgroundColor` variable. The `get` accessor returns the `backgroundColor` value. The `OnPaint()` method is overriden. The back color of the custom control is set to `backgroundColor`, which is defined in the property `BackgroundColor`.

### 12.2.2 Properties and Methods of Control Class

The `UserControl` class is derived from the `Control` class and it inherits all the properties and methods of the `Control` class. The properties of the `Control` class allow you to modify and work with the custom controls and the container. Table 12.1 lists the most commonly used properties and methods of the `Control` class.

Name	Description
<code>Controls</code>	This property retrieves a collection of controls present in the specified control.
<code>Enabled</code>	This property specifies or retrieves a value that indicates whether the specified control can respond to user interaction.
<code>Location</code>	This property specifies or retrieves the coordinates of the control with respect to its container.
<code>Name</code>	This property specifies or retrieves the name of the control.
<code>Parent</code>	This property specifies or retrieves the parent container of the control.
<code>Text</code>	This property specifies or retrieves the text associated with the control.
<code>Invalidate</code>	This method invokes the <code>OnPaint()</code> method to send a paint message to the control.
<code>OnPaint</code>	This method fires the <code>Paint</code> event, which occurs when the user redraws the control.

Table 12.1: Control Class Members

#### → Adding Properties Using Accessors

Properties allow you to apply different settings to your control. Properties are private data members and they represent the way in which the user interacts with the custom control. Properties contain codes that allow you to get and set the values of the private members. This can be done by using the `get` and `set` accessors of the property. The `set` accessor assigns a new value and the `get` accessor returns the property value back to the user. The property becomes read-only if it contains only the `get` accessor and it becomes write-only if it contains only the `set` accessor.

The following code demonstrates how to create properties using accessors.

**Code Snippet:**

```
public string StringText
{
    get
    {
        return stringText;
    }
    set
    {
        stringText = value;
        Invalidate();
    }
}
```

The code creates a property named `StringText`. The `set` accessor assigns a value to the `stringText` variable. The `Invalidate()` method calls the `OnPaint()` method, which fires the `Paint` event. The `get` accessor returns the `stringText` value.

#### → Events of Control Class

Events act as a link between an application and the user. Whenever the user performs any event such as clicking a mouse, the generated event is handled by an event-handler. An event-handler is a method that is invoked whenever an appropriate event is raised. This event-handler causes the application to respond to the user's interaction.

C# is an event-driven programming model. The event-driven programming model consists of publishers to generate the events and subscribers to fire the different event-handlers.

The `Control` class events are triggered when you generate certain events. Table 12.2 lists the most commonly used events of the `Control` class.

Event	Description
Click	Occurs when the user clicks the control.
DoubleClick	Occurs when the user double-clicks the control.
Enter	Occurs when the user enters the control.
KeyPress	Occurs when the user presses the key while the control is in focus.
MouseClick	Occurs when the user clicks the control by the mouse.
MouseDoubleClick	Occurs when the user double-clicks the control by the mouse.
Paint	Occurs when the user redraws the control.

Table 12.2: Events of Control Class

#### 12.2.3 Associating Events with Custom Controls

C# uses delegates to bind events to methods. These methods are invoked when events are raised. The delegate specifies an event handler that can be used to register events. The `+=` operator is used to assign

an event to a delegate. Whenever an event is generated, the delegate invokes the respective event handler. Figure 12.1 displays the events with custom controls.

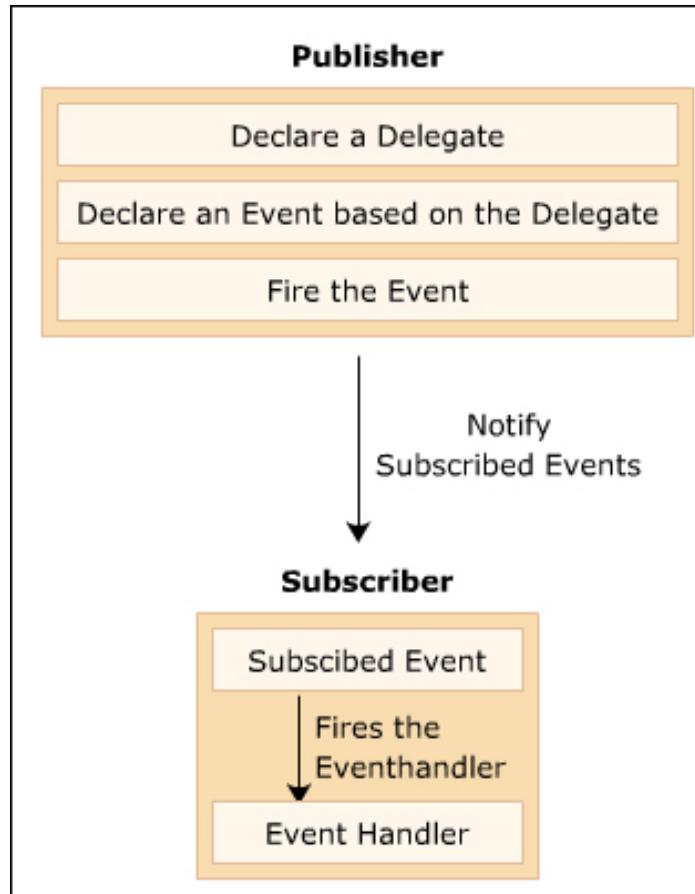


Figure 12.1: Events with Custom Controls

The following code defines an event and associates it with the custom control.

#### Code Snippet:

```

public event EventHandler Leave;
this.btnExit.Leave += new System.EventHandler(this.btnExit_Leave);
private void btnExit_Leave(object sender, EventArgs e)
{
    btnExit.ForeColor = Color.Cyan;
}

```

The code creates an event, `Leave`. The `Leave` event of the custom control `btnCustom` is associated with the event-handler `btnCustom_Leave`. The `+=` operator is used to associate the event with `btnCustom`. When the custom control loses focus, the `Leave` event occurs. This changes the fore color of the custom button.

## Knowledge Check 2

1. Can you match the properties and methods of the `Control` class against their corresponding descriptions?

Description			Properties and Methods
(A)	This property retrieves a collection of controls in the specified control.	(1)	Enabled
(B)	This property specifies or retrieves the parent container associated with control.	(2)	OnPaint
(C)	This property checks whether the control responds to user interaction.	(3)	Invalidate
(D)	This method sends a paint message to the control.	(4)	Parent
(E)	This method triggers the <code>Paint</code> event.	(5)	Controls

2. You want to display a custom `TextBox` that accepts only the current year. Which one of the following codes will help you to achieve this?

```
(A) public partial class CurrentYearTextBox : TextBox
{
    protected void OnLostFocus(EventArgs e)
    {
        if (this.Text == "" || this.Text.Length == 4)
        {
            if (this.Text != DateTime.Today.Year.ToString())
            {
                MessageBox.Show("Please enter the current year.",
                               "CurrentYearTextBox");
            }
        }
    }
}
```

(B)	<pre>public partial class CurrentYearTextBox : TextBox {     protected override void OnLostFocus(EventArgs e)     {         if (this.Text != ""    this.Text.Length == 4)         {             if (this.Text != DateTime.Today.Year.ToString())             {                 MessageBox.Show("Please enter the current year.",                     "CurrentYearTextBox");             }         }     } }</pre>
(C)	<pre>public partial class CurrentYearTextBox : Control {     protected override void OnLostFocus(EventArgs e)     {         if (this.Text != ""    this.Text.Length == 4)         {             if (this.Text != DateTime.Today.Year.ToString())             {                 MessageBox.Show("Please enter the current year.",                     "CurrentYearTextBox");             }         }     } }</pre>

(D)

```

public partial class CurrentYearTextBox : Control
{
    protected void OnLostFocus(EventArgs e)
    {
        if (this.Text == "" || this.Text.Length == 4)
        {
            if (this.Text != DateTime.Today.Year.ToString())
            {
                MessageBox.Show("Please enter the current year.",
                               "CurrentYearTextBox");
            }
        }
    }
}

```

## 12.3 Using Custom Controls

In this third lesson, **Using Custom Controls**, you will learn to:

- Describe how to consume custom controls in a form.
- Explain how to specify toolbox bitmaps for the custom control.
- Explain how to create custom controls from existing Windows Forms controls.

### 12.3.1 Using Custom Controls

Windows Forms allows you to use custom controls in Windows applications. Before using the custom control, you need to first create the control and then, add it to the Toolbox. The steps for using the custom control in your application are as follows:

1. Right-click the Visual Studio IDE Toolbox.
2. Click the **Choose Items** option.
3. Select the **.NET Framework Components** tab.
4. Click **Browse** and select the appropriate custom control .dll file.

Windows Forms allows you to programmatically use the custom controls in Windows applications.

You can use the custom control programmatically by using the namespace in which the custom control is created. An instance of the custom control class can be created to use the control in the application.

The following code demonstrates how to use a custom control in your application.

#### Code Snippet:

```
using CustomControls;
public partial class frmEmployee : Form
{
    . .
    private void frmEmployee_Load(object sender, EventArgs e)
    {
        Label lblName = new Label();
        Label lblAge = new Label();
        TextBox txtName = new TextBox();
        NumericTextBox numTextAge = new NumericTextBox();
    }
}
```

The code uses the custom control in the application by using the `CustomControls` namespace. The `CustomControls` namespace is a user-defined namespace in which a custom control `NumericTextBox` is created. An instance of the `NumericTextBox` class is created to use the control in the application.

#### 12.3.2 Toolbox Bitmaps

Toolbox bitmaps represents an image for the control that appears in the Toolbox. The size of the toolbox bitmaps are 16 by 16 pixels. You can specify a particular image for your control in the Toolbox by using the `ToolboxBitmapAttribute` class. The `ToolboxBitmapAttribute` class is an attribute that allows you to provide various images or icons to the custom control in the Toolbox.

The `ToolboxBitmapAttribute` class allows you to specify a string parameter that represents the path and file name of the bitmap image. This string is included in the class declaration of your control before the `class` keyword. The class can also include the `Type` parameter, which loads the bitmap associated with the specified type.

The `ToolboxBitmapAttribute` class is inherited from `Attribute` class and it exists in `System.Drawing` namespace. The `Attribute` class acts as a base class for all custom attributes.

The following code adds the `ToolboxBitmapAttribute` in the class declaration.

#### Code Snippet:

```
[ToolboxBitmap(@"c:\Images\myImage.bmp")]
public partial class CustomControl : UserControl
{
    . .
}
```

The code specifies the toolbox bitmap for the custom control, `CustomControl`. The `ToolboxBitmap` specifies the bitmap file for the control.

### 12.3.3 Inherit Existing Controls

Windows Forms allows you to inherit the existing Windows Forms controls such as button and label. For example, if you inherit the `Label` control, the new control will appear and behave like a label. Also, you can extend and modify the functionalities of the new control.

The steps for inheriting from existing Windows Forms controls are listed as follows:

- Create a new Windows Application project.
- From the **Project** menu, select **Add New Item**. The Add New Item dialog box appears.
- Double-click **Inherited User Control**. A new custom control is added to your application project.
- Browse the control from which you want to inherit.
- In the code editor, change the name of the base class to the name of the control class from which your custom control is inherited. For example, if you are inheriting from the `Button` class, change the code as:

```
public class UserControl1: System.Windows.Forms.Button
```

- Include the required properties and methods.
- Override the `OnPaint()` method to change the control's appearance.
- Save and build the control.

The steps for inheriting from the `Control` class are listed as follows:

- Create a new **Windows Application** project.
- From the **Project** menu, select **Add User Control**. The **Add New Item** dialog box appears.
- Double-click **Custom Control**. A new custom control is added to your application project.
- In the **Code Editor**, ensure that the line that specifies the base class mentions the `Control` class.

The steps for inheriting from the `UserControl` class are listed as follows:

- Create a new **Windows Control Library** project.

- Add the controls from the Toolbox to the designer.
- Include the required properties and methods.
- Save and build the control.

### *Knowledge Check 3*

1. Which of the following statements about custom controls are false?

(A)	Custom controls are added by right-clicking the toolbox and clicking the .NET Framework Components option.
(B)	Windows Forms existing controls cannot be inherited in your application.
(C)	The <code>ToolboxBitmapAttribute</code> class is used to specify a particular image for the custom control.
(D)	Toolbox bitmap is specified by adding <code>ToolboxBitmapAttribute</code> inside the class declaration of the custom control.
(E)	Custom controls are programmatically used in your application by using the namespace in which the control is created.

### *12.4 Creating and Using Composite Controls*

In this last lesson, **Creating and Using Composite Controls**, you will learn to:

- Describe the steps to create a composite control.
- List the steps to capture events of composite controls.

#### *12.4.1 Creating a Composite Control*

Composite controls are user-defined controls that allow you to create graphical interfaces using one or more Windows Forms controls. Composite controls are also called as user controls and they are created by inheriting `System.Windows.Forms.UserControl` class. Composite controls can be inherited like any other Windows Forms controls.

The advantages of composite controls are as follows:

- Allows you to merge data easily.
- Creates a .dll file that acts as an assembly.

Windows Forms allows you to create a composite control that can be reused across applications.

You can create a composite control either as a part of a **Windows Application** project, or a **Windows Control Library** project. The steps for creating a composite control in a Windows Application are as follows:

- Create a new **Windows Application** project.
- From the **Project** menu, select **Add User Control**. The User Control is added to your application.
- Add Windows Forms existing controls to the composite control design surface.
- Place code in event procedures.
- Save the file.
- In the **Build** menu, click **Build Solution**.
- The composite control appears in the Toolbox when you open any form in the designer mode.

Figure 12.2 displays the composite control.

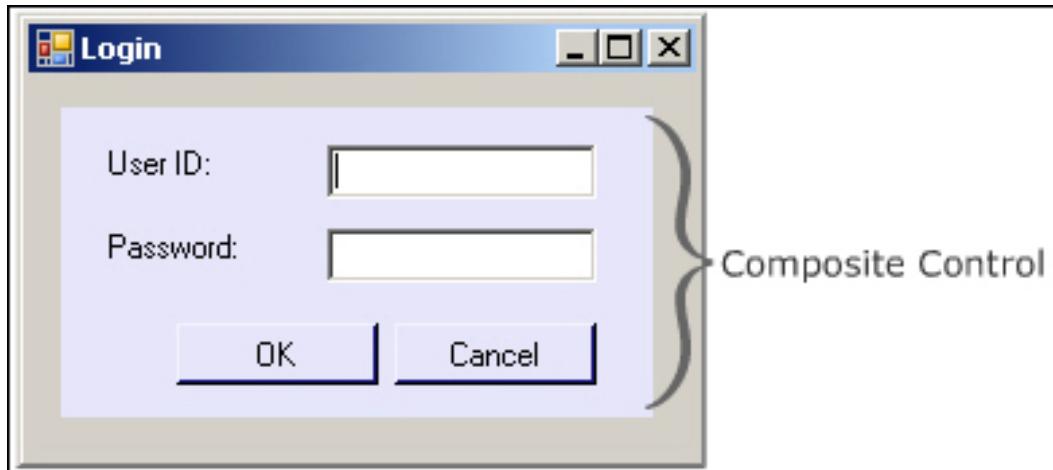


Figure 12.2: Composite Control

### 12.4.2 Capture Events

Windows Forms allows you to add events to composite controls. You can also customize event-handlers such that they raise events that you define. There are two methods to customize an event. One of the methods is to create a custom event. The steps for creating a custom event are as follows:

- Add the existing Windows Forms controls to create a composite control.
- Create the event for the controls included in the composite control.

- Add the event-handler for the different controls.

The other way of customizing an event is by overriding the base class events. By overriding the base class events, you can add more functionality to the events.

The following code customizes an event by overriding the base class `Leave` event.

**Code Snippet:**

```
public partial class CustomProductTextBox : TextBox
{
    public CustomProductTextBox()
    {
        InitializeComponent();
    }
    protected override void OnLeave(EventArgs e)
    {
        if (this.Text == "")
        {
            MessageBox.Show("Please enter the product id.", "Product
                Details", MessageBoxButtons.OK,
                MessageBoxIcon.Information);
            this.Focus();
        }
        else
        {
            if (this.Text.Length < 5 || !this.Text.Substring(0,
5).Equals("ML-F-"))
            {
                MessageBox.Show("Please enter the valid product id.", "Product Details", MessageBoxButtons.OK,
                MessageBoxIcon.Information);
                this.BackColor = Color.LightPink;
                this.Focus();
            }
            else
            {
                this.ForeColor = Color.LightGreen;
            }
        }
    }
}
```

The code creates a custom control namely, `CustomProductTextBox`. This class is inherited from the `TextBox` class. The base class event `Leave` is overridden. This event occurs when the focus on the control is lost. The control checks whether the product ID entered is in valid format.

## Knowledge Check 4

1. Which of the following statements about composite controls are false?

(A)	Composite controls are used for creating custom graphical interfaces.
(B)	Composite controls are reused across multiple applications.
(C)	Composite controls are created only as a part of the Windows Application project.
(D)	Composite controls contain only one Windows Forms control in a container.
(E)	Composite controls cannot be inherited from other controls.

2. You want to display a custom button that changes its fore color and font when it receives focus. Which one of the following codes will help you to achieve this?

(A)	<pre>public partial class BoldButton : Button {     protected void OnGotFocus(EventArgs e)     {         this.Font = new Font(this.Font, FontStyle.Bold               FontStyle.Underline);         this.ForeColor = Color.LightSalmon;     } }</pre>
(B)	<pre>public partial class BoldButton : Button {     protected override void OnGotFocus(EventArgs e)     {         this.Font = new Font(this.Font, FontStyle.Bold               FontStyle.Underline);         this.ForeColor = Color.LightSalmon;     } }</pre>

(C)	<pre>public partial class BoldButton : Button {     protected override void OnGotFocus(EventArgs e)     {         Font = new Font[this.Font, FontStyle.Bold,         FontStyle.Underline];         this.ForeColor = Color.LightSalmon;     } }</pre>
(D)	<pre>public partial class BoldButton : Button {     protected void OnGotFocus(EventArgs e)     {         this.Font = new Font(this.Font, FontStyle.Bold,         FontStyle.Underline);         this.ForeColor = Color.LightSalmon;     } }</pre>

## Module Summary

In this module, **Custom Controls**, you learnt about:

→ **Introduction to Custom Controls**

Custom controls are controls that can be reused across multiple applications. This saves the time of the developer to create the same kind of interface multiple times. User-validation and customized design are the key features of custom controls. The different types of custom controls are composite, extended, and custom.

→ **Creating Custom Controls**

Custom controls can be created by inheriting the Control class, or the UserControl class, or any Windows Forms existing controls. The properties of the custom controls can be read and set using the get and set accessors respectively. The += operator can be used to associate the events with the custom controls.

→ **Using Custom Controls**

Custom controls can be added to your application by right-clicking the toolbox, clicking the Choose Items option and browsing for the appropriate file. You can also programmatically add the custom control in your application by including the namespace in which the custom control is created.

→ **Creating and Using Composite Controls**

Composite controls allow you to create a graphical interface by using various existing Windows Forms controls. Composite controls can be created by inheriting the UserControl class. You can customize events either by creating a new custom event or overriding the base class event.

# ASK to LEARN

**Questions**  
*in your*  
**mind?**



are here to **HELP**

Post your questions in the **ASK to LEARN** section  
for solutions.

# Module - 13

## Crystal Reports

Welcome to the Module, **Crystal Reports**.

Crystal reports summarize data by generating interactive reports. It allows you to display reports in a graphical format by displaying them through pie diagrams, bar diagrams, and so on. The Crystal Report Creation Wizard allows you to easily generate the reports. The generated reports can also be customized at runtime and these reports can be exported into various formats.

In this Module, you will learn about:

- ➔ Crystal Reports
- ➔ Crystal Report Expert
- ➔ Runtime Customization
- ➔ Working with Crystal Reports

## 13.1 Crystal Reports

In this first lesson, **Crystal Reports**, you will learn to:

- Explain Crystal Reports and its use.
- List the features of Crystal Reports.

### 13.1.1 Crystal Reports

Consider a scenario of an automobile company that manufactures cars. The company wants to keep track of the production and sales of cars. The manager needs the production and sales details of previous months to view the progress of the company. Also, the manager wants to view the progress in a graphical format as it allows easy comparison. This is where crystal reports are useful.

Crystal Reports allows you to present it in a graphical format.

### 13.1.2 Uses of Crystal Report

Crystal Reports is a reporting tool that allows you to summarize a variety of data by generating interactive reports. Crystal reports allow you to use different graphs such as bar diagram or pie diagram to provide graphical impact to your reports. Some of the advantages of crystal reports are as follows:

- Simplifies the process of analyzing data by generation of graphs.
- Accelerates the process of computing complex calculation.
- Displays data easily based on the search criteria.
- Displays data easily using various charts and graphs.
- Formats data easily.

### 13.1.3 Features of Crystal Report

Crystal reports provide various features to ease the process of generating reports and to enhance the productivity. Some of the features are as follows:

#### → Data Access

Crystal reports allow you to easily access databases, files, application systems, and so on. Crystal reports have more than 35 data drivers that help in easy access of data from various data sources. It provides flexibility in connecting to the database by creating your own SQL commands or using query generation feature of Crystal Reports.

#### → Runtime Customization

Crystal Reports provides various options to customize your reports. It allows you to create interactive reports by customizing them at runtime as per your requirements.

#### → Interaction Between Report Viewer and Other controls

The Report Engine Object Model helps in providing interactivity between the report viewer control and other controls in the Windows Form page. The report viewer control allows you to display the reports on the form. This can be done by adding code to the source file of the Windows application.

#### → Reports as Web Services

Crystal Reports created in a project can be converted to Report Web Service. The Report Web Service is compiled to a .dll file and then, an Extensible Markup Language (XML) is generated. This XML file contains all the input parameters, public functions, and data types of the Report Web Service. The Web Service is invoked by the client through HTTP and XML. This allows you to transfer data between the Web Service and the client.

#### → Reusability

Crystal Reports help in accelerating the process of designing reports by storing report objects such as SQL commands, formulas, and bitmaps in a central library. This feature makes these objects reusable and easy to access across various reports.

#### → Security

Crystal Reports automatically updates itself with latest hot fixes and service packs thereby, keeping you updated with the latest versions. It also provides with extra security features such that a user needs to sign in only once.

### Knowledge Check 1

- Can you match the features of crystal reports against their corresponding descriptions?

Description		Features
(A)	Provides options for designing and sorting the database.	(1) Reusability
(B)	Facilitates access of the databases by using more than 35 data drivers.	(2) Security
(C)	Stores all object used in a report in a centralized library for easy access.	(3) Data Designing and Formatting
(D)	Allows the user to preview the report before generating it.	(4) Data Access
(E)	Updates itself to all the latest service packs and fixes.	(5) Viewing Reports

## 13.2 Crystal Report Expert

In this second lesson, **Crystal Report Expert**, you will learn to:

- List the steps of creating reports using Crystal Report Creation Wizard.
- Describe how to create a report using ADO.NET datasets.
- Identify the process of binding a report in an application.

### 13.2.1 Crystal Report Creation Wizard

Wizards are tools that assist you in carrying out a particular process easily. Windows Forms provides you with a Crystal Report Wizard that guides you in generating reports. The Report Wizard in Crystal Reports provides a step by step guide to generate reports. It allows you to present the report data into a tabular or matrix format. Figure 13.1 displays the Crystal Report Creation Wizard.

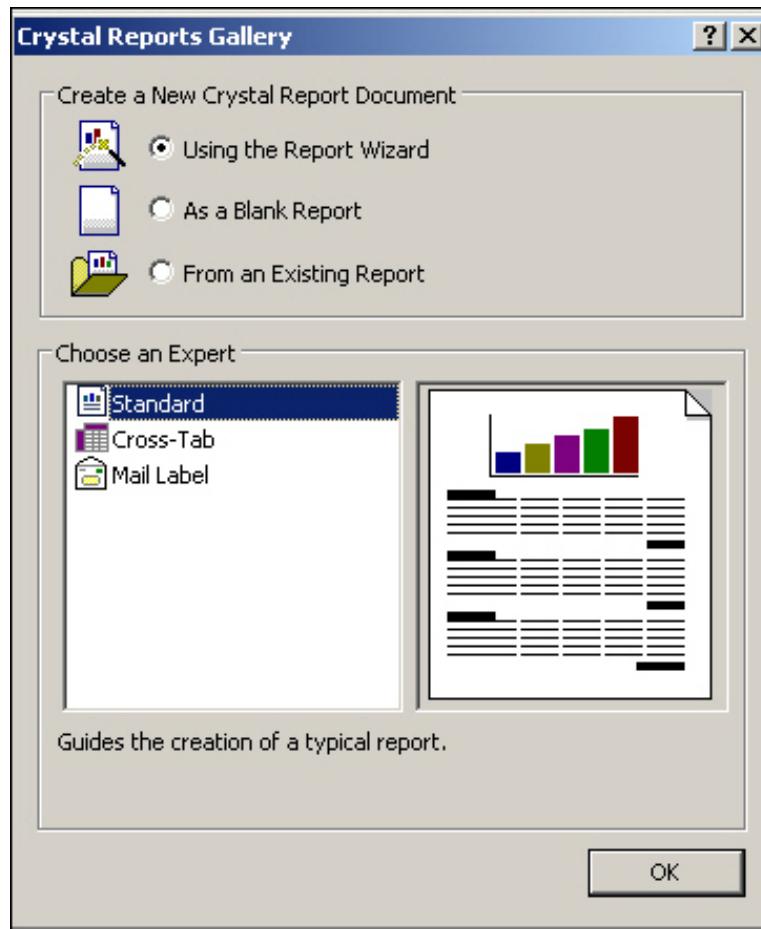


Figure 13.1: Crystal Report Wizard

The advantages of the Crystal Report Wizard are that it allows you to easily generate reports and it is user-friendly. The Crystal Report Creation Wizard provides various wizards that allow you to create different types of reports. The different wizards are listed as follows:

→ **Standard**

The Standard Report Creation Wizard guides you to choose a data source and link to the tables. It also guides you to sum up the totals, sort data, create charts, and select records. The wizard provides predefined layouts that can be applied to make your report more attractive.

→ **Cross-Tab**

The Cross-Tab Report Creation Wizard is useful when you want to display data as a Cross-Tab object. It also allows you to create and format the cross-tab itself.

→ **Mail Label**

The Mailing Labels Report Creation Wizard guides you to create a report that prints on a mailing label. It allows you to select a label type or define the layout for rows and columns.

**Note** - The Cross-Tab report appears as a grid from which you can retrieve the desired values. This format eases the process of comparing and identifying data.

You need to follow a series of steps to generate reports using Crystal Reports Creation Wizard. The steps for generating reports using Crystal Report Creation Wizard are as follows:

→ **Open the Crystal Report Creation Wizard**

You can open the Crystal Report Creation Wizard by clicking **Project** menu, selecting the Add New Item option, and selecting **Crystal Report**.

→ **Name the Report**

You can name the report by entering the name of the report in the **Name** text box.

→ **Select Report Document and Report Expert**

The report document allows you to use the report wizard, a blank report, or the existing report. The report expert allows you to choose the report layout. You can select the appropriate report document and a report expert from the **Crystal Reports Gallery** dialog box, which appears when the report file is added to the project.

→ **Select a Data Source**

You can select a data source by selecting a data source from the **Available Data Sources** section.

→ Choose a Report Type and Style

You can choose a report type and style of the report by selecting the desired type and style from the set of available styles in the **Available Styles** section.

**Note** - A Crystal Report application can also, be created by clicking New Project from File menu.

### 13.2.2 Creating Report Using ADO.NET

Crystal reports allow you to generate reports using ADO.NET DataSet. The DataSet can be created from different data sources. The steps for generating a report using ADO.NET DataSet are as follows:

1. Create an instance of a DataSet, which contains only the description of data and not the actual data.
2. Connect the report to the DataSet object.

From these steps, a report is created based on data given in the DataSet object. Since, the DataSet object is not populated with data, the actual data is not displayed. The data can be displayed by pushing the data into the DataSet object of the ADO.NET object model. Finally, you should bind the report object to the DataSet object.

### 13.2.3 Binding Reports to Application

Windows Forms allows you to bind reports to the application by using CrystalReportViewer control. This control allows you to view the reports. You can customize the reports by using the properties, methods, and events of CrystalReportViewer class. Figure 13.2 displays the Binding Reports.

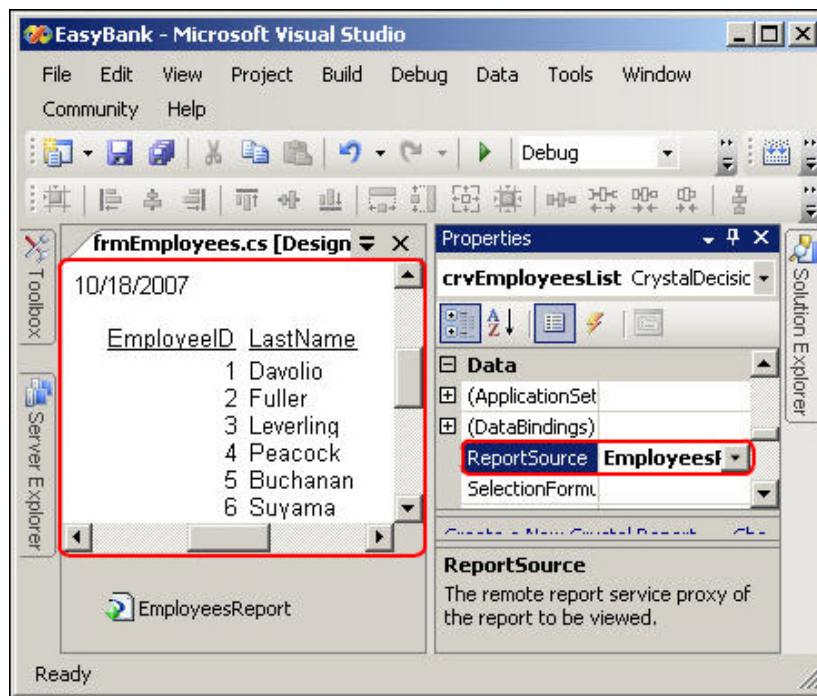


Figure 13.2: Binding Reports

Once the report is created, you should bind it to your application. The `SetDataSource` property of the `ReportDocument` class binds the `DataSet` object to the report. This helps in displaying the required records in the report from the `DataSet`. The `ReportSource` property of the `CrystalReportViewer` class allows you to specify which report the `CrystalReportViewer` control should be bound to.

For example, you may want to display a particular report as a result of some event such as a button click. Binding the report allows you to display the appropriate report. The following syntax demonstrates how the dataset object is bound to the report using the `SetDataSource` property.

#### Syntax:

```
<ReportDocumentObject>.SetDataSource (<DataSetObject>);
```

where,

`ReportDocumentObject`: Is an object of the `ReportDocument` class.

`DataSetObject`: Is an object of the `DataSet` class.

The following syntax demonstrates how to specify the report to be bound to the `CrystalReportViewer` control using the `ReportSource` property.

#### Syntax:

```
<CrystalReportViewerObject>.ReportSource = <ReportDocumentObject>;
```

`CrystalReportViewerObject`: Is an object of the `CrystalReportViewer` class.

`ReportDocumentObject`: Is an object of the `ReportDocument` class.

The following code generates a report using datasets and binds the report to the application.

#### Code Snippet:

```
...
CrystalReportViewer crvEmployeeList = new CrystalReportViewer();
SqlDataAdapter sqldaEmployees = new SqlDataAdapter("SELECT * FROM Employees", sqlconEmployees);
ReportDocument rdEmployeesReport = new ReportDocument();
rdEmployeesReport.Load(@"D:\WindowsForms\EmployeesReport.rpt");
DataSet dsetEmployees = new DataSet();
sqldaEmployees.Fill(dsetEmployees, "Employees");
rdEmployeesReport.SetDataSource(dsetEmployees);
crvEmployeeList.ReportSource = rdEmployeesReport;
```

The code creates an instance of `SqlDataAdapter` class and retrieves the data from the `Employees` table. The `ReportDocument` class represents a report and allows you to work with the report. The `Load()` method loads the `EmployeesReport.rpt` report. A dataset is created and the `Fill()` method fills the dataset with the records. The `SetDataSource()` method binds the dataset to the report. The `ReportSource` property indicates the data source for the report, which is an instance of the `ReportDocument` class. This class holds the reference to the dataset.

## Knowledge Check 2

- Which of the following statements about crystal reports are true?

(A)	The ReportSource property helps in specifying which report the CrystalReportViewer control should be bound to.
(B)	Crystal Reports allows reports to be generated only using ADO.NET datasets.
(C)	The Crystal Report Creation Wizard provides a step by step guide to generate reports.
(D)	The CrystalReportViewer control binds the report to the application.
(E)	The report document allows you to choose the layout for reports.

### 13.3 Runtime Customization

In this third lesson, **Runtime Customization**, you will learn to:

- Explain the process of customizing reports.
- Describe how to present the report data.
- Explain the process of sorting fields at runtime.

#### 13.3.1 Customizing Reports at Runtime

You can customize a report at runtime in Windows Forms. This is done by adding the required code to the application using the Report Engine Object Model or by using the class for the Windows Forms Viewer. The Report Engine Object Model provides access to the Crystal Reports engine. This engine provides a set of properties and methods to process, save, export, and print reports. This model is implemented by using the ReportDocument control, which allows you to process, save, and print reports. The other control that can be used to customize reports at runtime is the CrystalReportViewer control, which is the Windows Forms Viewer control.

Customizing the report includes changing the appearance, font, and view of the report. It also includes handling events and taking inputs from the user, once the event is generated. Suitable events can be raised when the user is navigating, searching, or entering a value.

#### 13.3.2 SelectionFormula Property

You can use the selection formulas to filter and display the required records in a report. A selection formula can also be used in grouping and sorting fields of a table displayed in the report.

The SelectionFormula property of CrystalReportViewer class is used to specify a selection formula. Consider an example, where you want to retrieve records of employees with **Fuller** as the last name, and not residing in **London** city. You can achieve this by using a selection formula. The RefreshReport() method can be used to refresh the report after making some changes.

### 13.3.3 Parameter Fields

Selection formula is used to customize and display reports as per user requirements by selecting the particular field or value. Finally, a report is generated based on the parameter values.

A parameter is like getting an answer to a question, from the user, without which the report will not be generated. To take values from the user at runtime, parameter fields are created programmatically. These parameter fields can be used for various purposes, which are as follows:

- Passing a parameter on a database field allows the user to specify values for that field to filter the records.
- Applying conditional formatting to the fields in reports, which the user has specified.
- Specifying the order and the field name according to which data is sorted.
- Grouping the required fields, which the user has specified.

The following code demonstrates creating parameters programmatically.

#### Code Snippet:

```
...
ParameterFields paramFields = new ParameterFields();
ParameterField pfCity = new ParameterField();
pfCity.ParameterFieldName = "SetCity";
ParameterDiscreteValue discreteVal = new ParameterDiscreteValue();
discreteVal.Value = "London";
pfCity.CurrentValues.Add(discreteVal);
paramFields.Add(pfCity);
crvEmployeeList.ParameterFieldInfo = paramFields;
```

The code uses the object of the `ParameterFields` class, which represents the collection of parameters. An instance of the `ParameterField` class is created, which is used to set the parameter field as `SetCity`. This is done using the `ParameterFieldName` property. Here, `SetCity` refers to the parameter field which is based on `City` field of `Employees` table. An instance of the `ParameterDiscreteValue` class is created to specify the parameter value. This is done using the `Value` property. This value is added to the parameter using the `Add()` method. The parameter is added to the `ParameterFields` collection using the `Add()` method. The `ParameterFieldInfo` property of the `CrystalReportViewer` control is assigned the `ParameterField` object to indicate the parameter field and value.

### 13.3.4 Sorting Fields at Runtime

Runtime customization in reports includes sorting fields based on user input. To do so, you can use sort fields to display the records in a particular sorted order. You can take the field name from the user based on which sorting will take place.

In addition, the direction in which the data will be sorted can also be specified. These directions can be ascending, descending, top N, and bottom N order.

Consider a scenario where a user has to sort all the names of customers according to their country name. Therefore, by setting the sort field name to the required country name, you can display the names of customers residing in that country.

The following code demonstrates how to sort the fields in a report.

#### Code Snippet:

```
SqlDataAdapter sqldaEmployees = new SqlDataAdapter("SELECT * FROM Employees", sqlcon);
ReportDocument rdEmployeesReport = new ReportDocument();
rdEmployeesReport.Load(@"D:\WindowsForms\EmployeesReport.rpt");
DataSet dsetEmployees = new DataSet();
sqldaEmployees.Fill(dsetEmployees, "Employees");
rdEmployeesReport.SetDataSource(dsetEmployees);
CrystalReportViewer crvEmployeeList = new CrystalReportViewer();
crvEmployeeList.ReportSource = rdEmployeesReport;
FieldDefinition fdCity;
fdCity = rdEmployeesReport.Database.Tables[0].Fields["City"];
rdEmployeesReport.DataDefinition.SortFields[0].Field = fdCity;
rdEmployeesReport.DataDefinition.SortFields[0].SortDirection =
CrystalDecisions.Shared.SortDirection.DescendingOrder;
```

The code fills the report with the records of the `Employees` table. An instance of `FieldDefinition` class is created to hold the field name of the table. It refers to the `City` field of the table using the `Fields` property. This is done using `Field` property of the `ReportDocument` class. The `SortDirection` property is assigned the `DescendingOrder` value defined in the `SortDirection` enumeration. This will display the records in the descending order of the values in the `City` field.

## Knowledge Check 3

1. You want to display the records of employees from London whose salary is greater than 5000 dollars? Which one of the following codes will help you to achieve this?

(A)	<pre>SqlDataAdapter sqldaEmployees = new SqlDataAdapter("SELECT * FROM Employees");  ReportDocument rdEmployeesReport = new ReportDocument();  rdEmployeesReport.Load(@"D:\WindowsForms\EmployeesReport.rpt");  DataSet dsetEmployees = new DataSet();  sqldaEmployees.Fill(dsetEmployees, "Employees");  rdEmployeesReport.SetDataSource(dsetEmployees);  CrystalReportViewer crvEmployeeList = new CrystalReportViewer();  crvEmployeeList.ReportSource = rdEmployeesReport;  crvEmployeeList.SelectionFormula = {Employees.Salary} &gt;= 5000 AND {Employees.City} = London;</pre>
(B)	<pre>SqlDataAdapter sqldaEmployees = new SqlDataAdapter("SELECT * FROM Employees", sqlconEmployees);  ReportDocument rdEmployeesReport = new ReportDocument();  rdEmployeesReport.Load(@"D:\WindowsForms\EmployeesReport.rpt");  DataSet dsetEmployees = new DataSet();  sqldaEmployees.Fill(dsetEmployees, "Employees");  rdEmployeesReport.SetDataSource(dsetEmployees);  CrystalReportViewer crvEmployeeList = new CrystalReportViewer();  crvEmployeeList.ReportSource = rdEmployeesReport;  crvEmployeeList.SelectionFormula = "{Employees.Salary} &gt;= 5000 AND {Employees.City} = \"London\"";</pre>

(C)	<pre>SqlDataAdapter sqldaEmployees = new SqlDataAdapter("SELECT * FROM Employees");  ReportDocument rdEmployeesReport = new ReportDocument();  rdEmployeesReport.Load(@"D:\WindowsForms\EmployeesReport.rpt");  DataSet dsetEmployees = new DataSet();  sqldaEmployees.Fill(dsetEmployees, "Employees");  rdEmployeesReport.SetDataSource(dsetEmployees);  CrystalReportViewer crvEmployeeList = new CrystalReportViewer();  crvEmployeeList.ReportSource(rdEmployeesReport);  crvEmployeeList.SelectionFormula = "{Employees.Salary} &gt;= 5000 AND {Employees.City} = \"London\"";</pre>
(D)	<pre>SqlDataAdapter sqldaEmployees = new SqlDataAdapter("SELECT * FROM Employees", sqlconEmployees);  ReportDocument rdEmployeesReport = new ReportDocument();  rdEmployeesReport.Load(@"D:\WindowsForms\EmployeesReport.rpt ");  DataSet dsetEmployees = new DataSet();  sqldaEmployees.Fill(dsetEmployees, "Employees");  rdEmployeesReport.SetDataSource = dsetEmployees;  CrystalReportViewer crvEmployeeList = new CrystalReportViewer();  crvEmployeeList.ReportSource(sqldaEmployees);  crvEmployeeList.SelectionFormula = "{Employees.Salary} &gt;= 5000 AND {Employees.City} = \"London\"";</pre>

2. Which of these statements about customizing runtime reports are true?

(A)	Parameter fields help displaying the reports based on a criterion taken from the user.
(B)	Sorting fields help in sorting the data only in ascending and descending orders.
(C)	Selection formulas support passing formula to the SelectionFormulae property of the CrystalReportViewer class.
(D)	Selection formulas help to filter or group the data displayed in the report.
(E)	The RefreshReport() method of the CrystalReportViewer class refreshes the report after it has been changed.

## 13.4 Working with Crystal Reports

In this last lesson, **Working with Crystal Reports**, you will learn to:

- Explain the report viewer controls.
- Describe the process of exporting a report.

### 13.4.1 Report Viewer Controls

Microsoft Visual Studio 2005 has introduced various controls to view reports. The `CrystalReportViewer` control can be used to view Windows Forms reports. The control is used to display the report page wise. You can move between pages displayed in the control.

To use the control, the control should be dragged on to the Windows Form. Then, the `ReportSource` property must be used to assign the source for the report.

### 13.4.2 CrystalReportViewer Control

The `CrystalReportViewer` control supports data binding and displays the designed report on a form. Some of the important features of this control are as follows:

#### → Smart Tasks

The smart task is a panel present in the upper right corner of the control. This panel consists of check boxes and combo boxes, which help in selecting the connectivity and various options of the control. This reduces the code written to use the options provided by the smart task panel.

#### → ToolTip Disable Option

The `ToolTip` feature is made optional for crystal reports. This is implemented by using the `EnableTooltips` property of the `CrystalReportViewer` class. Setting this property to `False` will disable the `ToolTip` information globally.

#### → Error Event

The `Error` event of `CrystalReportViewer` class is a new event defined to catch all the exceptions. In the previous versions, the exceptions were caught by writing code for each exception. If any code is omitted, the report processing was disrupted. The event displays error messages without any disruption.

#### → Multilingual Client Support

The `CrystalReportViewer` control supports multiple languages using the dynamic localization feature. This feature allows displaying the tooltips and content of the control in various languages.

### → Report Selection from the Properties Window

The `CrystalReportViewer` control allows you to select reports directly by dragging the `ReportDocument` component on the application page and then, binding it with the control. The `ReportSource` property allows you to display a combo box, which when clicked provides all the details of reports contained within the project.

### → Design Time Preview

Crystal reports allow you to preview reports at design time. This is done by adding the report to the `ReportSource` property so as to preview the report in Windows Form.

## 13.4.3 CrystalReportViewer Class

The `CrystalReportViewer` control is used for displaying reports in an application. The `CrystalReportViewer` class is used to create the control. It exists in `CrystalDecisions.Windows.Forms` namespace.

### → Properties

The properties of the `CrystalReportViewer` class allow you to get the collection of parameter fields and the report source. Table 13.1 lists the most commonly used properties of `CrystalReportViewer` class.

Property	Description
<code>ActiveViewIndex</code>	Specifies or retrieves the index of the view, which is active in the viewer control.
<code>DisplayGroupTree</code>	Specifies or retrieves whether the tree view is displayed or hidden.
<code>ParameterFieldInfo</code>	Specifies or retrieves the collection of parameter fields.
<code>ReportSource</code>	Specifies or retrieves the report to be bound to the control.
<code>SelectionFormula</code>	Specifies or retrieves the record selection formula for the report.
<code>ViewCount</code>	Retrieves the number of views in the viewer control.

Table 13.1: `CrystalReportViewer` Properties

### → Methods

The methods of `CrystalReportViewer` class are used to export the report which is displayed in `CrystalReportViewer` control. Table 13.2 lists the most commonly used methods of `CrystalReportViewer` class.

Method	Description
<code>ExportReport</code>	Exports the report, which is displayed in the viewer control.
<code>GetCurrentPageNumber</code>	Retrieves the current page number of the report.
<code>PrintReport</code>	Prints the report, which is displayed in the viewer control.

Method	Description
RefreshReport	Refreshes the report, which is displayed in the viewer control.
SearchForText	Searches for a specified text in the report.
ShowFirstPage	Displays the first page of the report.
ShowLastPage	Displays the last page of the report.
ShowNthPage	Displays a particular page of the report.

Table 13.2: CrystalReportViewer Methods

## → Events

Table 13.3 lists the most commonly used events of CrystalReportViewer class.

Event	Description
Error	Occurs when any error arises in the viewer control.
Navigate	Occurs when the user navigates through the report.
ReportRefresh	Occurs when data is refreshed in the report.
Search	Occurs when a text is searched in the report.

Table 13.3: CrystalReportViewer Events

The following code binds the dataset to the report to display the records from the dataset.

### Code Snippet:

```

. . .
SqlDataAdapter sqldaStudents = new SqlDataAdapter("SELECT * FROM
Students", sqlconStudents);
ReportDocument rdStudentsReport = new ReportDocument();
rdStudentsReport.Load(@"D:\WindowsForms\StudentsReport.rpt");
DataSet dsetStudents = new DataSet();
sqldaStudents.Fill(dsetStudents, "Students");
rdStudentsReport.SetDataSource(dsetStudents);
CrystalReportViewer crvStudentsList = new CrystalReportViewer();
crvStudentsList.ReportSource = rdStudentsReport;
crvStudentsList.SelectionFormula = "{Students.City} = \"London\"";
crvStudentsList.RefreshReport();
Controls.Add(crvStudentsList);
. . .
private void crvStudentsList_Error(object source, CrystalDecisions.
Windows.Forms.ExceptionEventArgs e)
{
    MessageBox.Show("Error : " + e.ToString());
}

```

The code binds the dataset to the report, which will display the records from the Students table. This is done using the object of the ReportDocument class and the SetDataSource property.

The `ReportSource` property of the `CrystalReportViewer` class property is assigned the object of the `ReportDocument` class, which contains the `.rpt` file. The `SelectionFormula` property is used to filter the records based on the `City` field. The records of all the students residing in London city will be displayed. The `RefreshReport()` method refreshes the report after filtering the records. If any error takes place while processing or working on the report, the `Error` event is raised. When this event is raised, a message box is displayed indicating the error.

#### 13.4.4 ReportDocument Object Model

The `ReportDocument` object model is a powerful and flexible model used to handle the report at runtime. It provides many built-in classes in different namespaces that are used to create, update, and save the report at runtime. One of the most important classes in this model is the `ReportDocument` class. The data members of this class are used to define, load, and print reports. It exists in the `CrystalDecisions.CrystalReports.Engine` namespace.

Consider a scenario where a report displays the total orders for the last six months. This report has been generated using bar charts. This report needs to be sent to the senior management in a `.pdf` format through a mail. In this case, you can use the export feature of Crystal Reports to create such formats.

Crystal Reports helps in exporting the generated report in different file formats. The file formats in which a report can be exported are as follows:

- Adobe Acrobat (.pdf)
- Crystal Reports for Visual Studio .NET (.rpt)
- HTML 3.2 and 4.0 (.html)
- Microsoft Excel (.xls)
- Microsoft Rich Text (.rtf)
- Microsoft Word (.doc)

#### 13.4.5 Exporting a Report

You can export a crystal report using the **Export** button on the toolbar of `CrystalReportViewer` control. Alternately, you can export the report by invoking the `ExportReport()` method of `CrystalReportViewer` class. This method invokes the **Export Report** dialog box. This dialog box will ask the user the location to save the report and the report format. The following code exports a report by using the `ExportReport()` method.

##### Code Snippet:

```
SqlDataAdapter sqldaStudents = new SqlDataAdapter("SELECT * FROM Students",
sqlconStudents);
```

```

ReportDocument rdStudentsReport = new ReportDocument();
rdStudentsReport.Load(@"D:\WindowsForms\StudentsReport.rpt");
DataSet dsetStudents = new DataSet();
sqlDAStudents.Fill(dsetStudents, "Students");
rdStudentsReport.SetDataSource(dsetStudents);
CrystalReportViewer crvStudentsList = new CrystalReportViewer();
crvStudentsList.ReportSource = rdStudentsReport;
crvStudentsList.ExportReport();

```

The code binds the dataset to the report, which will display the records from the Students table. This is done using the object of the ReportDocument class and the SetDataSource method. The ReportSource property of CrystalReportViewer class is assigned an object of the ReportDocument class, which contains the .rpt file. The ExportReport() method exports the specified report in the specified format.

## Knowledge Check 4

- Which of these statements about the report viewer controls are true?

(A)	The ReportDocument class exists in CrystalDecisions.CrystalReports.Engine namespace.
(B)	The CrystalReportViewer control supports the Errors event to track exceptions.
(C)	A crystal report supports the .doc format for exporting the report.
(D)	The CrystalReportViewer class provides the Export() method to export a crystal report.
(E)	The ViewCount property of CrystalReportViewer class retrieves the number of views for the viewer control.

- Can you match the properties, methods, and events of CrystalReportViewer class against their corresponding descriptions?

Description		Property, Method, and Event
(A)	This property identifies the formula for filtering records in a report.	(1) ExportReport
(B)	This method exports the report displayed in the control.	(2) SelectionFormula
(C)	This method prints the report displayed in the control.	(3) ReportSource
(D)	This event occurs when the user navigates through a report.	(4) PrintReport
(E)	Updates itself to all the latest service packs and fixes.	(5) Viewing Reports

## Module Summary

In this module, **Crystal Reports**, you learnt about:

→ **Crystal Reports**

Crystal Reports allows you to summarize data by creating interactive reports. It provides various advantages such as computing complex calculations, displaying data in various graphs, and so on. It provides features such as better security, more control over viewing the reports, and so on.

→ **Crystal Reports Expert**

The Crystal Report Creation Wizard allows you to easily generate reports by providing step by step guidance. You can also generate reports that use ADO.NET datasets as the data source. Once the report is generated, you can bind it with the application by using the CrystalReportViewer control.

→ **Runtime Customization**

Crystal Reports provides various ways of customizing the reports at runtime. The SelectionFormula property of the CrystalReportViewer class can be used to sort, filter and group records. The SortFields property allows you to sort the records in a specified sort direction.

→ **Working with Crystal Reports**

The CrystalReportViewer control helps in displaying a report. The CrystalReportViewer class uses various properties to bind the report to the control. The reports generated using Crystal Reports can be exported to various formats, such as .PDF, .DOC, .HTML, and so on.

# Module - 14

## Printing and Adding Help

Welcome to the Module, **Printing and Adding Help**.

You can print documents through a Windows Forms application using the various classes defined in System.Drawing.Printing namespace. A Windows Forms application needs to be distributed to various users. To make them aware about the features and functionalities of the application, a help file must be provided by the application.

In this Module, you will learn about:

- ➔ Printing in Windows Forms
- ➔ Help in Windows Forms
- ➔ Help Components and Controls

## 14.1 Printing in Windows Forms

In this first lesson, **Printing in Windows Forms**, you will learn to:

- Explain printing and `System.Drawing.Printing` namespace.
- List and explain the basic printing classes.
- List and explain the properties, methods, and events of printing classes.
- Describe `PrinterSettings` and `PageSettings` classes.
- Describe dialog boxes and component controls for printing.

### 14.1.1 Purpose

Consider a scenario where the Order Details application of a company is used to generate monthly order report. This report helps them to analyze the total number of orders for a given month to identify their sales level.

They need to submit this report to their top management at the end of every month. To achieve this, they need to take a printout of the report. If the application does not provide this functionality, the user will be disappointed. This requires the application to provide the printing functionality, to meet the requirements. This can be achieved by using built-in classes of the `System.Drawing.Printing` namespace of Microsoft Visual Studio 2005.

Printing is the commonly performed task in business world. A printout is preferred for reading facts and information, as unlike a soft copy, it can be carried along anywhere.

#### → `System.Drawing.Printing Namespace`

The `System.Drawing.Printing` namespace defines various classes, which provide printing services. You can use these services in a Windows Forms application by using the data members of these classes. Some of these services include setting the printer, specifying the page layout and its margins, canceling the print job, and previewing the document before printing. These tasks are similar to the tasks performed by the **Print** dialog box in Microsoft Word.

### 14.1.2 Printing Classes

The `System.Drawing.Printing` namespace defines several classes that can be used for printing. It also helps in managing the printers and documents sent to the printing queue.

The basic classes for printing, defined in the namespace are as follows:

→ **PrintDocument Class**

The object of the `PrintDocument` class is used to send the output from a Windows Forms application, to the printer.

→ **PrintController Class**

The `PrintController` class is used to control the printing process. Its purpose is to let the printer know how the document is to be printed. This is done using the methods of this class.

→ **PrintEventArgs Class**

The `PrintEventArgs` class provides information about the `BeginPrint` and `EndPrint` events of `PrintDocument` class. The `BeginPrint` event occurs when the `Print()` method is called and before the first page of the document has been printed. The `EndPrint` event occurs when the last page of the document has printed.

→ **PrintPageEventArgs Class**

The `PrintPageEventArgs` class makes the information available for the `PrintPage` event. The class provides information such as printer settings and the total area of the page. The `PrintPage` event occurs when there is a requirement to print the current page.

The `PrintDocument` and `PrintPageEventArgs` classes are used for printing documents. To print the document, you need an object of the `PrintDocument` class. Then, invoke the `Print()` method of the `PrintDocument` class. This will raise the `PrintPage` event which takes an object of the `PrintPageEventArgs` class. This object contains all printing information. However, there are some basic steps that must be performed for printing a document. They are as follows:

→ **Specifying the Printing Logic**

The first step is to invoke the `Print()` method. This method will first raise the `BeginPrint` event, then, the `PrintPage` and `EndPrint` events. In the `PrintPage` event handler, you use the information in the handler arguments for printing. These arguments contain the Graphics for the printer, page settings and bounds, and margins size.

→ **Defining Print Document**

You can write a common logic to be used across Windows-based applications. For writing this logic, you should create a new class that inherits the `PrintDocument` class. Then, you must handle the `PrintPage` event by overriding the `OnPrintPage()` method.

### → Choosing a Printer

The next step is to choose a printer. To do so, the `PrintDocument` object must be passed to the `PrintDialog`. The printer knows what to print from the `Graphics` object, which is assigned to the `Graphics` property of the `PrintPageEventArgs` class. The object is passed to the printer by `Print()` method.

### → Choosing Page Settings

You should allow the user to choose page settings such as the orientation and the margin size. To do so, you must create an object of `PageSettings` class and pass it to the `PageSetupDialog` class.

**Note** - The `BeginPrint` event can be raised to initialize the fonts and create file streams used during the printing process. The code to do so can be added in its event handler. The `EndPrint` event can be raised to free the resources such as file streams after the printing is completed.

### 14.1.3 *PrintDocument Component*

The `PrintDocument` component provides the ability to print documents from a Windows Forms application. It provides various properties used to specify the documents to be printed. The component can be used to print any files such as .txt, .pdf, or .doc. The component appears in the tray, which lies at the bottom of Windows Forms designer. The `PrintDocument` class is used to set the properties for the component. Two main purposes of this component are as follows:

- **Simple Print Jobs:** This includes printing a text file or any single file. This job follows the normal steps of printing the file.
- **Complex Print Jobs:** This includes reusing the printing logic. Here, you should create a new component derived from the `PrintDocument` component. Then, you must override the `PrintPage` event.

The `PrintDocument` class contains various properties, methods, and events used to manage the printing activity. They help in identifying the page settings, printer settings, and provide a scope for specifying and executing the printing logic.

### → Properties

The properties of the `PrintDocument` class allow you to specify the default page and printer for printing. Table 14.1 lists the most commonly used properties of `PrintDocument` class.

Property	Description
<code>DefaultPageSettings</code>	Specifies or retrieves page settings, which are the default settings for every page that is to be printed.
<code>DocumentName</code>	Specifies or retrieves the name of the document to be displayed while printing the document.

Property	Description
PrintController	Specifies or retrieves the print controller, which helps the printing process.
PrinterSettings	Specifies or retrieves the printer that prints the document.

Table 14.1: PrintDocument Properties

## → Methods

The methods of `PrintDocument` class allow you to generate the `BeginPrint`, `PrintPage`, and `EndPrint` events. Table 14.2 lists the most commonly used methods of `PrintDocument` class.

Method	Description
<code>OnBeginPrint</code>	Generates the <code>BeginPrint</code> event. It is called before the printing of the first page of the document has started and after the <code>Print()</code> method is invoked.
<code>OnEndPrint</code>	Generates the <code>EndPrint</code> event. It is called once the last page of the document is printed.
<code>OnPrintPage</code>	Generates the <code>PrintPage</code> event. It is called before the printing of a page starts.
<code>Print</code>	Begins the printing process.
<code>ToString</code>	Provides information about the print document in string form.

Table 14.2: PrintDocument Methods

## → Events

Table 14.3 lists the most commonly used events of `PrintDocument` class.

Event	Description
<code>BeginPrint</code>	Occurs when the <code>Print()</code> method is invoked and before the first page of the document prints.
<code>EndPrint</code>	Occurs when the last page of the document has printed.
<code>PrintPage</code>	Occurs when there is a need to print the current page as an output.

Table 14.3: PrintDocument Events

The following code demonstrates how to print a value stored in the `string` variable using the `PrintDocument` class.

### Code Snippet:

```
Button btnPrint = new Button();
PrintDocument printDoc = new PrintDocument();
Font printFont;

private void printDoc_PrintPage(object sender, PrintPageEventArgs e)
{
```

```

        string message = "Welcome to Windows Forms Programming";
        e.Graphics.DrawString(message, SystemFonts.DefaultFont,
            Brushes.Black, 5, 10, new StringFormat());
    }
    private void btnPrint_Click(object sender, EventArgs e)
    {
        printFont = new Font("Verdana", 12);
        printDoc.DocumentName = "Welcome";
        printDoc.Print();
    }
}

```

The code adds a button on the form to initiate the printing task. When the user clicks the button on the form, the `Click` event is raised. When the event is raised, the `DocumentName` property is used to specify the document name that is to be displayed in the printer queue. Now, the `Print()` method is invoked that raises the `PrintPage` event. When this event is raised, the printing process is initiated by executing its corresponding event handler. In this handler, the `Graphics` property of the `PrintPageEventArgs` class is used to hold the graphic object, which is a string. This string is passed to the printer for printing using the information stored in `PrintPageEventArgs` object.

#### → Properties and Methods of `PrintController` Class

The `PrintController` class regulates the printing process of a `PrintDocument`. This class is an abstract class. The functionalities of this class are implemented in its derived classes. The derived classes are `StandardPrintController`, `PreviewPrintController`, and `PrintControllerWithStatusDialog`.

The `StandardPrintController` class is used to print the document. The `PreviewPrintController` class is used to generate the preview of the document before printing. The `PrintControllerWithStatusDialog` class provides a dialog box displaying the printing status. Table 14.4 lists the most commonly used properties and methods of the `PrintController` class.

Name	Description
<code>IsPreview</code>	This property retrieves a value that indicates whether the <code>PrintController</code> can be used for print preview.
<code>OnEndPage</code>	This method completes the control sequence, which identifies when to print a page and in what manner.
<code>OnEndPrint</code>	This method completes the control sequence, which identifies when to print a document and in what manner.
<code>OnStartPage</code>	This method begins the control sequence, which identifies when to print a page and in what manner.
<code>OnStartPrint</code>	This method begins the control sequence, which identifies when to print a document and in what manner.

Table 14.4: `PrintController` Properties and Methods

The following code demonstrates how to use the `IsPreview` property of the `PrintController` class.

#### Code Snippet:

```
PrintDocument printDoc = new PrintDocument();
printDoc.DocumentName = "My Document";
if (!printDoc.PrintController.IsPreview)
{
    printDoc.Print();
}
// Code to handle the PrintPage event
```

The code uses the `IsPreview` property to check whether the document can be previewed before printing. If the property returns `true`, the document preview will display the document as it will appear after printing. If it returns `false`, the document is sent for printing using the `Print()` method.

#### → Properties of `PrintEventArgs` Class

The `PrintEventArgs` class defines various properties indicating the kind of print operation and whether the execution of an event is to be cancelled. Table 14.5 lists the most commonly used properties of `PrintEventArgs` class.

Property	Description
<code>Cancel</code>	Specifies or retrieves a value that indicates whether the event should be cancelled.
<code>PrintAction</code>	Retrieves a value that indicates the type of print operation that is currently occurring.

Table 14.5: `PrintEventArgs` Properties

#### → Properties of `PrintPageEventArgs` Class

The `PrintPageEventArgs` class provides various properties used to get the object of the `Graphics` class and the page settings. Table 14.6 lists the most commonly used properties of `PrintPageEventArgs` class.

Property	Description
<code>Cancel</code>	Retrieves a value that indicates whether the current print job is to be canceled.
<code>Graphics</code>	Retrieves the object of the <code>Graphics</code> class used to paint the page.
<code>HasMorePages</code>	Specifies or retrieves a value that indicates whether an extra page is to be printed.
<code>PageBounds</code>	Retrieves the rectangular area, which is the total area of the page.
<code>PageSettings</code>	Retrieves the page settings for the active page.

Table 14.6: `PrintPageEventArgs` Members

The following code demonstrates how to print a rectangle using the `PrintDocument` and `PrintPageEventArgs` classes.

**Code Snippet:**

```
Button btnPrint = new Button();
PrintDocument printDoc = new PrintDocument();
private void printDoc_PrintPage(object sender, PrintPageEventArgs e)
{
    e.PageSettings.Color = true;
    e.Graphics.DrawRectangle(Pen.Blue, 50, 50, 100, 100);
}
private void btnPrint_Click(object sender, EventArgs e)
{
    printDoc.DocumentName = "My Document";
    printDoc.Print();
}
```

The code creates an instance of the `Button` class. When the user clicks the button on the form, the `Click` event is raised. When the event is raised, the `DocumentName` property is used to specify the document name to be displayed in the printer queue. Now, the `Print()` method is invoked that raises the `PrintPage` event. When this event is raised, the printing process is initiated by executing its corresponding event handler. In this handler, the `Graphics` property of the `PrintPageEventArgs` class is used to hold the graphic object, which is a rectangle. The rectangle is printed in blue color using the information stored in `PrintPageEventArgs` object.

#### 14.1.4 PrinterSettings Class

The `PrinterSettings` class specifies how a document is printed, including the printer used to print the document. You can access the object of this class by using the `PrintDocument.PrinterSettings` or `PageSettings.PrinterSettings` properties to modify printer settings.

→ **Properties**

The properties of `PrinterSettings` class are used to retrieve the number of copies to be printed and the page settings. Table 14.7 lists the most commonly used properties of `PrinterSettings` class.

Property	Description
Copies	Specifies or retrieves the total number of copies of the document to print.
DefaultPageSettings	Retrieves the default page settings for the current printer.
InstalledPrinters	Retrieves the names of all printers, which are installed on the computer.
PrinterName	Specifies or retrieves the name of the printer, which is to be used.
PrinterResolutions	Retrieves all the resolutions, which the current printer supports.

Property	Description
PrintFileName	Specifies or retrieves the name of the file, while printing to a file.
ToPage	Specifies or retrieves the last page number to print.

Table 14.7: PrinterSettings Properties

## → Methods

The methods of `PrinterSettings` class can be used to return a value that indicates whether printing an image file is supported by the printer. Table 14.8 lists the most commonly used methods of `PrinterSettings` class.

Method	Description
CreateMeasurementGraphics	Returns an object of the <code>Graphics</code> class, which contains the information of the printer.
IsDirectPrintingSupported	Returns a value that indicates whether printing an image file is supported by the printer.
ToString	Provides the information concerning the <code>PrinterSettings</code> in string form.

Table 14.8: PrinterSettings Methods

The following code demonstrates how to set the printer, the number of copies and pages for printing using the `PrinterSettings` class.

### Code Snippet:

```

...
PrintDocument printDoc = new PrintDocument();
printDoc.DocumentName = "My Document";
printDoc.PrinterSettings.PrinterName = "HPLaserJet";
printDoc.PrinterSettings.ToPage = 4;
printDoc.PrinterSettings.Copies = 1;
if (printDoc.PrinterSettings.IsValid)
{
    printDoc.Print();
}
// Code for handling the PrintPage event

```

The code uses the `PrinterName` property of the `PrinterSettings` class to identify the printer for printing. The `ToPage` property is used to specify the page number of the document, which will be the last page to be printed. The value assigned to this property is 4, which means only the first four pages will be printed. The `Copies` property is used to set the number of copies for printing. The value of the property is set to 1, which means that only one copy of the document will be printed. The `IsValid` property is used to check whether the printer settings are valid for printing the document. If true, the `Print()` method of the `PrintDocument` class is invoked.

### → Properties and Methods of `PageSettings` Class

The `PageSettings` class is used to specify the page settings such as layouts, margins, and size for a single, printed page. This is done using the properties and methods of `PageSettings` class.

### → Properties

The properties of the `PageSettings` class are used to specify or retrieve the page size and orientation. Table 14.9 lists the most commonly used properties of `PageSettings` class.

Property	Description
Bounds	Retrieves the page size by considering the orientation of the page specified using the <code>Landscape</code> property.
Landscape	Specifies or retrieves a value that indicates whether the page is to be printed in landscape or portrait orientation.
Margins	Specifies or retrieves the current page margins.
PaperSize	Specifies or retrieves the size of the paper for the current page.
PrintableArea	Retrieves the boundaries of the printable region of the page for the printer.
PrinterSettings	Specifies or retrieves the settings of the printer linked with the page.

Table 14.9: `PageSettings` Properties

### → Methods

The methods of the `PageSettings` class are used to create a copy of the specified `PageSettings`. Table 14.10 lists the most commonly used methods of `PageSettings` class.

Method	Description
Clone	Makes a copy of the current <code>PageSettings</code> .
ToString	Converts the <code>PageSettings</code> into string format.

Table 14.10: `PageSettings` Methods

The following code demonstrates how to set the margins and orientation of a document that is to be printed using the `PageSettings` class.

#### Code Snippet:

```

...
PrintDocument printDoc = new PrintDocument();
printDoc.DocumentName = "My Document";
MessageBox.Show("Printer Name: " + printDoc.DefaultPageSettings.
PrinterSettings.PrinterName);
printDoc.DefaultPageSettings.Margins = new Margins(100,100,100,100);
printDoc.DefaultPageSettings.Landscape = true;

```

```

if (printDoc.PrinterSettings.IsValid)
{
    printDoc.Print();
}

```

The code uses the `DefaultPageSettings` property of the `PrintDocument` class. A message box is used to display the name of the printer using the `PrinterName` property. The `Margins` property is used to set the left, right, top, and bottom margins. This is done by invoking the constructor of the `Margins` class. The margin size is specified as 100 pixels for each left, right, top and bottom margins. This is equal to 1-inch for each of the margins. The `Landscape` property is set to `true`. This means that the document will be printed in landscape layout.

#### 14.1.5 Dialog Boxes and Component Controls

Information can be retrieved or displayed to the user using a dialog box. Microsoft Visual Studio 2005 provides dialog boxes and controls for printing purposes. They allow you to invoke the `Print` dialog box for printing and to preview the document before printing. Previewing the document in Windows Forms applications is similar to previewing the document in Microsoft Word. Some of the dialog boxes and controls that are commonly used for printing are as follows:

##### → PrintPreviewDialog Control

Consider a case where the user wants to preview the document as to how it will appear after printing. This is done using the `PrintPreviewDialog` control of Microsoft Visual Studio 2005. The control is a pre-defined dialog box, which is used to display a document as it will appear after printing. The control provides various buttons for printing, zooming, and displaying one or more pages.

The `PrintPreviewDialog` control appears in the tray at the bottom of the Windows Forms Designer window, after it is added to the form. Table 14.11 lists the most commonly used properties of `PrintPreviewDialog` class.

Property	Description
<code>AcceptButton</code>	Specifies or retrieves the button on the form, which is clicked when the <b>ENTER</b> key is pressed by the user.
<code>CancelButton</code>	Specifies or retrieves the cancel button for the <code>PrintPreviewDialog</code> control.
<code>DataBindings</code>	Retrieves the control's data bindings.
<code>Document</code>	Specifies or retrieves the document, which is to be previewed.
<code>Text</code>	Specifies or retrieves the text that is displayed on the control.

Table 14.11: `PrintPreviewDialog` Properties

The following code demonstrates how to invoke the PrintPreviewDialog box using PrintPreviewDialog class.

#### Code Snippet:

```
...
PrintDocument printDoc = new PrintDocument();
printDoc.DocumentName = "My Document";
PrintPreviewDialog ppdlgPreview = new PrintPreviewDialog();
ppdlgPreview.Document = printDoc;
ppdlgPreview.ShowDialog();
```

The code uses the Document property of the PrintPreviewDialog class. This property is assigned the object of the PrintDocument class to get the print settings. The ShowDialog() method is invoked to display the **Print Preview** dialog box and to invoke the PrintPage event.

#### → PrintDialog Component

The PrintDialog component is used to select a printer, pages to be printed, and other print-related options. This is similar to the **Print** dialog box in Windows. The user can print the selected pages or the whole document using the PrintDialog component. The class defines various properties and methods that are used to identify printer settings and to reset the page. Table 14.12 lists the most commonly used properties and methods of PrintDialog class.

Name	Description
AllowCurrentPage	This property specifies or retrieves a value that indicates whether the <b>Current Page</b> option button is to be displayed.
AllowPrintToFile	This property specifies or retrieves a value that indicates whether the <b>Print to file</b> check box is enabled.
AllowSelection	This property specifies or retrieves a value that indicates whether the <b>Selection</b> option button is enabled.
AllowSomePages	This property specifies or retrieves a value that indicates whether the <b>Pages</b> option button is enabled.
Document	This property specifies or retrieves a value that indicates the PrintDocument, which is used to obtain PrinterSettings.
PrinterSettings	This property specifies or retrieves the printer settings, which are modified by the dialog box.
PrintToFile	This property specifies or retrieves a value that indicates whether the <b>Print to file</b> check box is selected.
ShowHelp	This property specifies or retrieves a value that indicates whether the Help button is to be displayed.
Reset	This method resets all options, the last selected printer, and the page settings to their default values.

Table 14.12: PrintDialog Properties and Methods

The following code demonstrates how to invoke the standard **Print** dialog box using the **PrintDialog** class.

**Code Snippet:**

```
PrintDocument printDoc = new PrintDocument();
PrintDialog pdlgPrint = new PrintDialog();
pdlgPrint.ShowHelp = true;
pdlgPrint.Document = printDoc;
pdlgPrint.ShowDialog();

private void pdlgPrint_HelpRequest(object sender, EventArgs e)
{
    MessageBox.Show("You clicked the Help button");
}
```

The code uses the **ShowHelp** property to **true**, which indicates that the **Help** button will appear on the title bar of the dialog box. The **Document** property is assigned the object of the **PrintDocument** class to get page settings. Now, the **ShowDialog()** method invokes the **Print** dialog box. When the user clicks the **Help** button, the **HelpRequest** event is raised. When this event is raised, a message box appears stating that the **Help** button is clicked.

→ **PrintPreviewControl Component**

The **PrintPreviewControl** control represents the raw part of print previewing. The **PrintPreviewControl** class is used to implement this control. The class defines various properties and methods used to identify the document to be previewed and to refresh the document preview. Table 14.13 lists the most commonly used properties and methods of **PrintPreviewControl** class.

Name	Description
Columns	This property specifies or retrieves the number of pages that are displayed horizontally across the screen.
Document	This property specifies or retrieves a value that indicates the document, which is to be previewed.
Rows	This property specifies or retrieves the number of pages that are displayed vertically down the screen.
StartPage	This property specifies or retrieves the page number of the upper left page.
Zoom	This property specifies or retrieves a value that indicates how large the pages will appear.
InvalidatePreview	This method refreshes the document's preview.

Table 14.13: **PrintPreviewControl Properties and Methods**

The following code demonstrates how to set the document for preview using the `PrintPreviewControl` class.

#### Code Snippet:

```
PrintDocument printDoc = new PrintDocument();
printDoc.DocumentName = "My Document";
PrintPreviewControl ppcontPreview = new PrintPreviewControl();
ppcontPreview.Dock = DockStyle.Fill;
ppcontPreview.Document = printDoc;
ppcontPreview.StartPage = 1;
ppcontPreview.Zoom = 0.50;
ppcontPreview.InvalidatePreview();
//Code for PrintPage event
```

The code uses the `Dock` property of the `PrintPreviewControl` class. The property is assigned the `Fill` value of the `DockStyle` enumeration. This will dock the control to the edges of the form. The `StartPage` property sets the first page that needs to be printed. By default, the zoom size of the control is 30 percent. To increase this size to 50 percent, the `Zoom` property is used. The `InvalidatePreview()` method refreshes the document preview. This method refreshes the document with its latest appearance, if it has been changed.

### Knowledge Check 1

- Can you match the properties and methods of `PrinterSettings` class against their corresponding descriptions?

Description		Property, Method, and Event	
(A)	This method provides the printer settings information in string format.	(1)	InstalledPrinters
(B)	This property indicates the last page number to print.	(2)	PrinterName
(C)	This property indicates the names of all printers installed on the machine.	(3)	PrintFileName
(D)	This property identifies the printer for printing.	(4)	ToPage
(E)	This property identifies the file, while printing to a file.	(5)	Tostring

- Which of these statements about printing classes are true?

(A)	The information needed for <code>BeginPrint</code> and <code>EndPrint</code> events are provided by <code>PrintEventArgs</code> class.
(B)	The <code>IsPreview</code> property of <code>PrintController</code> class is used to indicate whether the document can be previewed or not.
(C)	The <code>PrintAction</code> property of <code>PrintEventArgs</code> class is used to print the document.
(D)	The <code>Bounds</code> property of <code>PageSettings</code> class is used to get the page coordinates.
(E)	The <code>Reset()</code> method of <code>PrintDialog</code> class is used to select a printer.

3. You want to print a circle using the PrintDocument class. Which one of the following codes will help you to achieve this?

(A)	<pre>Button btnPrint = new Button();  PrintDocument printDoc = new PrintDocument();  private void printDoc_PrintPage(object sender, PrintEventArgs e) {     e.PageSettings.Color = true;     e.Graphics.DrawEllipse(Pens.Green, 30, 30);  }  private void btnPrint_Click(object sender, EventArgs e) {     printDoc.DocumentName = "My Document";     printDoc.Print(); }</pre>
(B)	<pre>Button btnPrint = new Button();  PrintDocument printDoc = new PrintDocument();  private void printDoc_PrintPage(object sender, PrintEventArgs e) {     e.PageSettings.Color = true;     e.Graphics.DrawEllipse(Pens.Green, 100, 100, 30, 30);  }  private void btnPrint_Click(object sender, EventArgs e) {     printDoc.DocumentName = "My Document"; }</pre>

(C)	<pre>Button btnPrint = new Button(); PrintDocument printDoc = new PrintDocument();  private void printDoc_PrintPage(object sender, PrintPageEventArgs e) {     e.PageSettings.Color = true;     e.Graphics.DrawEllipse(Pens.Green, 100, 100, 30, 30); }  private void btnPrint_Click(object sender, EventArgs e) {     printDoc.DocumentName = "My Document";     printDoc.Print(); }</pre>
(D)	<pre>Button btnPrint = new Button(); PrintDocument printDoc = new PrintDocument();  private void printDoc_PrintPage(object sender, PrintPageEventArgs e) {     e.PageSetting.Color = true;     e.Graphics.DrawEllipse(Pens.Green, 30); }  private void btnPrint_Click(object sender, EventArgs e) {     printDoc.DocumentName = "My Document"; }</pre>

## 14.2 Help in Windows Forms

In this second lesson, **Help in Windows Forms**, you will learn to:

- Identify the use of help in applications.
- Explain the different components of help.
- Explain the `Help` class in .NET.

### 14.2.1 Use of Help in Applications

Consider a scenario where a new employee has joined in a software firm. The firm uses an application to allow the employees to fill in their leave information. The new employee is appointed as the accountant and does not know how to operate the application. Now, the employee wants to understand the working of the application.

This functionality can be provided by creating help files. If the particular application provides a help file to the users, it would give the user full guidance and description on how to perform a particular task. Windows Forms allows creating help files similar to the help file of the standard Windows applications such as Internet Explorer and Microsoft Word.

A help file allows you to view the help contents, search for a particular task or concept, and so on. These functionalities of a help file are provided by its four different components. These components are as follows:

#### → Content

The Content Window represents the information structure provided by the help file. The contents of a particular topic can be viewed by navigating through the topic and clicking it. You can specify a filter criterion to view a particular topic.

#### → Index

The Index Window displays a list of topics organized in an alphabetical order. It is similar to an index page of a book. It helps in searching for keywords and topics related to the keywords. This searching is easy, fast, and efficient.

#### → Search

The Search dialog box searches for a specified keyword or a phrase. It can be used when the users are not aware of where to search, for the desired information. You can use different options while searching. Some of these options include Look for, filtered by, and match related words.

#### → Favorites

The Favorites Window lists the links to items such as Help topics and Web pages.

You can add these items to favorites, if you want to access them frequently. You can also organize the items into folders, rename, and delete items.

### 14.2.2 Help Class

The `Help` class is used to access and display the contents of HTM or CHM help files. These help files include the contents, index, and search components. The `Help` class exists in `System.Windows.Forms` namespace. You cannot instantiate the `Help` class. The class encapsulates the HTML Help 1.0 engine and defines static methods used to display the help file. Table 14.14 lists the most commonly used methods of `Help` class.

Method	Description
<code>ShowHelp</code>	Displays the help file contents.
<code>ShowHelpIndex</code>	Displays the index of a help file.
<code>ShowPopup</code>	Displays a Help pop-up window.

Table 14.14: Help Methods

The following syntax demonstrates the `ShowHelp()` method:

**Syntax:**

```
public static void ShowHelp(Control parentControl, string url)
```

where,

`parentControl`: Is a control that is the parent of the `Help` dialog box, which is to be displayed.

`url`: Is the complete path along with the name of the help file. This `url` can be a local path such as 'D:\Help' or a HTTP URL such as 'http://Help.htm'.

The following syntax demonstrates the `ShowHelpIndex()` method:

```
public static void ShowHelpIndex (Control parent, string url);
```

The following code demonstrates the use of the `ShowHelp()` and `ShowHelpIndex()` methods of the `Help` class.

**Code Snippet:**

```
Button btnShowHelp = new Button();
this.Controls.Add(btnShowHelp);

private void btnShowHelp_Click(object sender, EventArgs e)
{
    Help.ShowHelp(this, @"c:\Employees.chm");
    Help.ShowHelpIndex(this, @"c:\Employees.chm");
}
```

The code adds the button on the form. When the user clicks the button, the specified help file is displayed to the user.

This is done using the `ShowHelp()` method of the `Help` class, which takes the full location of the help file as the parameter. The `ShowHelpIndex()` method displays the Index tab when the help file is displayed to the user.

### 14.2.3 `ShowHelp()` Method

The `ShowHelp()` method of the `Help` class takes in the third parameter, which is the `HelpNavigator`. The `HelpNavigator()` method helps to set the element of the help file to display. The value of the third parameter can be `TableOfContents`, `Find`, `Index`, `Topic` or a keyword.

### Knowledge Check 2

- Which of these statements about Help in Windows Forms are true?

(A)	The <code>Help</code> class exists in <code>System.Windows.Forms.Help</code> namespace.
(B)	The Content component of a help file displays the favorite topics in the right pane.
(C)	The <code>ShowHelp()</code> method of <code>Help</code> class displays the help file contents.
(D)	The <code>ShowPopup()</code> method of <code>Help</code> class invokes the help file in the application.
(E)	One of the parameters in the <code>ShowHelp()</code> method is the element of a help file to be displayed.

- You want to display the `TableOfContents` of the `Students` help file using the `Help` class. Which one of the following codes will help you to achieve this?

(A)	<pre>Button btnShowIndex = new Button();  private void btnShowIndex_Click(object sender, EventArgs e) {     Help.ShowHelp(this, @"c:\Students.chm",     HelpNavigator.TableOfContents); }</pre>
(B)	<pre>Button btnShowIndex = new Button();  private void btnShowIndex_Click(object sender, EventArgs e) {     Help.ShowHelp(this, @"c:\Students.chm",     HelpNavigators.TableOfContents); }</pre>

(C)	<pre>Button btnShowIndex = new Button();  private void btnShowIndex_Click(object sender, EventArgs e) {     Help.ShowHelp(this, "c:\Students.chm",         HelpNavigator.TableOfContent); }</pre>
(D)	<pre>Button btnShowIndex = new Button();  private void btnShowIndex_Click(object sender, EventArgs e) {     Help.ShowHelp(this, "c:\Students.chm",         HelpNavigators.TableOfContent); }</pre>

### 14.3 Help Components and Controls

In this last lesson, **Help Components and Controls**, you will learn to:

- List the steps to create a .chm file using HTML help.
- Describe the `HelpProvider` component.
- Explain the `HelpButton` property.
- Describe the Pop-Up help.
- Explain the `ToolTip` component.

#### 14.3.1 HTML Help Workshop

Microsoft HTML Help Workshop (Hhw.exe) is used to design a help file by creating and integrating Hypertext Markup Language (HTML) files as source files. HTML Help Workshop allows you to create a single project for a help file. This project contains all the different files that make up the help system and is saved as .hhp file. It also enables you to create and edit Table of Contents (.hhc) and Index (.hhk).

HTML Help Workshop includes the HTML Help compiler, which allows the user to view help files using the HTML Help Viewer. This is done by compiling the project.

Figure 14.1 displays the HTML Help Workshop.

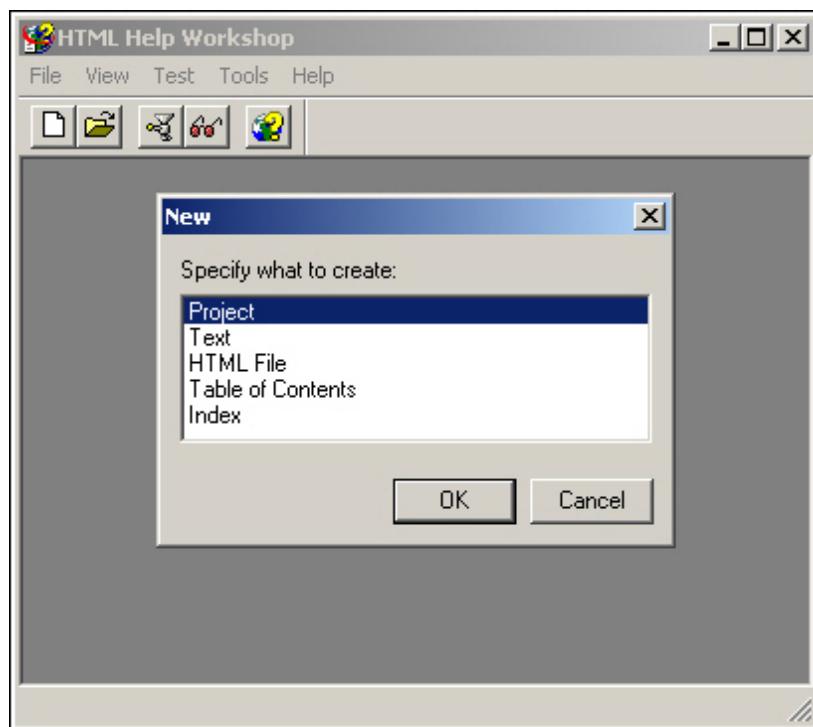


Figure 14.1: HTML Help Workshop

### 14.3.2 Types of Help

A help file created using HTML Help Workshop can be displayed within a Windows-based application. It can also be deployed as a stand-alone system on a Web site. HTML Help Workshop can be used to create two kinds of help that are as follows:

- **Window-Level Help:** The requested help topic appears in a new window. The content of this help topic is shown in an HTML file displayed within the window. This is similar to the help file that opens in Microsoft Word when you press the **F1** key.
- **Context-Sensitive Help:** The requested help topic appears in a pop-up window. The content of this help topic is saved in a .txt file from which the content is shown in the popup window.

### 14.3.3 Steps to create a .chm File

Each help topic is created separately and finally they are integrated in an HTML Help Workshop project. The steps to create a help file using the HTML Help Workshop are as follows:

1. Open **HTML Help Workshop**.
2. Click the **New** option from the **File** menu. The **New** dialog box appears.
3. Click **OK**. The **New Project** wizard appears.

4. Click **Next** to open the **New Project – Destination** page. Here, you should select the location where the file will be saved with the file name.
5. Click **Next** to go to **New Project – Existing Files** page to add any existing help files.
6. Click **Finish**.

You can now create HTML files by selecting the **HTML File** option in the **New** dialog box. You can use the **Contents** and **Index** tabs along with these files. Finally, you must compile the project using the **Compile** button on the toolbar.

#### *14.3.4 HelpProvider Component*

The **HelpProvider** component is used to link the help file to a Windows-based application. It also supports context-sensitive help for controls and dialog boxes. The component is capable of opening specific parts of a help file such as Table of Contents, Index and Search.

You can associate an HTML Help 1.x Help File (.chm or .htm) with Windows-based applications. The component functions in coordination with the properties of the controls existing on the form. To display the help file, the user must focus on the control for which help is required, and press the **F1** key to display the help file.

##### → Properties

The properties of **HelpProvider** class identify the help file associated with the **HelpProvider** object and provide secondary information about the object. Table 14.15 lists the most commonly used properties of **HelpProvider** class.

Property	Description
HelpNamespace	Specifies or retrieves a value that identifies the help file name, which is associated with the mentioned object of the <b>HelpProvider</b> class.
Tag	Specifies or retrieves the object that holds supplemental information about the <b>HelpProvider</b> .

Table 14.15: **HelpProvider** Properties

##### → Methods

The methods of the **HelpProvider** class can be used to set the help keyword and help command. Table 14.16 lists the most commonly used methods of **HelpProvider** class.

Method	Description
GetHelpKeyword	Retrieves the Help keyword for the given control.
GetHelpNavigator	Retrieves the present <b>HelpNavigator</b> settings specified for the particular control. These settings specify which components of the help file are to be displayed. The settings are defined in the <b>HelpNavigator</b> enumeration.

Method	Description
GetShowHelp	Retrieves a value that indicates whether the help for a particular control is to be displayed.
SetHelpKeyword	Specifies the keyword, by which the Help is retrieved, once the user asks for Help for a particular control. The keyword should be specified as <code>topicName.htm</code> to open help regarding a particular topic.
SetHelpNavigator	Specifies the Help command, while Help is being retrieved from the Help file for a particular control.
SetShowHelp	Specifies whether Help is to be displayed for a particular control.
ToString	Retrieves a string, which represents the current <code>HelpProvider</code> .

Table 14.16: HelpProvider Methods

The following code demonstrates how to invoke the help file using the `HelpProvider` class.

**Code Snippet:**

```
...
HelpProvider hlpCustomerDetails = new HelpProvider();
hlpCustomerDetails.HelpNamespace = @"c:\Help\Customer.chm";
hlpCustomerDetails.SetShowHelp(txtCustomerName, true);
```

The code uses the `HelpNamespace` property to refer to the help file to be displayed. The `SetShowHelp()` method is used to display the help for the `TextBox` control, namely `txtCustomerName`. It takes two parameters, which are the name of the `TextBox` control and a boolean value indicating whether to display the help file for the control. When the `txtCustomerName` text box receives focus from the user and when the **F1** key is pressed, the help file is displayed to the user.

You must first add the control on the form for which you need to provide help. Now, drag the `HelpProvider` component and set the Help properties of the control. Next, set the `HelpNamespace` property of the `HelpProvider` component to a help file and use the three properties provided by the `HelpProvider` component to invoke the help file. These properties are present in the Properties window for each control. Table 14.17 lists these three properties.

Property	Description
HelpKeyword	Indicates a string passed through the component to a help file, which will display the required help topic.
HelpNavigator	Indicates a value defined in the <code>HelpNavigator</code> enumeration. This value specifies the component of the help file, which will be displayed to the user. It also identifies how the value of the <code>HelpKeyword</code> property is passed to the help file.
HelpString	Indicates a string, which will be displayed as a tool tip.

Table 14.17: Properties

### 14.3.5 HelpButton Property

The `HelpButton` property is provided by the `Form` class. By default, this property is set to `false`. Once the property is set to `true`, the form displays the Help button with a question mark in the Title bar on the left side of the close button.

To display the Help button, the `Maximize` and `Minimize` properties of the `Form` class must be specified as `false`. The user can display the help file by clicking the Help button and focusing on a particular control on the form. When a control receives focus after clicking the Help button, the `HelpRequested` event of the `Form` class is invoked. You can write the code in its event handler to display the help file.

The following code demonstrates how to display the Help button on the form.

**Code Snippet:**

```
this.HelpButton = true;
this.FormBorderStyle = FormBorderStyle.FixedDialog;
this.MaximizeBox = false;
this.MinimizeBox = false;
```

The code uses the `HelpButton` property, which is set to `true`, to display the Help button. The `FormBorderStyle` property is set to `FixedDialog` to convert the form into a dialog box. The `.MaximizeBox` and `MinimizeBox` properties of the `Form` class are set to `false`. This is a necessary step to display the Help button on the Title bar.

### 14.3.6 Pop-Up Help

Help in Windows Forms can be implemented using the Help button. The Help button is used to display the pop-up help. This help is displayed in a pop-up window for a particular control. It is context-sensitive help, which is similar to **What's This** button on the title bar of the standard dialog boxes. This type of help is more useful for modal dialog boxes. This is because a modal dialog box does not allow you access any other file or window, until the dialog box is closed. This will not allow the users to use the help file. Figure 14.2 displays the Pop-Up Help.



Figure 14.2: Pop-Up Help

The following steps are used to create a pop-up help:

1. Drag a HelpProvider component on the form.
2. Set the HelpButton property to true.
3. Set the form's MinimizeBox, MaximizeBox, and the FormBorderStyle properties.
4. Set the HelpString property of the control in the Properties window.

### 14.3.7 ToolTip Component

The `ToolTip` component is used to provide tool tips to the controls on the form. It provides brief help information to the user when the cursor is pointed at a particular control. The component needs to be associated to the controls for which the tool tip is required. To do so, the `ToolTip` class is used. The class provides a rectangular pop-up window that contains the help information.

The `ToolTip` class provides various properties, methods, and events to provide tool tips for controls. It exists in `System.Windows.Forms` namespace.

#### → Properties

The properties of the `ToolTip` class allow you to set the icons and title for the pop-up window. Table 14.18 lists the most commonly used properties of `ToolTip` class.

Property	Description
Active	Specifies or retrieves a value that indicates whether the <code>ToolTip</code> is currently active.
IsBalloon	Specifies or retrieves a value that indicates whether the <code>ToolTip</code> should use a balloon window.
ShowAlways	Specifies or retrieves a value that indicates whether a <code>ToolTip</code> window is to be displayed, although its parent control is not active.
ToolTipIcon	Specifies or retrieves a value, which defines the icon type to be displayed next to the <code>ToolTip</code> text.
ToolTipTitle	Specifies or retrieves a title for the <code>ToolTip</code> window.

Table 14.18: `ToolTip` Properties

#### → Methods

The methods of `ToolTip` class are used to set or hide the tool tip. Table 14.19 lists the most commonly used methods of `ToolTip` class.

Method	Description
GetToolTip	Retrieves the <code>ToolTip</code> text linked with a particular control.
Hide	Hides the particular <code>ToolTip</code> window.
RemoveAll	Removes all <code>ToolTip</code> text linked with the <code>ToolTip</code> component.

Method	Description
SetToolTip	Links the ToolTip text with a particular control.
Show	Specifies the text linked with a ToolTip and subsequently displays it.
ToString	Retrieves the string representation for the current control.

Table 14.19: ToolTip Methods

→ **Events**

Table 14.20 lists the most commonly used events of ToolTip class.

Event	Description
Draw	Occurs when the ToolTip is drawn and the OwnerDraw property is set to true.
Popup	Occurs before the initial display of the ToolTip. This event is set as the default event for the ToolTip class.

Table 14.20: ToolTip Events

The following code demonstrates how to set a tool tip for a text box using the ToolTip class.

**Code Snippet:**

```
...
ToolTip tipTextHelp = new ToolTip();
tipTextHelp.SetToolTip(txtCustomerName, "Enter a customer name");
```

The code uses the SetToolTip() method of the ToolTip class. The method takes in the control name namely, txtCustomerName and the help message to be displayed for the control as the two parameters. When the user focuses on txtCustomerName control, the help message is displayed to the user in a small rectangular window.

### Knowledge Check 3

- Which of these statements about the help files, HelpProvider component, pop-up help, and ToolTip component are true?

(A)	HTML Help workshop allows you to create Table of Contents with .hhc extension.
(B)	The New Project – Destination page of HTML Help Workshop allows you to specify the path to save the help file.
(C)	The HelpNamespace property of Help class specifies the component of a help file to be displayed.
(D)	The HelpButton property of Form class supports the MinimizeBox property of the Form class.
(E)	The GetToolTip() method of ToolTip class fetches the control for which the tool tip is displayed.

2. You want to display a tool tip as "Click the button to save the information in the database" for a button using the `ToolTip` class. Which one of the following codes will help you to achieve this?

(A)	<pre>Button btnClick = new Button();  ToolTip tipTextHelp = new ToolTip();  tipTextHelp.ToolTip(btnClick, "Click the button to save the information in the database.");</pre>
(B)	<pre>Button btnClick = new Button();  ToolTip tipTextHelp = new ToolTip();  tipTextHelp.SetToolTip(btnClick, "Click the button to save the information in the database.");</pre>
(C)	<pre>Button btnClick = new Button();  ToolTip tipTextHelp = new ToolTip();  tipTextHelp.SetToolTip(btnClick, "Click the button to save the information in the database.");</pre>
(D)	<pre>Button btnClick = new Button();  Tooltip tipTextHelp = new Tooltip();  tipTextHelp.SetToolTip(btnClick, "Click the button to save the information in the database.");</pre>

3. Can you match the properties, methods, and events of the `ToolTip` class against their corresponding descriptions?

	Description		Property, Method, and Event
(A)	This event occurs before the <code>ToolTip</code> is displayed initially.	(1)	<code>SetToolTip</code>
(B)	This method retrieves the <code>ToolTip</code> text of a particular control.	(2)	<code>Active</code>
(C)	This property indicates whether a <code>ToolTip</code> window is to be always displayed.	(3)	<code>Popup</code>
(D)	This property indicates whether the <code>ToolTip</code> is active currently.	(4)	<code>ShowAlways</code>
(E)	This method associates the <code>ToolTip</code> text with a particular control.	(5)	<code>GetToolTip</code>



## Module Summary

In this module, **Printing and Adding Help**, you learnt about:

→ **Printing in Windows Forms**

Print-related services for Windows Forms applications are provided by `System.Drawing.Printing` namespace. Some of the classes, dialog boxes, and controls used for printing are `PrintDocument`, `PrintPreviewDialog`, `PrintDialog`, and `PrintPreviewControl`. The `Print()` method and `PrintPage` event of the `PrintDocument` class is used for printing the document.

→ **Help in Windows Forms**

The four main components of the help file are Content, Index, Search, and Favorites. The `ShowHelp()` method of the `Help` class is used to display the contents of the help file.

→ **Help Components and Controls**

HTML Help Workshop helps in creating compiled help files. To display this file in the Windows-based applications, the `HelpProvider` component is used. HTML Help Workshop can be used to create Window-Level Help and Context-Sensitive Help. The `ToolTip` component provides a tool tip to assist the user.

# Module - 15

## Packaging and Deploying

Welcome to the Module, **Packaging and Deploying**.

Packaging combines the required items that make up an application into a single package. Deployment is a procedure to install the software package on the target machine. To minimize the time and manual effort taken in the installation, the ClickOnce deployment technology has been introduced.

In this Module, you will learn about:

- ➔ Packaging and Deploying
- ➔ Modes of Packaging
- ➔ Deploying Applications
- ➔ Installation Editors
- ➔ ClickOnce Deployment

## 15.1 Packaging and Deploying

In this first lesson, **Packaging and Deploying**, you will learn to:

- Describe packaging and deployment in .NET.
- Explain the packaging concepts.
- Explain the deployment concepts.

### 15.1.1 Purpose

Consider a scenario of an IT firm who wants to sell their customized .NET software, Payroll application, to their customers. They want to secure their application by avoiding the distribution of the source code. To meet such requirements, Microsoft Visual Studio 2005 provides the packaging and deployment options.

Packaging is a process of integrating all the files that make up an application and are required for installing and executing the application. Deployment is a process of distributing the application so that it can be installed on the user's computer.

### 15.1.2 Packaging Concepts

Packaging creates a package that consists of one or more cabinet (.cab) files and assembly files. The .cab files make the process of distribution and installation less time-consuming over a network. They include the compressed project files, setup programs, secondary .cab files, and any other files that you might require to install and run the application. These files can be freely distributed across various companies. The packaging wizard simplifies the process of packaging and distributing the files. The steps involved in the packaging process are as follows:

#### → Determine the Files You Want to Distribute

The Packaging Wizard determines the project files and dependent files for your application. Project files are the files that the application holds. Dependency files are saved with .dep extension. These are the files or components that your application requires to run.

#### → Create Dependency Files for Application's Components

If you feel your application needs dependency files, you should create them before creating the package and then, later include them in the package.

#### → Determine Where to Install Files on the User's Computer

You should install all the Program and setup files into a subdirectory within the **Program Files** directory. The dependency files should be installed into the **\Windows\System** or **\Winnt\System32** directory.

## → Create Your Package

The packaging wizard creates the package and the setup program (setup.exe), which refers to all necessary files.

### **15.1.3 Deployment Concepts**

Deployment allows you to distribute your application once it is built. These applications are installed on the target computer. In Microsoft Visual Studio 2005, you can deploy applications using Microsoft Windows Installer technology. This technology consists of Windows Installer, which is used for installing software and adds any required software components. It also helps in removing the registry keys of the software when the software is uninstalled.

Whenever you deploy an application, it is deployed in the form of assemblies. You can deploy an assembly as DLL or EXE files. There are many methods of deploying assemblies. One of the methods is using **XCOPY**. This method copies the assembly to a private folder without any registry entries. The other method is using the **Gacutil.exe** tool used for shared assemblies. This tool registers and installs the shared assemblies used across applications on the target machine.

Microsoft .NET Framework provides various features that allow you to easily deploy your Windows applications. Some of the deployment features are as follows:

#### → No-impact Applications

Microsoft .NET allows easy isolation of applications, which minimizes DLL conflicts. This means that each application is treated independently. The conflicts arise when a particular version is being used by an application and some other application tries to modify it.

#### → Private Components by Default

Microsoft .NET Framework allows you to deploy components, which are directly saved in the application directory. These components are visible only to the application that contains it. Therefore, each application can access only those components which they hold.

#### → Integration with Microsoft Windows Installer

Microsoft .NET Framework is integrated with Microsoft Windows Installer that allows you to publish, repair, and install-on-demand while deploying the application.

#### → Enterprise Deployment

Microsoft .NET Framework allows you to easily distribute an application, which can use the Active Directory feature.

#### → ClickOnce Deployment

Microsoft .NET Framework provides the ClickOnce deployment feature that is used to publish Windows-based applications to a network file share, or Web Server, for simplified installation.

The main advantage of this feature is that the user can upgrade the software with fewer efforts. This technology reinstalls only those files that have been changed.

There are some more deployment features. These features take care of versioning and updating applications. The remaining deployment features are as follows:

→ **XCOPY Deployment**

Microsoft .NET Framework allows you to deploy applications by using the XCOPY command. This command copies the files of the application from one directory to other. The independent applications are deployed without adding an entry in the registry and without considering the dependent files.

→ **Side-by-side Versioning**

Microsoft .NET Framework allows multiple versions of the components and application to exist together. You can choose the suitable version to use. The versioning policy defined by .NET CLR are then, applied to these items to make the correct versions available and to manage them efficiently.

→ **On-the-fly Updates**

Microsoft .NET Framework allows you to update the DLLs present on the remote computer. This can be done using ASP.NET.

→ **Partially Trusted Code**

The application identity is based on the code, instead of the user.

→ **Controlled Code Sharing**

By default, .NET Framework does not support code sharing. If you want to share your code across other users, you need to explicitly make it available so that other users can access it.

## *Knowledge Check 1*

- Which of the following statements about packaging and deployment are true?

(A)	The application package consists of the files required to run the application.
(B)	The packaging process makes a single package holding only the assembly files.
(C)	The deployment process involves deploying the application as assemblies.
(D)	The packaging process uses the Microsoft Windows Installer technology.
(E)	The side-by-side versioning feature of deployment supports multiple versions of components.

2. Can you match the deployment features against their corresponding descriptions?

Description			Feature
(A)	Provides application identity according to the code.	(1)	XCOPY Deployment
(B)	Avoids DLL conflicts.	(2)	Partially Trusted Code
(C)	Ensures that components are visible to the containing application.	(3)	Enterprise Deployment
(D)	Installs applications without considering the dependencies.	(4)	No-impact Applications
(E)	Supports the use of Active Directory.	(5)	Private Components by Default

## 15.2 Modes of Packaging

In this second lesson, **Modes of Packaging**, you will learn to:

- Describe the ways of packaging an application.
- List and explain the installer files.
- Explain the cab file project.
- Explain the merge modules.

### 15.2.1 Ways of Packaging an Application

Packaging an application involves integrating all the necessary files of the application into a single package. You can package an application in following three ways:

- **Assembly or Collection of Assemblies:** You can directly use the created .dll or .exe files, if you choose this option.
- **Cab Files:** You can compress files as .cab files, if you wish to download them quickly using a Web browser.
- **Microsoft Windows Installer 2.0 or Other Installer Formats:** You can create .msi files for the application to use Windows Installer. You can also package the application to use it with other installers.

### 15.2.2 Definition and Use

An installer file is used to easily and efficiently install an application on the target machine. It provides the benefit of low costs, by supporting the standard format for component and application management.

The common installer file is Microsoft Windows Installer, which is embedded in Windows operating system.

The installer file contains all the procedural details required for installing an application. It concentrates more on what is being installed rather than how it is installed. This is the most important feature of the installer. This feature helps in keeping track of the applications installed, its registry entries, and related files. When the user uninstalls an application, the installer ensures that other applications on the system are not dependent on the application being uninstalled. This guarantees the smooth functioning of other applications.

#### → **Installer Files**

Windows Installer uses .msi file, which is a file format of a package. This .msi file contains information about the application and its installation. For example, it can describe the installation method for an application, whose previous version has been installed on the target computer.

Windows Installer can automatically install a missing file, which might have been accidentally deleted. It also has the capability of canceling the installation procedure. It applies the installation rules, which prevent conflicts with the shared resources needed by different applications. For example, the conflict could be the deletion of a .dll file, which is needed by other applications along with the currently running application.

### **15.2.3 Cab File Project**

A .cab file is a cabinet file, which is used to download components using a Web browser or using a removable media. It is similar to a zip file that compresses multiple files into an easily accessible and installable package. You can create a .cab file if you wish to run the application code on the user's machine rather than on the server.

A cabinet file contains application related files and required registry entries. You must create the cabinet file, while creating a deployment package. The steps to create a cabinet file are as follows:

1. Click **Project** from the **New** option of **File** menu. The New Project dialog box appears.
2. Expand the **Other Project Types** node from the **Project Types** pane of New Project dialog box.
3. Select the **Setup and Deployment Projects** option under the Other Project Types node.
4. Select **CAB Project** from the **Templates** pane.

A .cab file is used to package latest application updates such as new .dll or .exe files providing new functionalities. However, a .cab file has certain restrictions. These are as follows:

- Only a single assembly file can be packaged in a single .cab file.
- Names of the assembly and the .cab file must match. For example, if the file containing the manifest is called **Project.dll**, the cabinet file must be named **Project.cab**.

You can reference a .cab file using the <codeBase> tag in a configuration file. In this tag, you must specify the path of the .cab file.

### 15.2.4 Merge Modules

A merge module consists of different shared components, which are used by various applications. It is a package that consists of component related registry entries and the setup logic. It provides a standard process for managing components and assures that the accurate version of the component is installed on the target machine.

A merge module is saved as .msm extension. A merge module cannot be installed directly on the target machine. A merge module consists of three main components. These are as follows:

- **Merge Module Database:** Consists of installation properties and setup logic
- **Merge Module Summary Information Stream Reference:** Provides details of the module
- **MergeModule.CAB File:** Consists of all the files required by the application related components

**Note** - If you want to run the merge module on the target machine, it must be combined with the .msi file of the application. If a merge module has been distributed once, it is recommended not to modify it. Rather, you must create another merge module for any changes or updates.

### Knowledge Check 2

1. Which of the following statements about installer files, cabinet files, and merge modules are true?

(A)	The packaging modes include packaging an application as a merge module.
(B)	Installer files support the undo operation for the installation procedure.
(C)	Cabinet files include shared components to be downloaded using a Web browser.
(D)	Merge modules contain a database, which contains of a list of applications and components to be installed.
(E)	Merge modules provide a standard method for handling components.

### 15.3 Deploying Applications

In this third lesson, **Deploying Applications**, you will learn to:

- Explain the setup programs.
- Describe how to add a deployment project to a solution.
- Describe how to set the properties for deployment.
- Explain how to add files to the deployment project.

### 15.3.1 Setup Programs

A setup program helps you to create a Windows Installer file that needs to be deployed on the target machine. This file is saved with .msi extension and is used to install the required files on the target machine. This is because the .msi file contains the application, its information such as registry entries, and installation details. This means that .msi file is the most important file needed to deploy an application.

When the .msi file runs on the target machine, all the required files are installed into the **Program Files** directory by default. For example, if you want to install Adobe Acrobat to view PDF files, you run the setup file, which is a .exe file. This will install all the required files to run Adobe Acrobat on the computer. These files will now appear in **Program Files** directory.

**Note** - If the installation of .msi file fails, the installation procedure is rolled back.

### 15.3.2 Adding a Deployment Project

A deployment project specifies the manner in which the solution will be deployed on the target machine. The first step in installing an application is to add a deployment project to the solution. To create a deployment project:

1. Click **New Project** from the **Add** option of the **File** menu. The New Project dialog box appears.
2. Expand the **Other Project Types** node from the **Project Types** pane of the **New Project** dialog box.
3. Select the **Setup and Deployment Projects** option under the Other Project Types node.
4. Select the desired deployment project from the **Templates** pane.

The different deployment projects available are as follows:

#### → **Setup Project**

This deployment project allows you to create setup programs for Windows-based applications.

#### → **Web Setup Project**

This deployment project allows you to create setup programs to deploy applications on a Web server.

#### → **Merge Module Project**

This deployment project integrates components, which will be shared by multiple applications. For example, if an application provides three functionalities using components, these components can be merged using the Merge Module Project. Now, the components can be implemented into similar applications.

→ **Setup Wizard**

This deployment project allows you to create a setup project in less time by following the steps of the wizard. It is used to customize the setup project by adding additional files.

→ **CAB Project**

This deployment project creates a cabinet file for downloading from Internet, Intranet or removable media.

→ **Smart Device CAB Project**

This deployment project creates a cabinet file for deploying applications on smart devices such as Pocket PC.

### *15.3.3 Types of Properties*

The second step in deploying an application is specifying the deployment properties. There are two types of properties that can be specified. These are as follows:

- **General:** These properties are specified using the **Properties** window. For example, they allow you to set the form name that will run as soon as the application starts.
- **Configuration-dependent:** These properties are project-specific and configuration-dependent. They are specified using the **Deployment Project Properties** dialog box.

To specify configuration-dependent properties:

1. Choose the **Property Pages** option from the **View** menu.
2. Select a configuration from the **Configuration** list. If you have common properties, **All Configurations** can be selected to avoid rework.
3. Select a **Category** from a list of categories. This will display the related properties of the category.

**Note** - To access the Deployment Project Properties dialog box, click the Property Pages option on the View menu. Now, select the Build check box in the Configuration Manager window.

Once the deployment project is created, you can access and manipulate the deployment properties in the **Properties** window. To access the properties, you should select the name of the deployment project from the **Solution Explorer**, window after creating the project. For example, if you have created a deployment project with the name **PayrollSetUp**, you must select this name in the **Solution Explorer** window.

Table 15.1 lists the deployment properties.

Property	Description
AddRemoveProgramsIcon	Sets an icon, which will be displayed in the Add/Remove Programs dialog box on the user's machine.
Description	Sets a string description of an application or component to be installed.
DetectNewerInstalledVersion	Sets whether a check for the newer version of the application will be performed or not, while installing the software.
Manufacturer	Sets the manufacturer name of the application or component.
ProductCode	Sets a unique identifier of the application.
ProductName	Sets a public name to identify the application or component.
SearchPath	Sets the location for assemblies or merge modules on the computer where they are built.
Version	Sets the version number of an installer, merge module, or cabinet file.

Table 15.1: Deployment Properties

#### 15.3.4 Adding Files to the Deployment Project

The third step in deploying an application is to create an installer file. This installer will contain the required files. It also contains the path of the target machine where these files will be installed. This task is achieved by adding items to the deployment project. The items that you can add to a deployment project are as follows:

- Project Output
- File
- Merge Module
- Assembly

The steps to add an item to the deployment project are as follows:

1. Choose **File System** from the **Editor** option of the **View** menu.
2. Select a folder on the target machine from the left pane where the item shall be installed.
3. Point to the **Add** option from the **Action** menu and choose the required item.
4. Select the item that needs to be added from the dialog box.

## Knowledge Check 3

1. Which of the following statements about deploying applications are true?

(A)	A setup program installs the necessary files in the <b>Program Files</b> directory by default.
(B)	The <b>Merge Module Project</b> allows creating a single package for shared components.
(C)	The <b>Deployment Property Page</b> option allows you to specify configuration-dependent properties.
(D)	The first step in deploying the software includes setting deployment properties.
(E)	The File System Editor allows selecting items to be added to the deployment project.

2. Can you match the deployment properties against their corresponding descriptions?

Description		Property
(A)	Indicates the icon to display in the Add/Remove Programs dialog box.	(1) DetectNewerInstalledVersion
(B)	Indicates the location of assemblies and merge modules.	(2) AddRemoveProgramsIcon
(C)	Provides details about an application.	(3) Version
(D)	Checks for new version of the software during installation.	(4) SearchPath
(E)	Represents the version of installer and cabinet files.	(5) Description

### 15.4. Installation Editors

In this fourth lesson, **Installation Editors**, you will learn to:

- List the various installation editors.
- Explain File System Editor.
- Describe Registry Editor.
- Explain File Types Editor.
- Explain User Interface Editor.
- Explain Custom Actions Editor.
- Explain Launch Conditions Editor.

### 15.4.1 Installation Editors

Deployment is a process in which an application package is installed on the target machine. The process requires some important details such as the name of target computer and the registry keys to be added. You might also want to display customized screens during the installation.

In Microsoft Visual Studio 2005, these tasks are achieved using the various deployment editors. The main purpose of these editors is to set the properties and custom tasks that will guide the deployment process. A deployment project consists of six main editors. These are as follows:

- File System
- Registry
- File Types
- User Interface
- Custom Actions
- Launch Conditions

An installer file can be configured using the deployment editors. However, none of the editors can be used for creating a cabinet file. The steps for accessing the various deployment editors are as follows:

1. Select a deployment project from the **Solution Explorer** window.
2. Click the required editor by choosing the **Editor** option from the **View** menu.

An alternative way of opening the deployment editor is by clicking the respective editor button on the **Solution Explorer** toolbar. This button appears on the **Solution Explorer** toolbar when a deployment project is selected.

### 15.4.2 File System Editor

The File System Editor is used to add files, folders, project outputs, and other items to a deployment project. The left pane of the editor displays the standard folders found on any computer. The right pane of the editor displays the files and sub-folders existing within the standard folders selected in the left pane. The purpose of this editor is to allow adding and deleting files, folders, and sub-folders within the standard folders on the target machine. Therefore, it also helps in specifying the location of saving the items on the target machine.

The File System Editor is opened by default when you create a deployment project and consist of the following:

- **Application Folder**

This folder is used to store application related files and the application output files.

→ **User's Desktop**

This folder corresponds to the **Desktop** folder on the target machine.

→ **User's Programs Menu**

This folder corresponds to the **Programs Menu** folder on the target machine. Figure 15.1 displays the File System Editor.

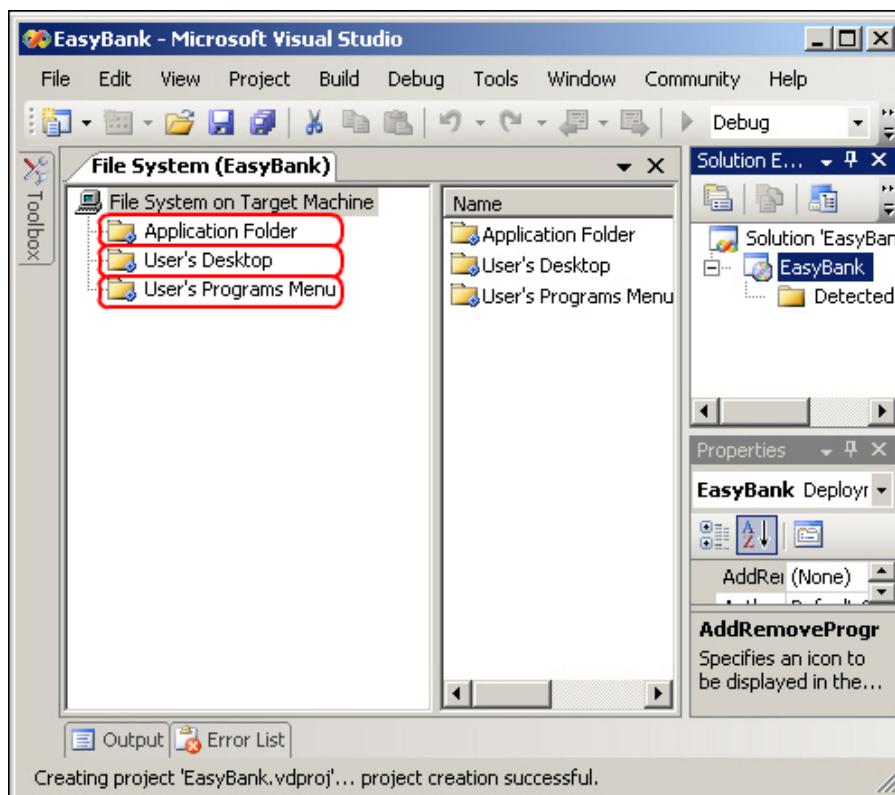


Figure 15.1: File System Editor

The properties for the File System Editor appear in the **Properties** window. These properties are categorized into six different types. The type of project and the currently selected item in the editor determines the availability of properties in the **Properties** window. Table 15.2 lists the six categories of File System Editor properties.

Property Type	Description
Folder Properties	Available when a folder in the editor is selected for a Setup or Merge Module project.
Web Folder Properties	Available when a folder in the editor is selected for a Web Setup project.
File Properties	Available when a file in the editor is selected.
Assembly Properties	Available when an assembly in the editor is selected.

Property Type	Description
Project Output Group Properties	Available when a project output group in the editor is selected.
Shortcut Properties	Available when a shortcut in the editor is selected.

Table 15.2: File System Editor Properties

### 15.4.3 Working with File System Editor

You can use the File System Editor to add HTML pages and readme files. To add files, you can use the menu bar while working with the editor, the **Solution** Explorer window, or drag files from **Windows Explorer**.

To add a file from the File System Editor:

1. Select the required folder in the editor.
2. Click **File** from the **Add** option of the **Action** menu. The **Add Files** dialog box appears.
3. Browse to the file location and add the file from the **Add Files** dialog box.

To remove a file:

1. Select the file to remove from the File System Editor.
2. Click **Remove** from the **Edit** menu.

### 15.4.4 Registry Editor

The Registry Editor helps in setting the registry keys and values for smooth functioning of the application. These keys and values are added to the registry of the target computer when the application is installed. A Registry Editor consists of two sections namely, a navigation pane on the left and a detail pane on the right.

The left pane of the editor displays a set of registry keys corresponding with the standard Windows registry keys. These keys are as follows:

- ➔ HKEY\_CLASSES\_ROOT
- ➔ HKEY\_CURRENT\_USER
- ➔ HKEY\_LOCAL\_MACHINE
- ➔ HKEY\_USERS
- ➔ HKEY\_PER\_USER

The right pane of the editor displays the values of the registry keys. You can create your own registry keys and set string, binary or DWORD values to any key. In addition, you can perform some customized tasks such as importing a registry file.

Figure 15.2 displays the Registry Editor.

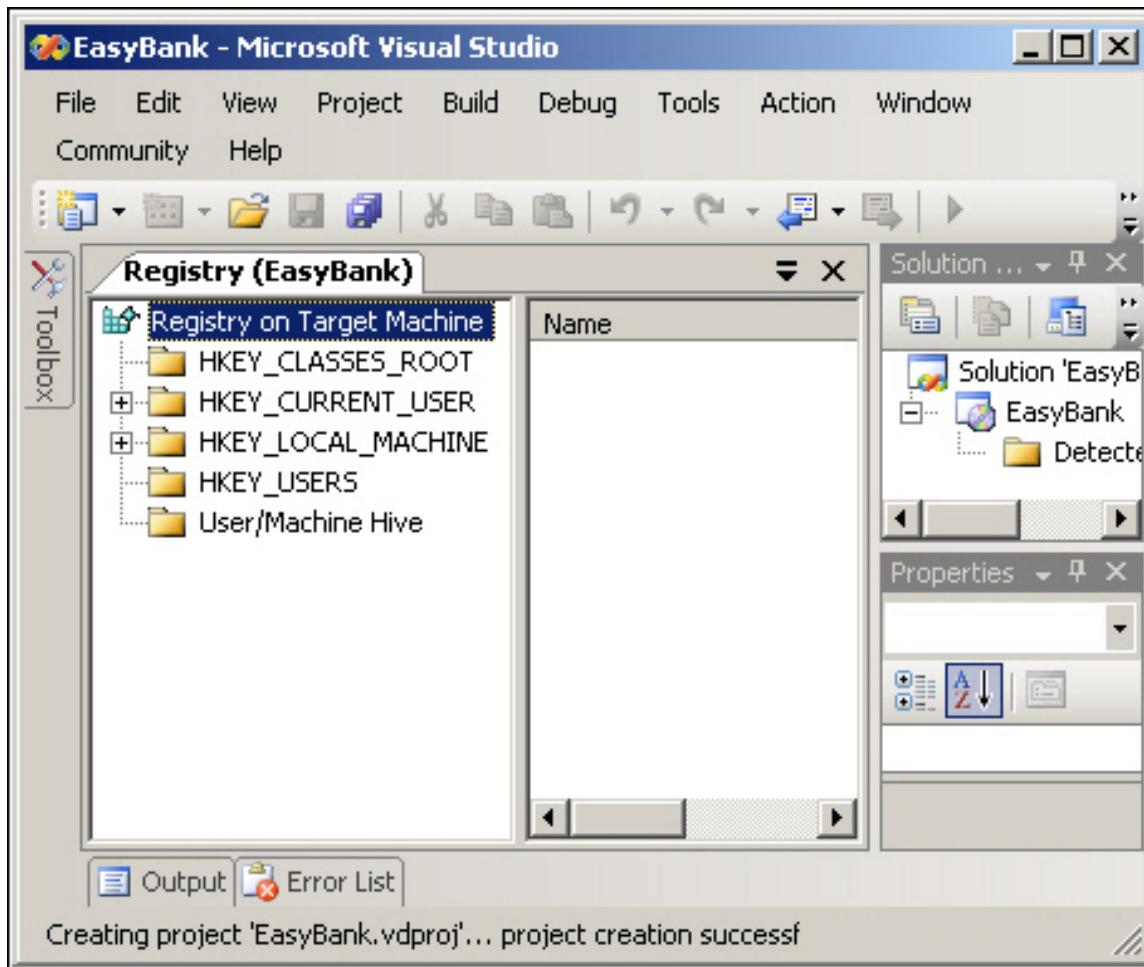


Figure 15.2: Registry Editor

The properties of the Registry Editor allow you to set the registry path for a key and to retrieve the key name. Table 15.3 lists the commonly used properties of Registry Editor.

Property	Description
AlwaysCreate	Indicates whether a selected registry key should be created as part of each installation procedure, even when the registry key is empty.
Condition	Indicates a Windows Installer condition, which must be fulfilled in order to install the selected item during installation.
DeleteAtUninstall	Indicates whether a selected registry key and any subkeys are to be deleted when the application is uninstalled.
FullPath	Indicates the full registry path for a particular registry key. It is a read-only property.
Name	Indicates the name of a particular registry key or value.
Transitive	Indicates whether the Condition property will be rechecked for the selected item during the installation or reinstallation.

Property	Description
Value	Indicates the data stored in the specified registry value.
ValueType	Indicates the type of data in the specified registry value.

Table 15.3: Registry Editor Properties

#### 15.4.5 Working with Registry Editor

Registry keys that are not present in the registry of the target computer are added during the application installation. To add a registry key using the Registry Editor:

1. Select the registry key in the Registry Editor.
2. Click **New Key** from the **Action** menu. A key is added to the node with the highlighted default name.
3. Enter the key name.

Now, you can use the **Properties** window to set the key properties. To delete a key, click **Delete** from the **Edit** menu.

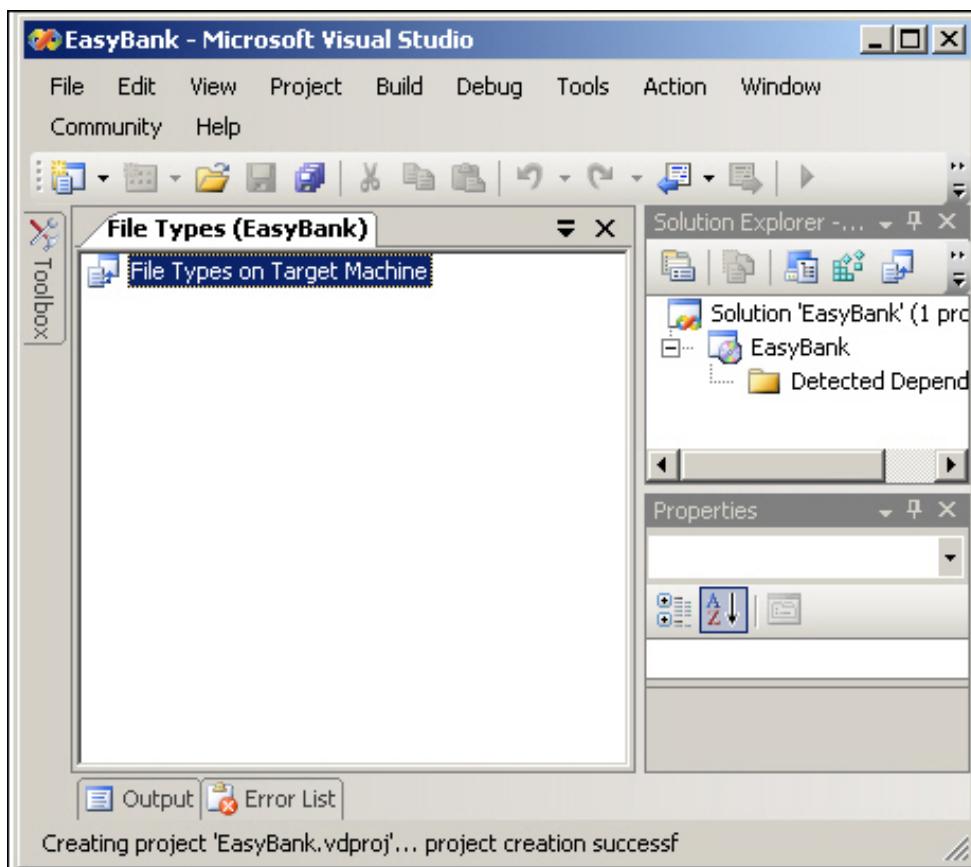
#### 15.4.6 File Types Editor

The File Types Editor is used to create a link between a file with a particular extension and an application. For example, you can use this editor to associate an .xls file with the Microsoft Excel application. You can also set actions that will take place while handling a particular file type. These actions appear in the context menu when a file is right-clicked in the **Windows Explorer** of the target machine.

The uses of the File Types Editor are as follows:

- ➔ Adding and Deleting the Associated File Extensions
- ➔ Adding and Deleting Actions According to the File Type
- ➔ Specifying a Default Action

Figure 15.3 displays the File Types Editor.



**Figure 15.3: File Types Editor**

The properties of the File Types Editor allow you to set the file extensions and provide details for a file type. Table 15.4 lists the various properties of File Types Editor.

Property	Description
Arguments	Indicates a command-line argument for a command, which is invoked by a particular action.
Command	Indicates an executable file, which is executed when an action is fired on a particular file type.
Description	Indicates the description for a particular file type.
Extensions	Indicates the file extensions to be linked to a particular file type.
Icon	Indicates an icon, which is displayed for a particular file type.
Name	Indicates the name, which is used to identify a particular file type in the editor or name of an action, which invokes a verb. It specifies the command name, which will be displayed in the context menu.
Verb	Indicates the verb, which is used to invoke a particular action for a file type. Examples of verbs include <b>open</b> , <b>edit</b> , and <b>play</b> .

**Table 15.4: File Types Editor Properties**

#### 15.4.7 User Interface Editor

The User Interface Editor is used to change the visual appearance of the interface that appears while installing an application. This is implemented by setting the properties of custom dialog boxes, which will appear during the installation procedure.

The User Interface Editor appears as a tree control divided into two sections namely, **Install** and **Administrative Install**. When an end user runs the installer, the dialog boxes in the **Install** section are displayed. When a system administrator uploads the installer on a network, the dialog boxes in the **Administrative Install** section are displayed.

Figure 15.4 displays the User Interface Editor.

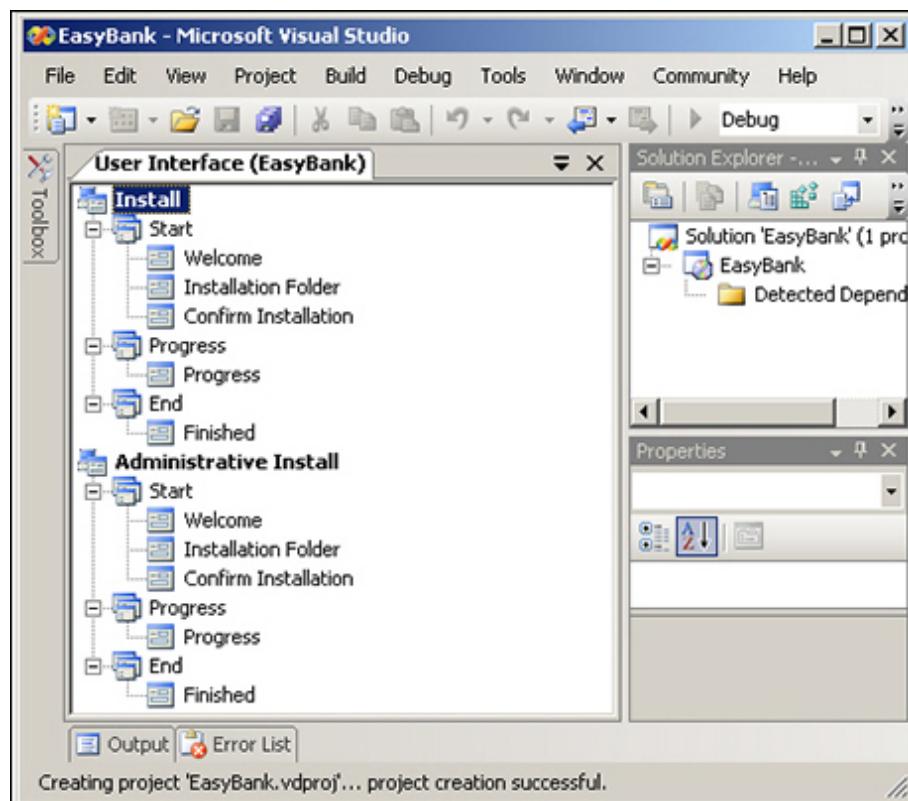


Figure 15.4: User Interface Editor

Both of these sections graphically represent the dialog boxes, which will be displayed during the installation. The dialog boxes will appear in their respective deployment phases namely:

→ **Start**

The Start dialog box is displayed before the installation starts. They are mostly used to collect information from the user or to allow the user to change the installation path.

→ **Progress**

The Progress dialog box is displayed to notify the progress of the installation.

→ End

The End dialog box is displayed when the installation process is completed.

The User Interface Editor provides various properties to set the text of various controls on the user interface dialog box. Table 15.5 lists the various properties of User Interface Editor.

Property	Description
Arguments	Indicates command-line arguments for an executable file, which is called from a <b>Register User</b> dialog box.
BannerText	Indicates the text, which is to be displayed in the banner area of a user interface dialog box.
BodyText	Indicates the text, which is to be displayed in the body of a user interface dialog box.
ButtonNLabel	Indicates the text of a <b>RadioButton</b> control appearing on the user interface dialog box, where <b>N</b> is number of the <b>RadioButton</b> control.
CheckboxNLabel	Indicates the text of a <b>CheckBox</b> control appearing on the user interface dialog box, where <b>N</b> is number of the <b>CheckBox</b> control.
Executable	Indicates an executable file, which will be launched from a Register User dialog box.
ShowSerialNumber	Indicates whether the Serial Number field should be displayed on the Customer Information dialog box.
WelcomeText	Indicates the text, which will be displayed in the body of a Welcome dialog box.

Table 15.5: User Interface Editor Properties

#### 15.4.8 Custom Actions Editor

The Custom Actions Editor is used to set extra actions, which will be performed at the end of the installation on a target computer. Custom action must be first compiled as .dll file, .exe file, assembly file, or script file. This is because they cannot be directly added to the deployment project.

Custom Actions Editor contains four folders, which are attached to the different phases of installation.

Figure 15.5 displays the Custom Actions Editor.

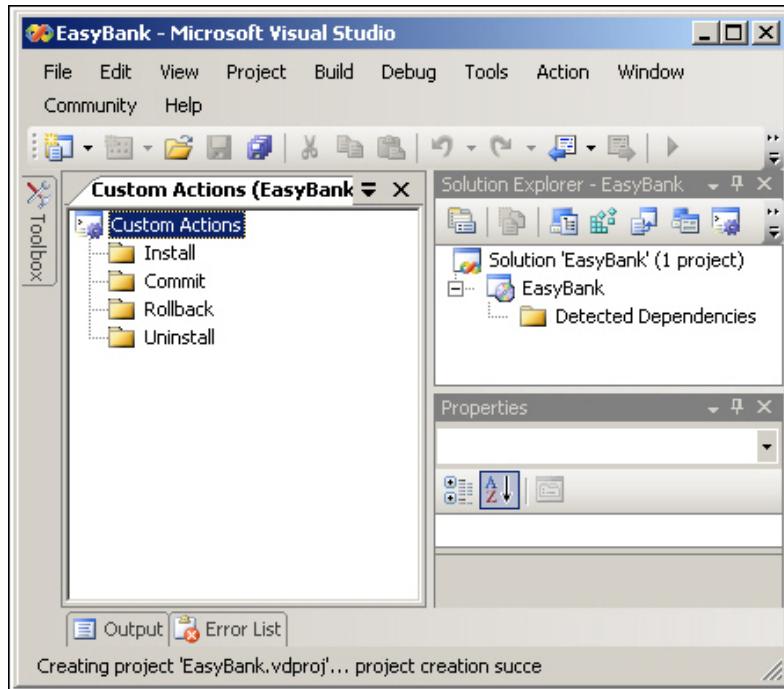


Figure 15.5: Custom Actions Editor

These phases are as follows:

- **Install:** These actions are performed after the files are installed but the installation has not yet committed.
- **Commit:** These actions are performed after the installation has been committed.
- **Rollback:** These actions are performed when the installation is not successful and entire process is rolled back.
- **Uninstall:** These actions are performed when the user uninstalls the application.

To add a custom action, select a folder in the editor and click **Add Custom Action** from the **Action** menu.

The properties of the Custom Actions Editor allow you to set the custom action and use the `Installer` class to handle the installation process. Table 15.6 lists the commonly used properties of Custom Actions Editor.

Property	Description
Arguments	Indicates the command-line arguments for a particular custom action. The property is not considered when the <code>InstallerClass</code> property is <code>true</code> .
Condition	Indicates a Windows Installer condition, which must be fulfilled in order to run a particular custom action during installation.

Property	Description
CustomActionData	Indicates the user-defined data, which is to be passed to the installer. Therefore, this property stores installation information, which can be used by the custom action, if required.
EntryPoint	Indicates an entry point within a .dll file that stores the custom action. The property is not considered when the <code>InstallerClass</code> property is <code>true</code> .
InstallerClass	Indicates whether a custom action is the <code>Installer</code> class of the .NET Framework. This class consists of methods, which manage the installation process. For example, it can roll back a particular installation process.
Name	Indicates the name of a particular custom action.
SourcePath	Indicates the location of the file on the development computer, which contains the custom action. It is a read-only property.

Table 15.6: Custom Actions Editor Properties

#### 15.4.9 Launch Conditions Editor

The Launch Conditions Editor is used when you want to set some conditions, which must be fulfilled for smooth functioning of the installation process. For example, you can check that the installation occurs on the machine having Windows 2000 as the operating system. If the user tries to install on any other operating system, the installation is rolled back according to the condition specified using the editor.

Using the editor, you can also search for a specific file or registry key on the target machine. You can specify the following conditions using the editor:

→ **File Launch Condition**

This condition is used to search for the file on the target machine. If the file does not exist, the installation is rolled back.

→ **.NET Framework Launch Condition**

This condition is used to search for the .NET Framework runtime files on the target machine. If the files do not exist, the installation is rolled back.

→ **Registry Launch Condition**

This condition is used to search for a specific registry on the target machine. If it does not exist, the installation is rolled back.

→ **Windows Installer Launch Condition**

This condition is used to search for a specific Windows Installer component on the target machine. If it does not exist, the installation is rolled back.

→ **Internet Information Services (IIS) Launch Condition**

This condition is used to check whether the Internet Information Services is installed on the target machine. If it is not installed, the installation is rolled back.

→ **Predefined Launch Condition**

This condition is used to set both the search and launch conditions simultaneously. This is done by passing the **Property** property as a reference to the **Condition** property of the editor. The predefined launch conditions can be set for files, registry entries, Windows Installer components, the .NET CLR, and IIS.

Figure 15.6 displays the Launch Conditions Editor.

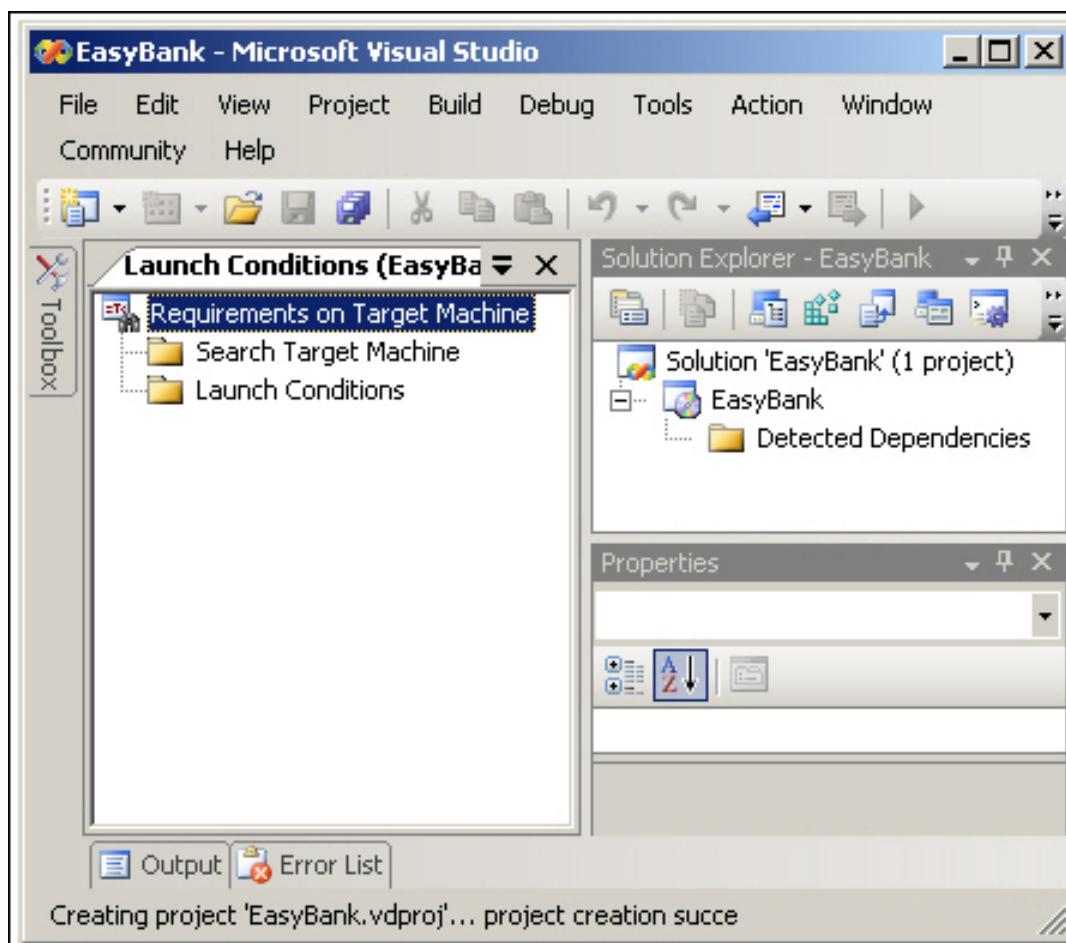


Figure 15.6: Launch Conditions Editor

The properties of the Launch Conditions Editor allow you to search for a file, registry entries, and components on the target machine. They can be set using the **Properties** window.

Table 15.7 lists the most commonly used properties of Launch Conditions Editor.

Property Type	Description
File Search Properties	Available when you select the file search node in the editor.
Registry Search Properties	Available when you select the registry search node in the editor.
Component Search Properties	Available when you select the component search node in the editor.
Launch Condition Properties	Available when you select the launch condition node in the editor.

Table 15.7: Launch Conditions Editor Properties

## Knowledge Check 4

1. Which of these statements about the deployment editors are true?

(A)	The Launch Conditions Editor contains four folders to perform actions during Install, Commit, Rollback, and Reinstall deployment phases.
(B)	The Registry Editor allows you to set binary values to the keys.
(C)	The File System Editor provides the Windows folder to save files in the Windows folder of the user's machine.
(D)	The Extensions property of Registry Editor associates a file extension to a file type.
(E)	The User Interface Editor allows you to create custom dialog boxes, which will be displayed at the time of installation.

2. Can you match the properties of Custom Actions Editor against their corresponding descriptions?

	Description		Property
(A)	Determines whether the <code>Installer</code> class in specifying the custom action.	(1)	<code>CustomActionData</code>
(B)	Determines the file path of custom action.	(2)	<code>InstallerClass</code>
(C)	Determines the custom data for the installer.	(3)	<code>SourcePath</code>
(D)	Determines the Windows Installer condition to be met.	(4)	<code>EntryPoint</code>
(E)	Determines the starting point in a .dll file.	(5)	<code>Condition</code>

## 15.5 ClickOnce Deployment

In this last lesson, **ClickOnce Deployment**, you will learn to:

- ➔ Describe the concept of ClickOnce deployment.
- ➔ Explain the ClickOnce application.

- List and explain the steps to publish the application to Web, file share, and CD-ROM or DVD-ROM.

### 15.5.1 Purpose

Consider an example where an employee of a firm wants to upgrade the payroll software created using Windows Forms. The complete software needs to be reinstalled for upgradation, which is time consuming. It also requires manual effort to continue the installation.

To reinstall the software quickly, Microsoft Visual Studio 2005 has introduced the ClickOnce deployment feature. ClickOnce deployment is used to create setup projects and to make available the setup file from a single Web page. A copy of the package is saved on the user's drive to install the software locally. Moreover, the copy will frequently check for updates on Web and will reinstall only the changed files. ClickOnce deployment has overcome the three main problems regarding installation. They are as follows:

- Reinstalling the complete software.
- Accessing shared components leading to version conflict. This is solved by treating each application separately.
- Requiring only administrative permissions for installation.

ClickOnce deployment is a technology, which allows you to create auto-updating applications. The technology makes it easier to install and upgrade the software on the target machine. This technology is used when the software in use might undergo frequent changes. The reason for these frequent changes might be due to changing user requirements or providing more features due to market competition.

The ClickOnce technology provides a secured environment to deploy applications. You can download and reinstall the software, but with restricted permissions based on a security zone. To install the software from Internet, the security settings are based on the Internet zone. To install and run the software from a Network File Share, the security settings are based on the Intranet zone.

### 15.5.2 ClickOnce Application

A ClickOnce application is an application created using Windows Forms and published using ClickOnce technology. A ClickOnce application is installed on the user's local machine. It runs on the machine even if the computer is not on the network. It can also run when the computer is connected to the network without installing the application permanently on the computer. This type of application checks for any new versions of the installed software. If found, it automatically updates the software with the latest files.

There are three ways to publish the ClickOnce application for making it available for installation. These are as follows:

- Web Page

- Network File Share
- CD-ROM or DVD-ROM

### 15.5.3 Working of ClickOnce Application

The working of a ClickOnce application is based on two XML manifest namely, application, and deployment. An XML manifest is an XML file that contains information about the application, components, and the installation details. The application manifest provides information about the following:

- Application
- Assemblies
- Dependent files
- Files that are a part of the application
- Desired user permissions
- Path where updates will be available

The deployment manifest provides information about how to deploy the application. It includes the location of the application manifest and the latest software version.

Microsoft .NET Framework provides the **Mage.exe** tool to create the manifests. Alternatively, the **Publish** Wizard of Microsoft Visual Studio 2005 is used to create the manifests. Now, the developer can create a copy of the deployment manifest at the target location such as a CD or Web server. From this location, the users can install, update, and run the application.

### 15.5.4 Publishing the ClickOnce Application

A ClickOnce application is published using the **Publish** wizard of Microsoft Visual Studio 2005. The wizard allows you to set the required publishing properties of the application.

### 15.5.5 Web Server Publishing

ClickOnce applications published on a Web server can be installed on the user's machine irrespective of the machine being online or offline. The steps to publish a ClickOnce application to a Web server are as follows:

1. Select the project from **Solution Explorer** window.
2. Right-click the project node and choose the Publish option. The Publish Wizard appears.

3. In the **Where do you want to publish the application?** page, enter a valid URL and click **Next**.
4. In the **Will the application be available offline?** page, select the suitable option:
  - a. Click the **Yes, this application will be available online or offline** option if you want to run the application even when the user is offline. A shortcut on the Start menu will appear to install the application.
  - b. Click the **No, this application is only available online** option if you want to run the application only if the user is connected to the network. A shortcut on the Start menu will not appear.
5. Click **Next**.
6. Click **Finish** for publishing the application.

### 15.5.6 File Share Publishing

ClickOnce applications published on a file share can be used on LAN networks. The steps to publish a ClickOnce application to a file share are as follows:

1. Select the project from **Solution Explorer** window.
2. Right-click the project node and choose the **Publish** option. The Publish Wizard appears.
3. In the **Where do you want to publish the application?** page, enter a valid file path, for example, **\machinename\applicationname** and click **Next**.
4. In the **How will users install the application?** page, select the location for installing the application:
  - a. Click the **From a Web site** option and enter a URL that corresponds to the specified file path. You must select this option if an FTP address is specified as the publishing location. This is because direct download from FTP is not supported.
  - b. Click the **From a UNC path or file share** option. You must select this option if the publishing location is specified as **\Server\[FolderName]**.
  - c. Click the **From a CD-ROM or DVD-ROM** option if you want the users to install from a removable media.
  - d. Click **Next**.
5. In the **Will the application be available offline?** page, click the suitable option:
  - a. Click the **Yes, this application will be available online or offline** option if you want to run the application even when the user is offline. A shortcut on the Start menu will appear to install the application.

- b. Click the **No, this application is only available online option** if you want to run the application only if the user is connected to the network. A shortcut on the **Start** menu will not appear.
6. Click **Next**.
7. Click **Finish** for publishing the application.

### 15.5.7 CD-ROM/DVD-ROM Publishing

ClickOnce applications published to a CD-ROM or DVD-ROM can automatically check for any application updates. This must be specified while publishing the application to these removable medias. The steps to publish a ClickOnce application to a CD-ROM or DVD-ROM are as follows:

1. Select the project from **Solution Explorer** window.
2. Right-click the project node and choose the **Publish** option. The Publish Wizard appears.
3. In the **Where do you want to publish the application?** page, enter the file path or FTP location and click **Next**.
4. In the **How will users install the application?** page, click the **From a CD-ROM or DVD-ROM** option and click **Next**.
5. In the **Where will the application check for updates?** page, choose the suitable update option:
  - a. Click **The application will check for updates from the following location** and specify the path for posting and downloading the updates. This could be a file location, Web site, or FTP Server. This option is useful if you want the application to automatically check for updates.
  - b. Click **The application will not check for updates** if you do not want the application to check for updates.
6. Click **Next**.
7. Click **Finish** to publish the application.

### Knowledge Check 5

1. Which of the following statements about ClickOnce deployment are true?

(A)	ClickOnce deployment reduces the manual effort required in installing an application.
(B)	ClickOnce deployment requires only administrative users to install an application.
(C)	ClickOnce applications use an application manifest to determine the software version.
(D)	ClickOnce applications use a deployment manifest to locate the application manifest.
(E)	Microsoft Visual Studio 2005 provides the <b>Publishing</b> wizard for publishing applications to a removable media.



## Module Summary

In this module, **Packaging and Deploying**, you learnt about:

→ **Packaging and Deploying**

Packaging is a process of merging files required for smooth functioning of an application. The package file consists of one or more .cab files and assembly files. Deployment is a process of distributing and installing the package. Deployment features include easy isolation of applications and integration with Microsoft Windows Installer.

→ **Modes of Packaging**

An application can be packaged as an assembly, cabinet file, or as an installer. The cabinet file is created when you want to download the application components using a Web browser. An installer file minimizes the cost of installation by providing a standard procedure to handle components and applications.

→ **Deploying Applications**

The steps for deploying applications involve creating a deployment project, setting its properties, and adding the items to the project. These items could be project outputs, files, assemblies, and components. Different deployment projects can be created such as Setup Project, Merge Module Project, and the CAB Project.

→ **Installation Editors**

The deployment process is assisted by six deployment editors. File System Editor facilitates adding files and folders to a project. Registry Editor is used for creating new registry keys and values. Launch Conditions Editor is used for specifying conditions to be fulfilled for proper functioning of the installation.

→ **ClickOnce Deployment**

The ClickOnce deployment technology provides incremental reinstallation by updating only those files, which have been changed. A ClickOnce application is created using this technology. This application uses the application and deployment XML manifest. The deployment manifest is deployed on the target machine to reinstall the software.

# Module - 16

## Exploring Windows Forms

Welcome to the Module, **Exploring Windows Forms**.

This session will explore the creation of nonrectangular Windows Forms, various types of ToolStripItems, and merging of DataSets.

In this Module, you will learn about:

- Describe the creation of nonrectangular Windows Forms
- Explain the TabControl and TrackBar controls
- Describe the NotifyIcon component
- Describe the various types of ToolStripItems
- Describe merging of ToolStrips
- Describe the ToolStripContainer class
- Explain custom dialog boxes
- Explain the selection properties of MonthCalendar
- Explain merging of DataSets

## 16.1 Introduction

Over the years, Windows Forms has evolved from a simple technology to a comprehensive application development environment. Windows Forms in Visual Studio 2008 helps you design and develop powerful applications faster and in a more efficient manner.

This session explores miscellaneous controls and features of Windows Forms. These include nonrectangular Windows Forms, TabControl and TrackBar controls, NotifyIcon component, ToolStripItems, ToolStripContainer class, and so forth. The session also describes how to merge ToolStrips, create and use custom dialog boxes, use the selection properties of MonthCalendar, and merge DataSets.]

## 16.2 Nonrectangular Forms

Most often, the user interface of a Windows Forms application is the standard shape of a rectangular form. However, it is also possible to render nonrectangular Windows Forms such as ellipse-shaped or in the form of an image. For example, consider a scenario where you are designing an application for a retail outlet that sells garments for kids. The user interface for this application must look eye-catching and attractive. To achieve this, you can create a nonrectangular Windows Form with a rich color.

Consider an example to create a nonrectangular form. The steps are described as follows:

1. Create a C# based Windows Forms application. Rename the form to **frmEllipse**.
2. Set the **FormBorderStyle** property of the form to **None** as shown in figure 16.1.

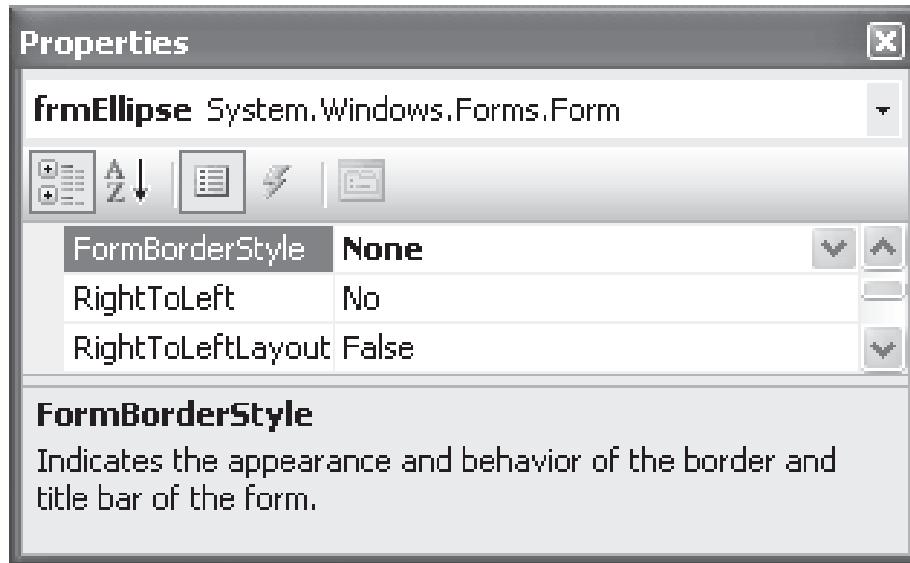


Figure 16.1: Setting FormBorderStyle property

3. Generate an event handler for **frmEllipse \_ Load** by double-clicking the form title in design view.

4. Type the code given in Code Snippet in the **Load** event handler..

**Code Snippet :**

```
private void frmEllipse_Load(object sender, EventArgs e)
{
    this.BackColor = Color.Coral;
    System.Drawing.Drawing2D.GraphicsPath path = new
        System.Drawing.Drawing2D.GraphicsPath();
    path.AddEllipse(0, 0, this.Width, this.Height);
    this.Region = new Region(path);
}
```

The **GraphicsPath** class is defined in the **System.Drawing.Drawing2D** namespace. This namespace contains classes that can render two-dimensional effects and shapes. The **GraphicsPath** class enables you to represent a series of connected lines and curves. In this example, it has been used to create and add an ellipse.

5. Add a button to close the form without which the user may not be able to close the form. Add the code given in Code Snippet to the **Click** event of the button.

**Code Snippet :**

```
private void btnClose_Click(object sender, EventArgs e)
{
    this.Close();
}
```

If required, you can also add functionality to resize or move the form.

Figure 16.2 shows the output of the example.

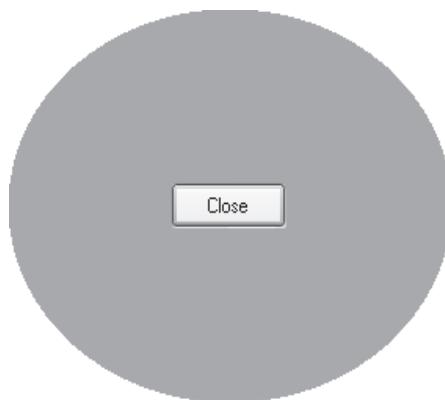


Figure 16.2: Output

You can also specify a bitmap as the background image for the form in order to create a nonrectangular Windows Form.

The steps to do this are described as follows:

1. Create a bitmap image representing the form shape. Ensure that the background color of the bitmap is easy to remember.
2. Set the **BackgroundImage** property of the form to the bitmap as shown in figure 16.3.

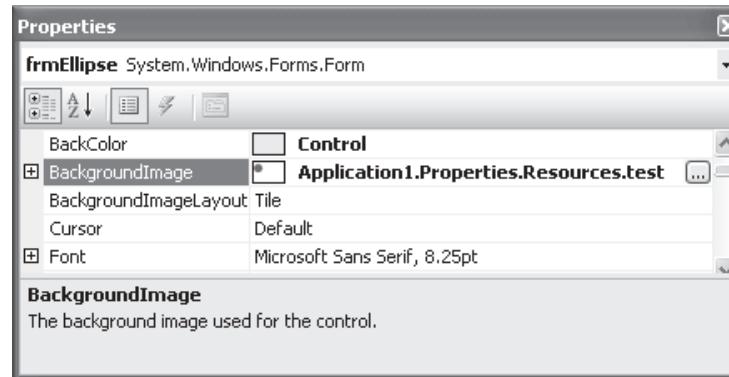


Figure 16.3: Setting **BackgroundImage** property

3. Set the **TransparencyKey** property to the background color of the bitmap as shown in figure 16.4.

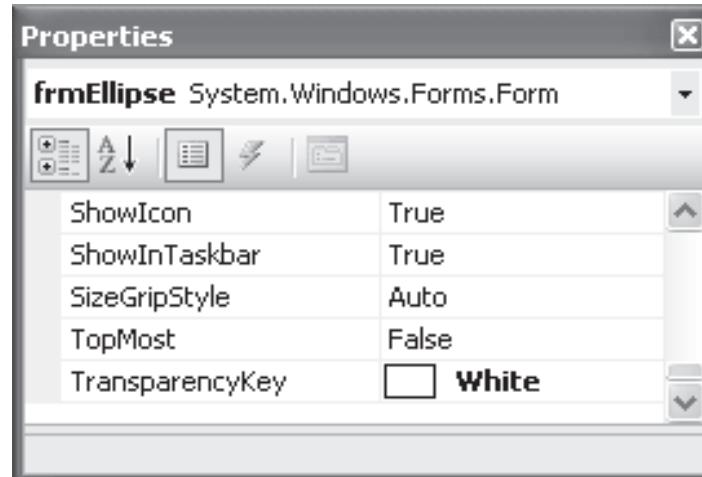


Figure 16.4: Setting **TransparencyKey** property

Figure 16.5 shows the output.

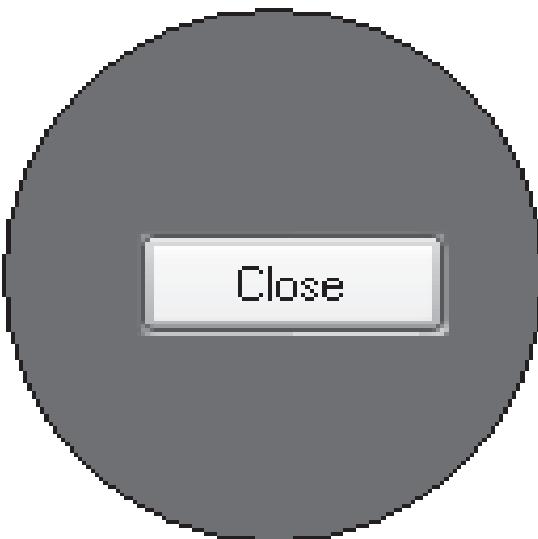


Figure 16.5: Output

Thus, in this manner, a bitmap is specified as the background image for the form in order to create a nonrectangular Windows Form.

**Note** - When your Windows Forms applications involve custom drawing, it is recommended that you test them on a variety of video cards so that satisfactory performance is obtained before deploying them to users.

### 16.3 The TabControl and TrackBar controls

The TabControl and TrackBar controls provided by Windows Forms are very useful in designing efficient user interfaces. This section explores the purpose and working of these controls.

#### 16.3.1 TabControl Control

Consider a scenario of an application which accepts and stores student data. The application should accept the student's personal as well as academic information such as course name, duration, and so on. If the controls to accept data for all the information are placed together, the form will look cluttered and disorganized. The solution for this is to use tabs. Using tabs, you can place a group of controls on a page-like view.

The TabControl control is a container control that is used to group controls on a form into tabs.

The TabControl control enables you to create tabbed pages or views, such that each view is like a container holding one or more child elements and each view is distinct from the other. The advantage is that even if you have many tabbed items, only one will be visible at a time. This saves screen space and makes the form look organized.

A TabControl can contain one or more TabPage controls, each of which represents a tabbed page. A user can change the current TabPage by clicking one of the tabs in the control. Within each TabPage,

you can have controls such as `TextBox`, `Label`, and so forth. A `TabPage` control is a special type of container control that is hosted only inside a `TabControl`. You cannot create a standalone `TabPage` on a form. A `TabPage` control can generate scroll bars as needed if its `AutoScroll` property is set to `True`.

Table 16.1 shows the commonly used properties, methods, and events of `TabControl`.

Property	Description
Appearance	Specifies or retrieves the visual style of the <code>TabControl</code> Can have one of three values: <code>Normal</code> , <code>Buttons</code> , or <code>FlatButtons</code>
Alignment	Specifies or retrieves the alignment of the tabs such as <code>Top</code> , <code>Bottom</code> , <code>Left</code> , or <code>Right</code>
Multiline	Specifies or retrieves whether more than one row of tabs is allowed on the <code>TabControl</code> When set to <code>True</code> , multiple rows of tabs are supported
TabPages	Specifies or retrieves the collection of <code>TabPage</code> controls in the <code>TabControl</code>
Method	Description
DeselectTab	Specifies or retrieves the tab that is following the given tab as the current tab An index or name of the tab is given as the argument
SelectTab	Specifies or retrieves the tab with the given index or name as the current tab
Event	Description
Deselected	Raises when a tab is deselected
Selected	Raises when a tab is selected
SelectedIndexChanged	Raises when the <code>SelectedIndex</code> property has changed

Table 16.1: Members of `TabControl`

The `TabPages` property of the `TabControl` enables you to specify the collection of `TabPage` controls that will be contained within the `TabControl`. Each `TabPage` has its own set of properties.

To access these properties, click the ellipsis next to the TabPages property of the TabControl as shown in figure 16.6

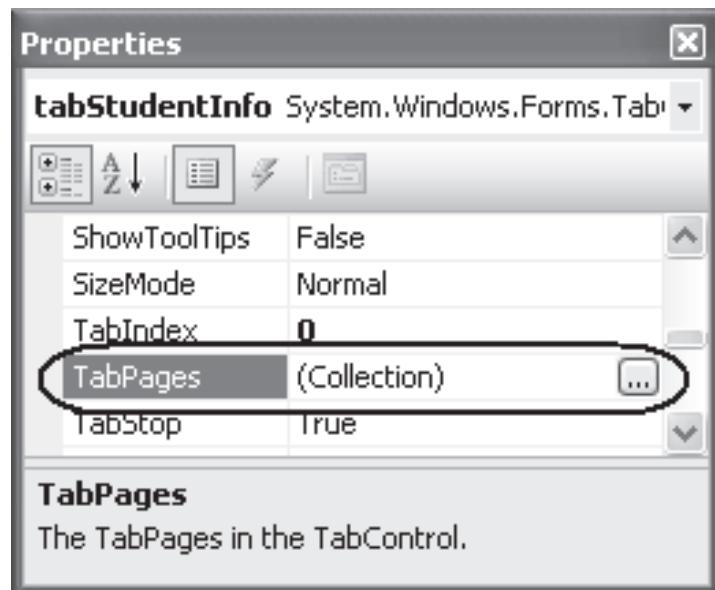


Figure 16.6: TabPages property of the TabControl

This will launch the **TabPage Collection Editor** dialog box as shown in figure 16.7.

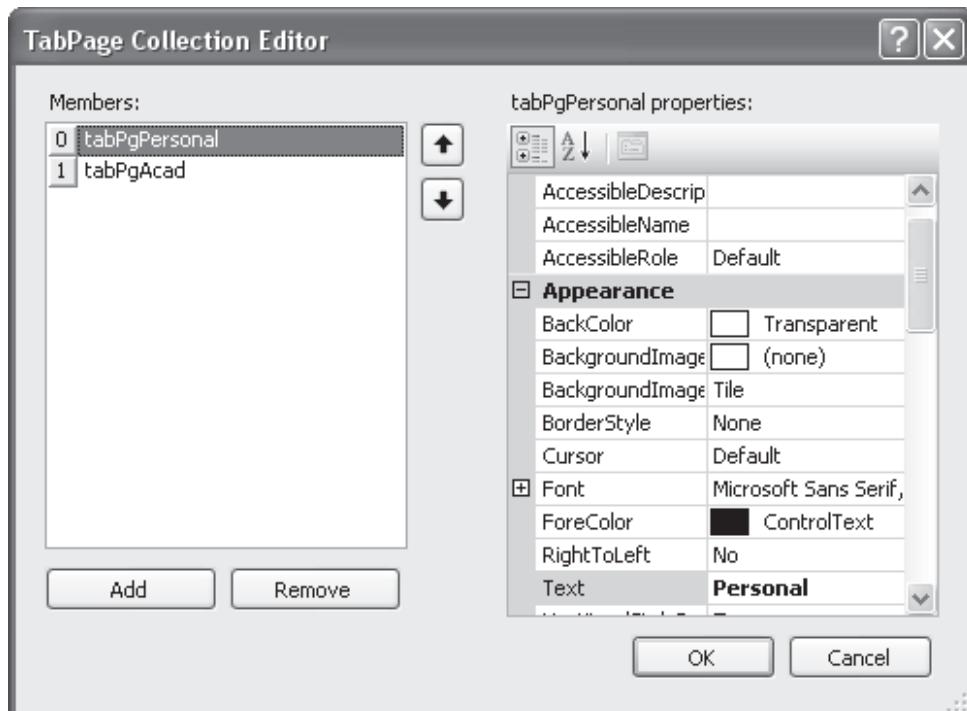


Figure 16.7: TabPage Collection Editor

Table 16.2 shows the commonly used properties of TabPage.

Property	Description
AutoScroll	Specifies whether scroll bars will be displayed or not when controls are hosted outside the visible bounds  When set to True, scroll bars are displayed and when set to False, they are not displayed
BorderStyle	Specifies or retrieves the visual appearance of the TabPage border  Can be set to None indicating no border; FixedSingle, indicating a single-line border; or Fixed3D, indicating a border with a three-dimensional appearance
Text	Specifies or retrieves the text displayed on the tab in the TabControl for the TabPage

Table 16.2: Properties of TabPage

The complete step by step procedure to implement the scenario described earlier is as follows:

1. Drag and drop the **TabControl** from Toolbox on to the form.
2. Set its properties as shown in table 16.3.

Property	Value
Appearance	Normal
Alignment	Top
Multiline	False

Table 16.3: Properties of TabControl

3. Click the ellipsis next to the **TabPage**s property in the Properties window.
4. Set the properties of the first **TabPage** as shown in table 16.4.

Property	Value
Name	tabPgPersonal
Text	Personal
AutoScroll	True
BorderStyle	FixedSingle

Table 16.4: Properties of First TabPage

5. Set the properties of the second **TabPage** as shown in table 16.5.

Property	Value
Name	tabPgAcad
Text	Academic
AutoScroll	True

Property	Value
BorderStyle	FixedSingle

Table 16.5: Properties of Second TabPage

6. Add controls such as **TextBox**, **Label**, and so on to the individual **TabPage**s as shown in figure 16.8.

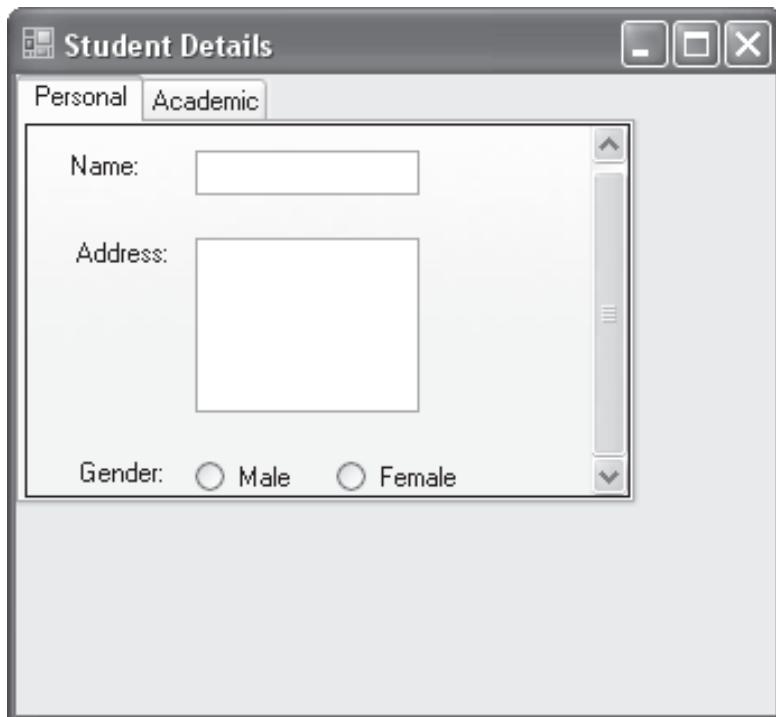


Figure 16.8: Output

The **SelectedIndexChanged** event can be used to run some action when a particular tab is clicked. For example, Code Snippet displays a message box when a tab is clicked.

#### Code Snippet :

```
private void btnClose_Click(object sender, EventArgs e)
{
    this.Close();
}

MessageBox.Show("You have selected Tab " +
    tabStudentInfo.SelectedIndex.ToString(), "Info",
    MessageBoxButtons.OK, MessageBoxIcon.Information);
}
```

Consider that you want to display the TabControl with a specified tab page already selected. You want to do this depending on user selection. To achieve this, a pair of option buttons can be used to accept the choice. Thereafter, code can be written to display the selected tab.

Add two option buttons (radio buttons) as shown in figure 16.9 so that you can display a tab depending on user selection.

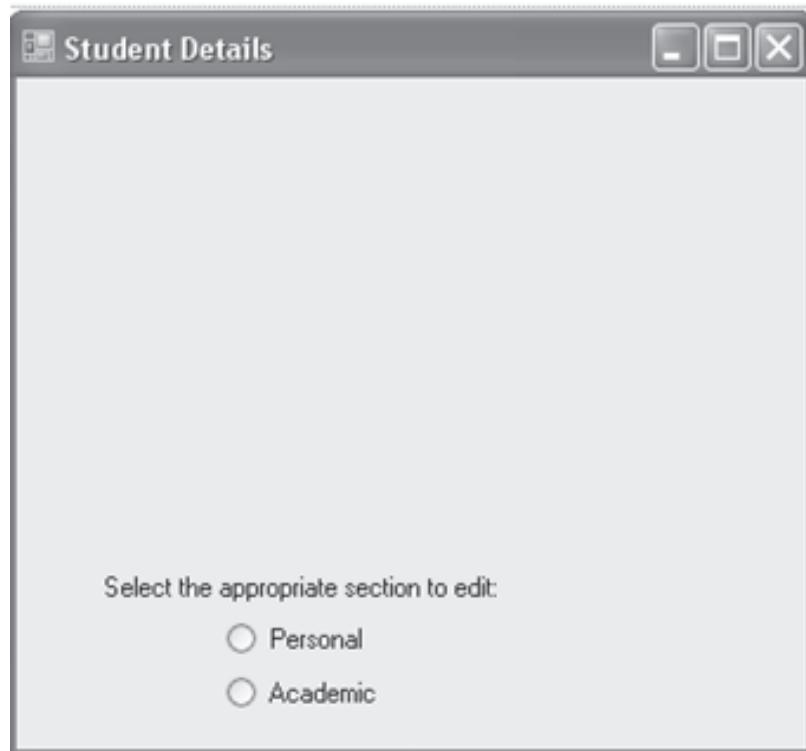


Figure 16.9: Output

The code to display a tab depending on user selection is shown in **Code Snippet**. These statements use the methods of TabControl.

#### Code Snippet:

```
private void radPersonal_CheckedChanged(object sender, EventArgs e)
{
    tabStudentInfo.SelectTab("tabPgPersonal");
}

private void radAcad_CheckedChanged(object sender, EventArgs e)
{
    tabStudentInfo.SelectTab("tabPgAcad");
}
```

Here, the `SelectTab` method of the `TabControl` is used to make the specified tab as the selected tab. The output is shown in figure 16.10.

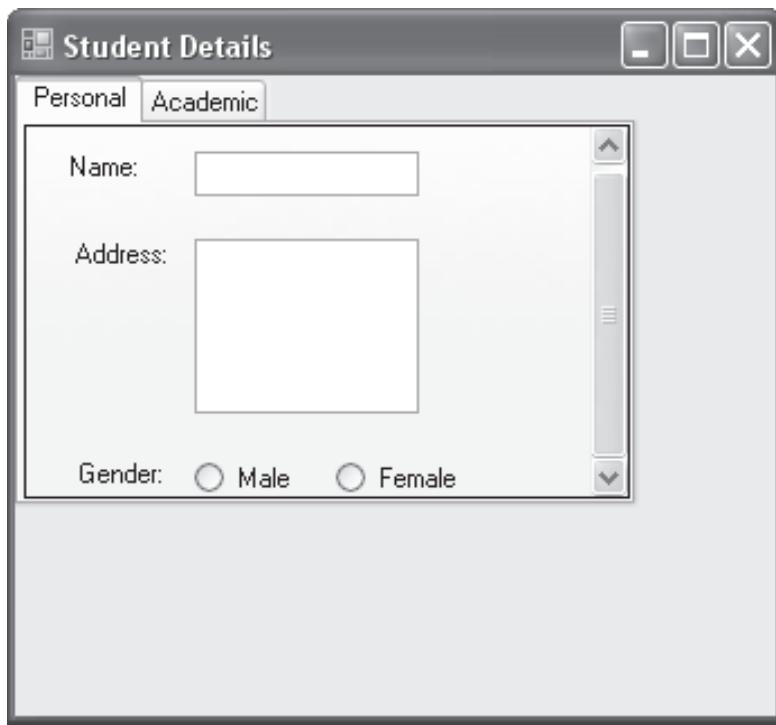


Figure 16.10: Output

Depending on the button clicked by the user, the specific tab will be opened at runtime in the `TabControl`.

### 16.3.2 *TrackBar Control*

Consider a scenario where you have a text box displaying some text and you want to change the size of the font for the text depending upon user's selection. To implement this, you can make use of the `TrackBar` control in Windows Forms. This control provides a slider-like interface that allows you to set a value by scrolling through the slider using either mouse or keyboard. The value that is set must be within a given range of values.

The properties of the `TrackBar` control are listed in table 16.6.

Property	Description
<code>LargeChange</code>	Specifies or retrieves the number of positions by which the slider moves in response to mouse clicks or the Page Up and Page Down keys
<code>Maximum</code>	Specifies or retrieves the maximum value for the <code>TrackBar</code>
<code>Minimum</code>	Specifies or retrieves the minimum value for the <code>TrackBar</code>
<code>SmallChange</code>	Specifies or retrieves the number of positions by which the slider moves in response to arrow keys

Property	Description
TickFrequency	Specifies or retrieves the number of positions between tick marks on the TrackBar
TickStyle	Indicates where ticks appear on the TrackBar. The possible options are None, TopLeft, BottomRight, and Both
Value	Specifies or retrieves the value returned by the TrackBar

Table 16.6: Properties of TrackBar Control

An example of using the **TrackBar** control is shown in figure 16.11.

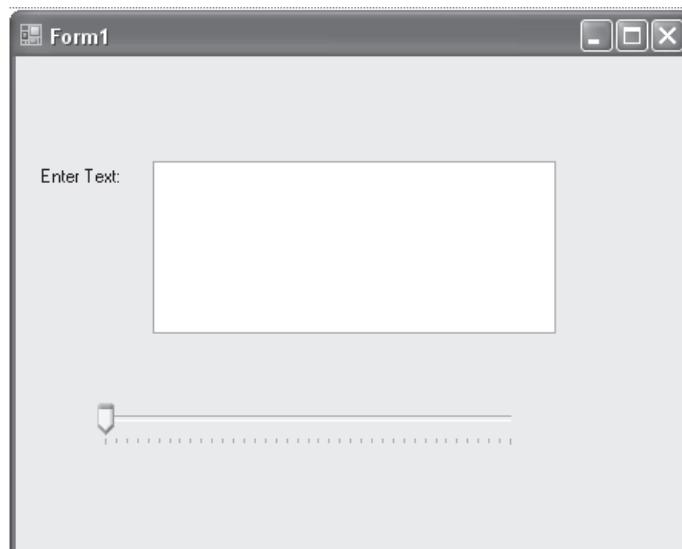


Figure 16.11: TrackBar Control Example

The steps for creating this application are as follows:

1. Drag and drop the **TrackBar** from Toolbox on to the form.

2. Set its properties as shown in figure 16.12.

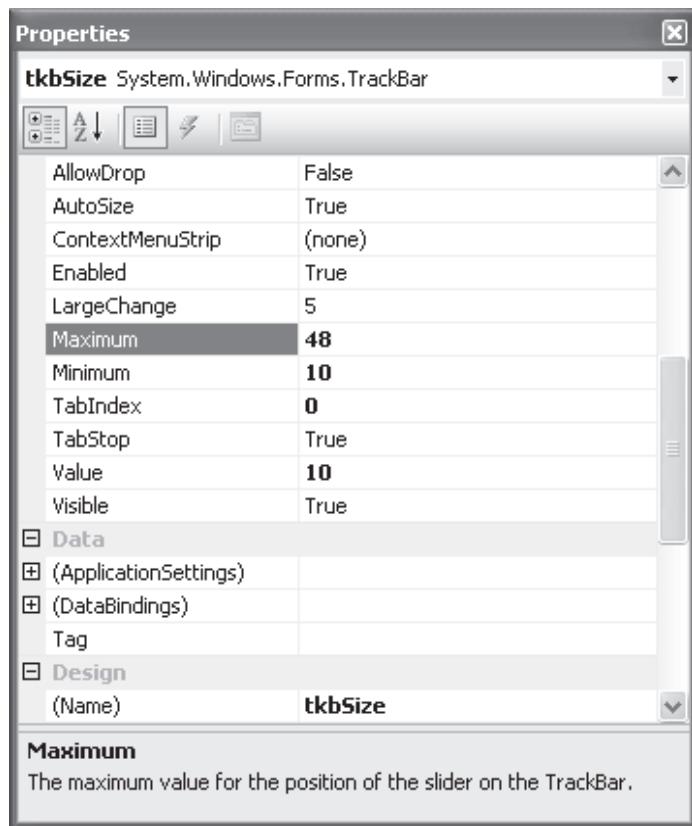


Figure 16.12: Setting Properties

3. Generate an event handler for **Scroll** event by double-clicking the event name.  
 4. Add the code given in Code Snippet to the **Scroll** event handler.

**Code Snippet :**

```
private void tkbSize_Scroll(object sender, EventArgs e)
{
    float value = tkbSize.Value;
    txtText.Font = new Font("Times New Roman", value);
}
```

The code is written so that during execution, when the slider on the trackbar is dragged, the size of the font increases according to the Value of the TrackBar.

The output is shown in figure 16.13.

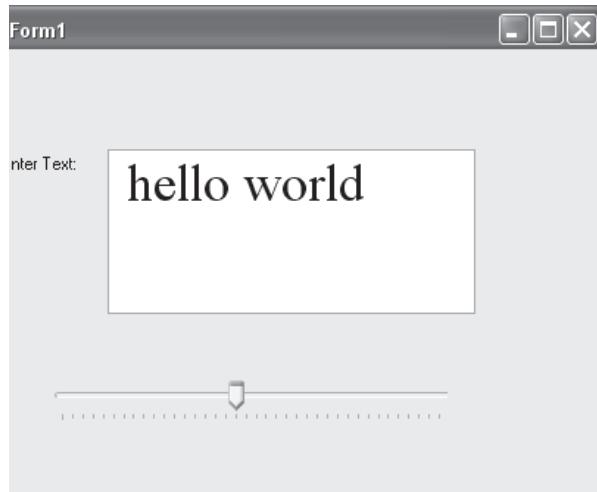


Figure 16.13: Output

## 16.4 The NotifyIcon Component

Often, you see icons in the notification area or system tray for shortcuts to processes such as a virus protection program, software updates, or a volume control. These processes run in background mode. If you create a Windows Forms application that has a process running in the background, you need some mechanism to notify users about information regarding it. The `NotifyIcon` class provides such a mechanism and enables to notify users when processes are running in the background.

The `NotifyIcon` component is not a control. Balloon tips, icons, and a `ContextMenuStrip` can be used with the `NotifyIcon` to provide notifications.

Table 16.7 shows some of commonly used properties of the `NotifyIcon` component.

Property	Description
<code>BalloonTipIcon</code>	Specifies or retrieves the icon that will be shown in the balloon tip. This can be set to <code>None</code> , which displays no icon, or to <code>Info</code> , <code>Warning</code> , or <code>Error</code>
<code>BalloonTipText</code>	Specifies or retrieves the text that is displayed in the balloon tip
<code>BalloonTipTitle</code>	Specifies or retrieves the title of the balloon tip
<code>Icon</code>	Specifies or retrieves the icon that is shown in the system tray
<code>Text</code>	Specifies or retrieves the text that is shown when the user's mouse hovers on the icon in the system tray
<code>Visible</code>	Indicates whether the icon is visible in the system tray

Table 16.7: Properties of the `NotifyIcon` component

The following example shows how to make use of the `NotifyIcon` component:

1. Drag and drop the `NotifyIcon` from Toolbox on to the form.

2. Set its properties as shown in figure 16.14.

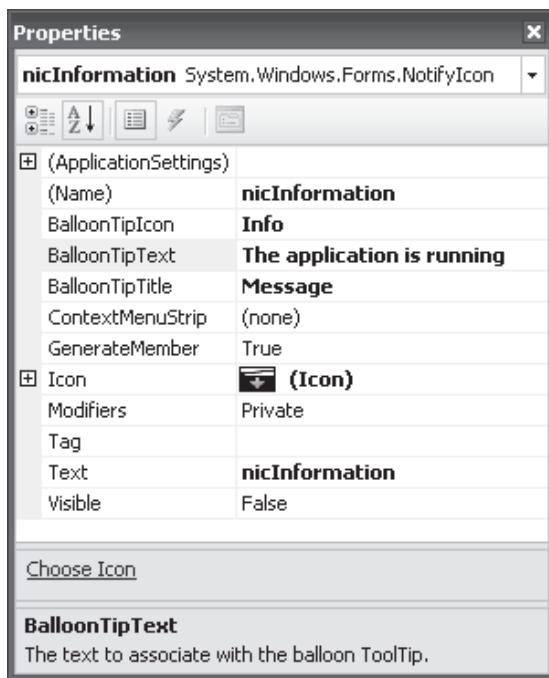


Figure 16.14: Setting the Properties of the component

The Icon property has been set to the icon to be displayed, the Visible property is set to False. To add icons to your application project, create a new instance of the System.Drawing.Icon class or add existing icon files by choosing Add Existing Item from the Project menu. In the current example, the latter approach has been chosen.

The BalloonTipIcon, BalloonTipText, and BalloonTipTitle properties have been set to Info, The application is running, and Message respectively.

After the appropriate properties are set, you can display the balloon tip by calling the ShowBalloonTip method. A parameter is passed to the ShowBalloonTip method that indicates the number of seconds for which the balloon tip is shown.

Assume that you have created a button named btnOK in the application. Clicking the button should result in the notification being shown.

Add the code given in Code Snippet to the event handler for the button.

#### Code Snippet:

```
private void btnOK_Click(object sender, EventArgs e)
{
    nicInformation.Visible = true;
    nicInformation.ShowBalloonTip(5);
}
```

Figure 16.15 shows the output.



Figure 16.15: NotifyIcon in action

In the code, the `NotifyIcon` component is set to visible and the `ShowBalloonTip` is passed a value of 5 to indicate five seconds as the duration for which the tip will be seen.

If you click the button during program execution, the notification will be shown.

## 16.5 Various types of ToolStripItems

Tool strip items are controls that are designed to be hosted inside a tool strip and enable users a wide variety of options and functionality.

The .NET Framework provides several items designed to be hosted in tool strips.

There are specialized `ToolStripItem` controls named `ToolStripLabel`, `ToolStripButton`, `ToolStripTextBox`, `ToolStripProgressBar`, and `ToolStripComboBox` which are similar to `Label`, `Button`, `TextBox`, `ProgressBar`, and `ComboBox` controls. These controls are designed to be hosted in tool strips and cannot exist on their own.

The `ToolStripSplitButton`, `ToolStripDropDownButton`, and `ToolStripSeparator` controls are designed to provide functionality specific to tool strips.

You can access these items by clicking the ellipsis next to the `Items` property of the `ToolStrip` in the Properties window as shown in figure 16.16.

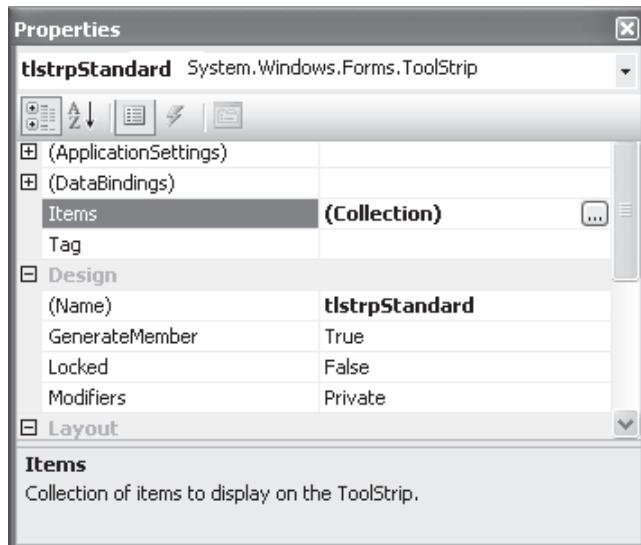


Figure 16.16: Items Property

This will display the Items Collection Editor.

Figure 16.17 shows various kinds of tool strip items that can be created through the **Items Collection Editor**.

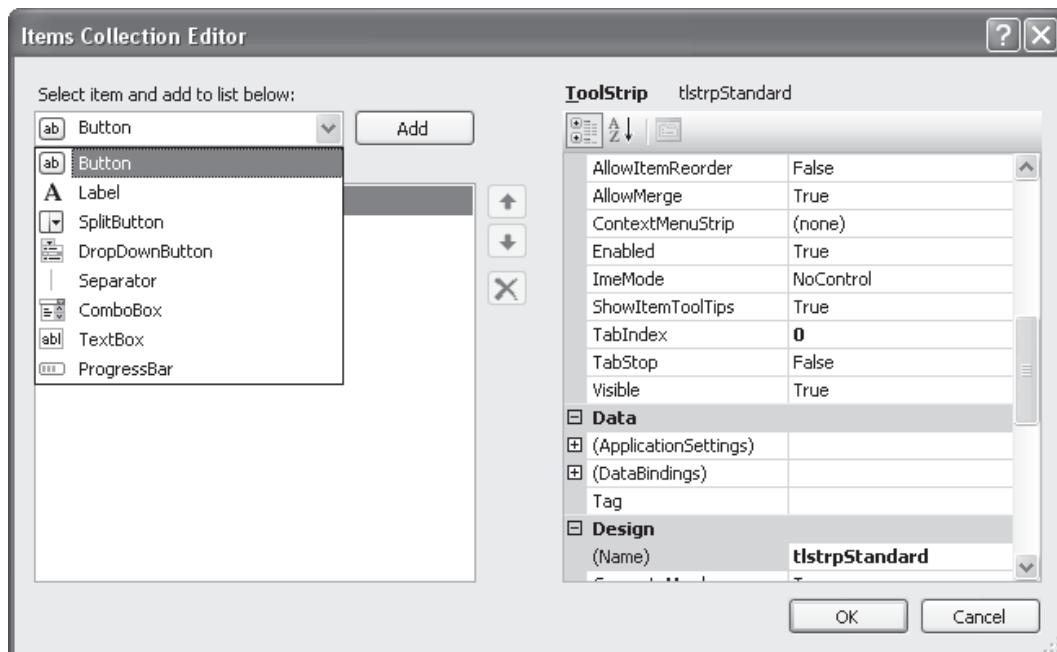


Figure 16.17: Items Collection Editor

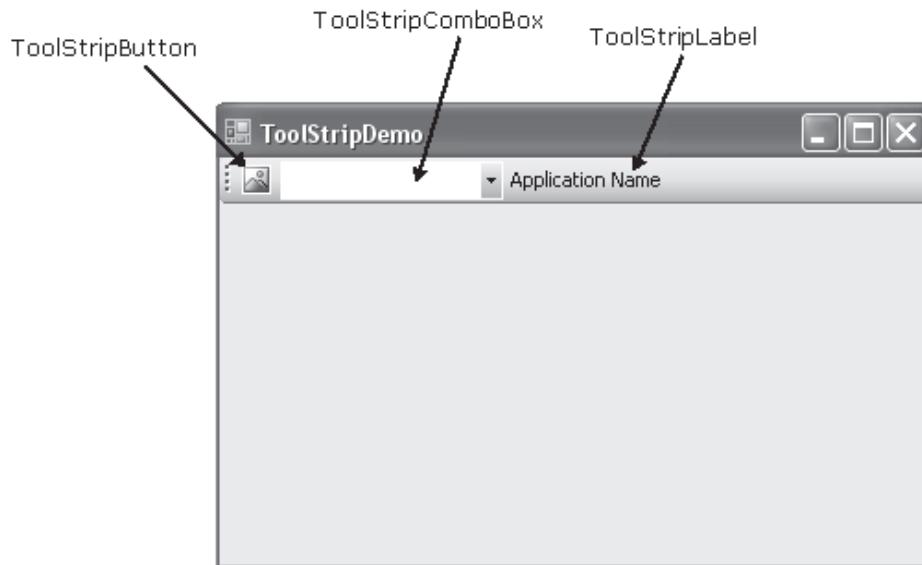
The various types of ToolStripItem controls are listed in table 16.8.

ToolStripItem	Description
ToolStripLabel	Displays a text label
ToolStripButton	Displays a button
ToolStripSeparator	Separates the items in a tool strip from one another
ToolStripComboBox	<p>Behaves similar to the ComboBox control and can be set to styles such as Simple, DropDown, or DropDownList</p> <p>Items are found in the Items collection</p> <p>The Text property retrieves the item that is selected or typed into the ToolStripComboBox</p>
ToolStripTextBox	<p>Behaves similar to the basic TextBox control</p> <p>Enables users to type a string into the text box</p> <p>ToolStripTextBox does not have a MultiLine property and can have only one line</p> <p>This is the main difference between TextBox and ToolStripTextBox</p>
ToolStripProgressBar	<p>Behaves similar to the standard ProgressBar control</p> <p>Is a control designed to provide feedback to the user about the progress of a time-consuming task</p>
ToolStripDropDownButton	Creates a drop-down menu
ToolStripSplitButton	Combines the functionality of the ToolStripButton and ToolStripDropDownButton controls

Table 16.8: Types of ToolStripItems

The ToolTipText property is common to all these controls except ToolStripSeparator. If the ToolStrip.ShowItemToolTips property is set to True, then, ToolTipText specifies or retrieves the text shown in a tooltip during mouse hover on the tool strip item.

Figure 16.18 shows a ToolStripButton, a ToolStripComboBox, and a ToolStripLabel created on a ToolStrip control.



**Figure 16.18: ToolStripItems**

The toolbars that you see on various applications often have image icons on the tool buttons. If you wish to display a similar toolbar, you can associate images with the ToolStripButton, ToolStripDropDownButton, and ToolStripSplitButton controls.

Table 16.9 lists some of the properties that can be set to achieve this.

Property	Description
DisplayStyle	Determines whether the control is displayed with text, image, or both
Image	Specifies or retrieves the image associated with this control
ImageAlign	Indicates how the image is aligned in the control

**Table 16.9: Image properties**

Consider an example to display an image on a ToolStripButton strip.

Select the ToolStripButton control. In the Properties window, ensure that the `DisplayStyle` property is set to `Image`.

Select the image for the control by clicking the `Image` property and selecting or browsing to the appropriate image in the **Select Resource** dialog box as shown in figure 16.19.

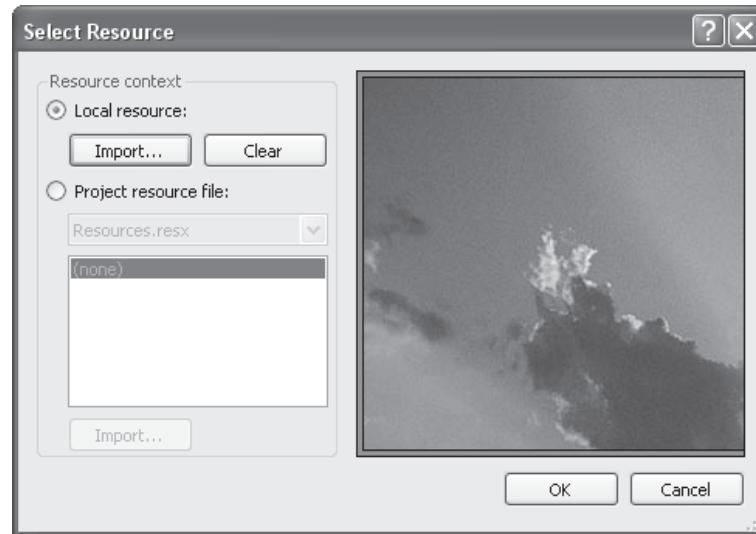


Figure 16.19: Select Resource Dialog Box

Figure 16.20 shows the output with the specified image on the button control.



Figure 16.20: Output

## 16.6 Merging Tool Strips

You can merge `ToolStrip` controls at run time and consolidate their items on one tool strip.

The `ToolStripManager.Merge()` method must be called to merge `ToolStrip` controls.

Its basic syntax is shown here:

### Syntax:

```
ToolStripManager.Merge(sourceToolStrip, targetToolStrip);
```

where,

`ToolStripManager`: A static class managing the display and layout of the tool strips on a form

`sourceToolStrip`: The first tool strip to be merged

`targetToolStrip`: The second tool strip with which the first tool strip is to be merged

The tool strip items on `sourceToolStrip` are merged with the items on `targetToolStrip` based on their `MergeAction` property value.

Table 16.10 displays some of the merge actions that are taken.

MergeAction Value	Action taken
Append	Appends the item at the end of the list of items
Insert	Inserts the item at the location specified by the <code>MergeIndex</code> property
MatchOnly	Looks for a match but takes no action
Remove	If a matching tool strip item is found, it is removed from the resulting tool strip

Table 16.10: Merge Actions

The primary condition for this is that the `ToolStrip` controls must have their `AllowMerge` property set to `True` as shown in figure 16.21.

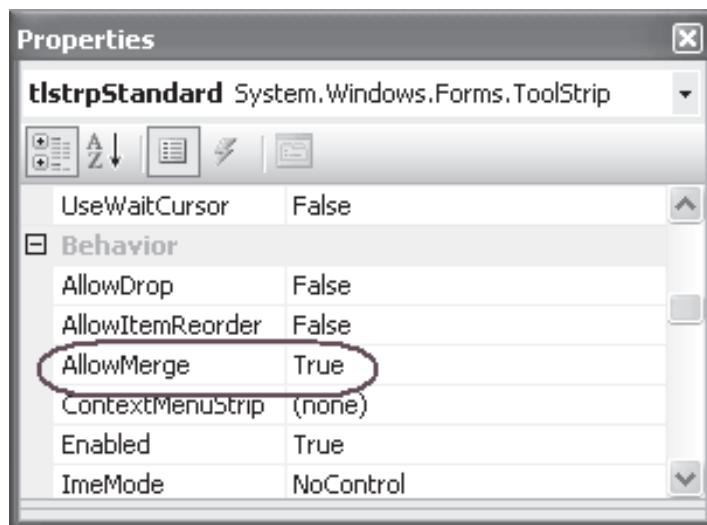


Figure 16.21: AllowMerge Property

Table 16.11 lists the properties of `ToolStripItem` used for merging. This could be any tool strip item such as `ToolStripLabel`, `ToolStripButton`, `ToolStripTextBox`, `ToolStripProgressBar`, and `ToolStripComboBox`.

Property	Description
<code>MergeAction</code>	Specifies the action that takes place on a tool strip when it is merged with another tool strip One of the following must be specified: Append, Insert, MatchOnly, Remove, and Replace
<code>MergeIndex</code>	Indicates the position of a tool strip item in a merged tool strip if the <code>MergeAction</code> property is set to Insert

Table 16.11: Properties used for merging

Figure 16.22 shows an example of this.

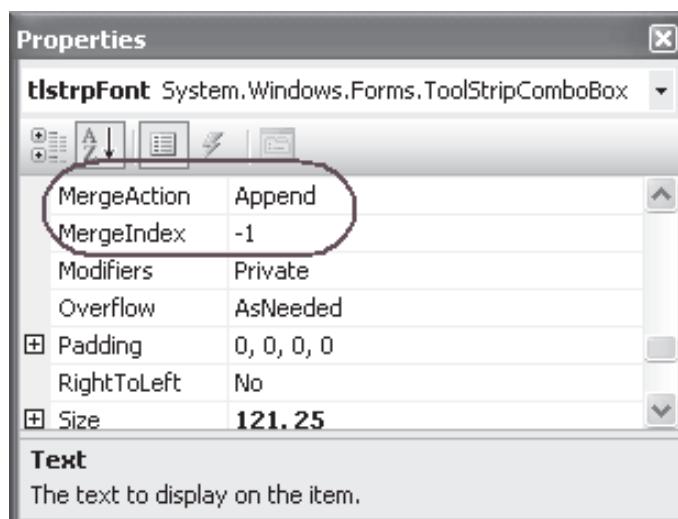


Figure 16.22: `MergeAction` and `MergeIndex` Properties

When two tool strips are merged, the following actions take place:

1. Each tool strip item in the source tool strip is compared to each tool strip item in the target tool strip.
2. If any two tool strip items have the same `Text` property, they are considered a match, regardless of their types. Consider a `ToolStripLabel` and a `ToolStripButton` having `Text` property as 'Execute'. They are considered a match.
3. If a match is found, the `MergeAction` property of the source tool strip item is analyzed.
4. If the source tool strip item has the `MergeAction` property set to `MatchOnly`, `Remove`, or `Replace`, then, relevant action is taken.
5. Otherwise, depending on the `MergeAction` property, the tool strip item is appended or inserted.

Consider an example. Two tool strips, tlstrpFormatting and tlstrpStandard are created as shown in figure 16.23.

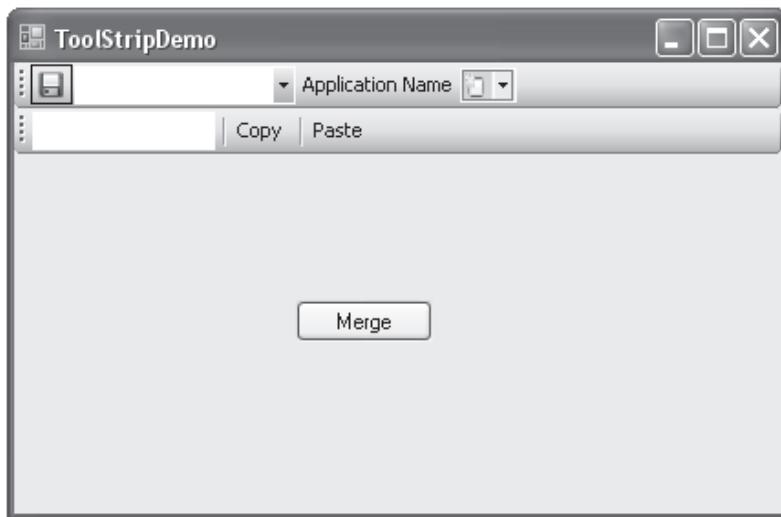


Figure 16.23: Example

Create a button named btnMerge. This button when clicked should merge the two ToolStrip controls. The code for the same is given in Code Snippet.

#### Code Snippet :

```
private void btnMerge_Click(object sender, EventArgs e)
{
    ToolStripManager.Merge(tlstrpFormatting, tlstrpStandard);
}
```

Assume that the ToolStripItems named Copy and Paste have their MergeAction property set to Insert and their MergeIndex property set to 1 and 2 respectively. The rest of the ToolStripItems have a default MergeAction of Append. As a result of this, when the button is clicked during execution, the two ToolStrips are merged such that Copy and Paste are now placed next to the Save button. This is because of the MergeAction and MergeIndex properties.

The output would be as seen in figure 16.24.



Figure 16.24: Output

## 16.7 The ToolStripContainer class

The ToolStripContainer class, as its name suggests, is used to contain tool strips. It supports rafting, the process by which users can drag and drop a tool strip from one edge of the container to another.

The ToolStripContainer contains five panels: four ToolStripPanels (one on each edge of the form) and one ContentPanel respectively.

Figure 16.25 shows the The ToolStripContainer.

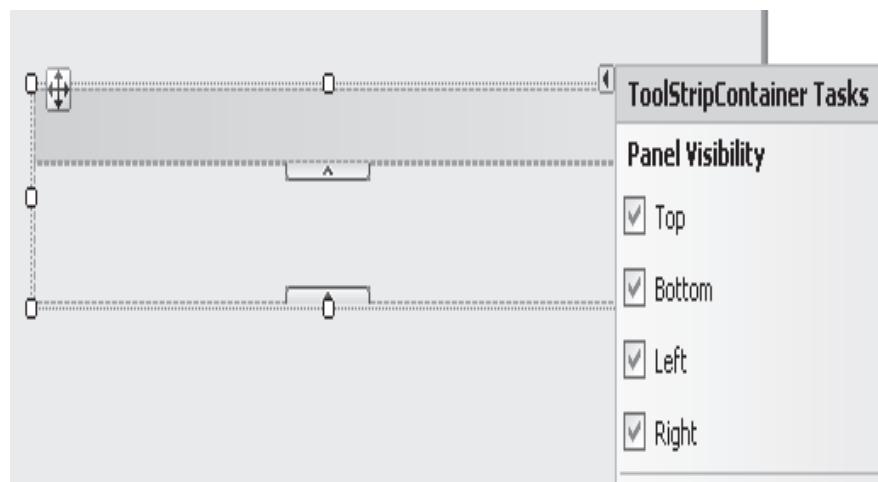


Figure 16.25: Output

You can control the availability of the tool strip panels depending on your requirements. This can be done using one of two ways:

- By setting the following properties: `TopToolStripPanelVisible`, `BottomToolStripPanelVisible`, `LeftToolStripPanelVisible`, and `RightToolStripPanelVisible`
- By checking the check boxes for the `ToolStripPanels`

When you set the `Dock` property of the `ToolStripContainer` to `Fill`, it causes the `ToolStripContainer` to fill the entire form. The tool strip panels would be available on all four sides.

## 16.8 Custom Dialog Boxes

Windows Forms provides many built-in dialog boxes such as `ColorDialog`, `OpenFileDialog`, `SaveFileDialog`, and so forth. It is also possible to create custom dialog boxes in Windows Forms.

Consider a scenario where you need to develop a `CopyFileDialog` similar to the built-in `SaveFileDialog`. Since, Windows Forms does not have any provision for such a dialog box, you can create your own.

A dialog box may include buttons such as `OK`, `Cancel`, as well as controls such as labels, text boxes, and so forth to obtain information from the user.

Consider an example to understand the concept of custom dialog boxes.

1. Click **Project**→**Add Windows Form**.
2. Type the name as 'DialogBox.cs'. Rename the form to `frmDialogBox` and specify `Text` as `DialogBox`.
3. Set the `FormBorderStyle` property in the Properties window to `FixedDialog`.
4. Set the `ControlBox`, `MinimizeBox`, and `MaximizeBox` properties to `False`.
5. Add controls as shown in figure 16.26.

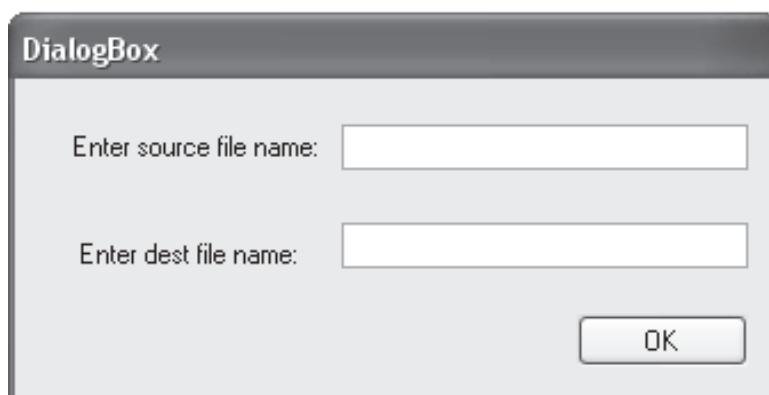


Figure 16.26: Custom Dialog Box

6. Set the `Button.DialogResult` property to `OK`.
7. Create a new form in the project and rename it as `frmMain`. Therefore, your application now has two forms – `frmMain` and `frmDialogBox`.
8. Generate `Load` event for the form, `frmMain`, and add the code shown in Code Snippet to the event handler.

**Code Snippet:**

```
private void frmMain_Load(object sender, EventArgs e)
{
    frmDialogBox dlg;
    DialogResult result;
    dlg = new frmDialogBox();
    result = dlg.ShowDialog();
    if (result == DialogResult.OK)
    {
        MessageBox.Show("Clicked OK");
    }
}
```

The `ShowDialog()` method is used to return the  `DialogResult`  value of the button that was clicked to close the form. If the `Button.DialogResult` property is not set, it returns  `DialogResult.None` .

You can set the `ParentForm` property of the dialog box by specifying the parent form as a parameter to the `ShowDialog()` method.

Modify the code shown earlier in Code Snippet as shown here in Code Snippet.

**Code Snippet:**

```
frmDialogBox dlg;
DialogResult result;
...
result = dlg.ShowDialog(this);
```

It is recommended that the information entered in the dialog box should be accessed through properties of the dialog box.

For example, a dialog box that allows the user to input a source file name as shown in figure 16.26 should allow access to that information through a `SourceFileName` property.

Code snippet creates a property called `SourceFileName` for the dialog box that takes user input from a

text box called `txtSourceFileName`.

**Code Snippet:**

```
public string SourceFileName
{
    get
    {
        return txtSourceFileName.Text;
    }
}
```

You can then retrieve the property of the dialog box to get the source file name as shown in Code Snippet :

**Code Snippet:**

```
string name;
name = frmDialogBox.SourceFileName;
```

## 16.9 Configuring the MonthCalendar control

Windows Forms applications frequently make use of date and time controls such as `DateTimePicker` and `MonthCalendar`.

The `MonthCalendar` control not only enables a user to select a particular date but also allows selecting a range of dates.

Table 16.12 shows the important selection properties of the `MonthCalendar` control.

Property	Description
<code>MaxSelectionCount</code>	Specifies or retrieves the total number of days that can be selected for the control
<code>SelectionRange</code>	Specifies or retrieves the range of dates selected by the user
<code>SelectionStart</code>	Specifies or retrieves the starting date and time of the <code>SelectionRange</code> property
<code>SelectionEnd</code>	Specifies or retrieves the ending date and time of the <code>SelectionRange</code> property

Table 16.12: Selection Properties of `MonthCalendar` control

To select a range of dates at design time, you must hold down the **Shift** key while clicking the starting date and the ending date. The range of dates that is selected must not be greater than the `MaxSelectionCount` property indicates.

The steps to use the selection properties of `MonthCalendar` control are described with an example:

1. Drag and drop a **MonthCalendar** control from the Toolbox onto the form.

Figure 16.27 shows the default selection values for the control.

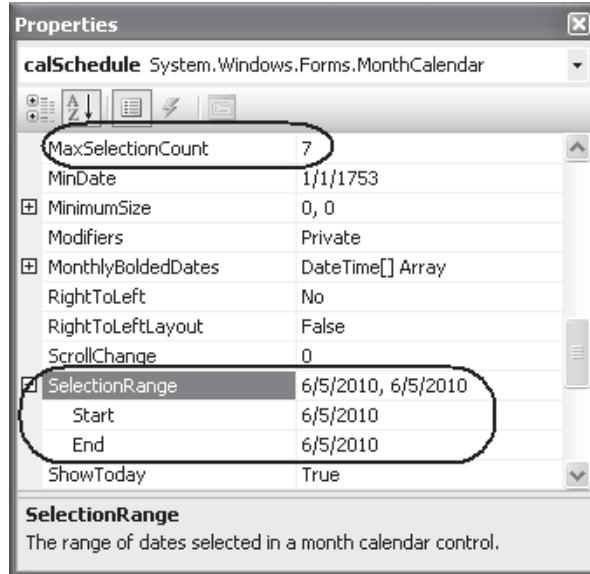


Figure 16.27: Selection Properties of MonthCalendar Control

2. Design the rest of the form as shown in figure 16.28.

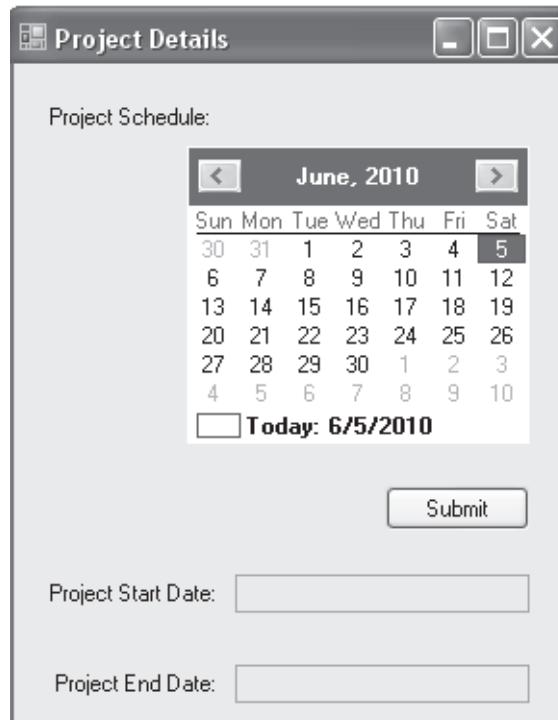


Figure 16.28: Form Design

3. Set the text boxes for Project Start Date and Project End Date to read-only.
4. Build and execute the application.

5. Use the **SelectionStart** and **SelectionEnd** properties of **MonthCalendar** control to determine at runtime the selection made by the user and assign them to the text boxes. Code Snippet shows how to achieve this.

**Code Snippet:**

```
private void btnSubmit_Click(object sender, EventArgs e)
{
    txtStartDate.Text = calSchedule.SelectionStart.ToString();
    txtEndDate.Text = calSchedule.SelectionEnd.ToString();
}
```

Figure 16.29 shows the output. In order to control the number of days, the user should be able to select and change the value of the **MaxSelectionCount** property. The user can select less days than the **MaxSelectionCount** value but cannot select more than that value.

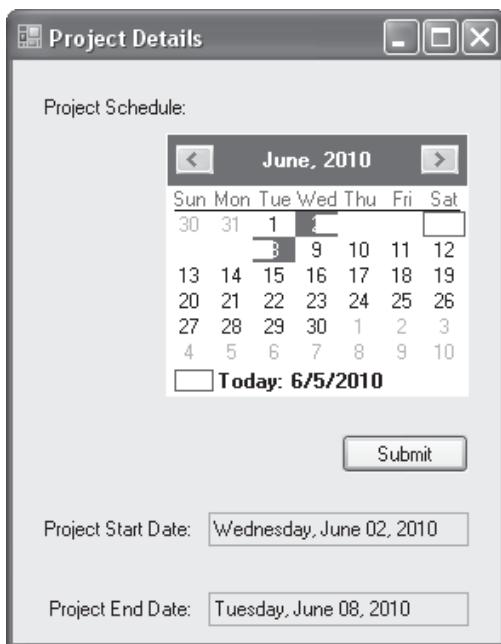


Figure 16.29: Output

## 16.10 Merging DataSets

Consider a scenario where you are working with a **DataSet** containing data. Assume that you now receive another **DataSet** containing updates or records to be added to the first **DataSet**. To combine the data accurately, you will need to use the **Merge()** method of **DataSet** class.

One of the important methods of the **DataSet** class is the **Merge()** method. This method merges a given **DataSet**, **DataTable**, or array of **DataRow** objects into the current **DataSet** or **DataTable**. However, one of the most common operations performed using this method is merging data from one **DataSet** with that in another **DataSet**. During the merge operation, the data from the two **DataSets**

are combined based on whether a similar record exists in the target `DataSet`.

The merge options can be configured to achieve goals such as:

- Adding new records to a `DataSet` by copying them from another `DataSet`
- Copying new records and changes as well as specifying action to be taken if the `DataSets` do not have the same schemas
- Updating the state of records in a `DataSet` based on another

Table 16.13 shows some overloads of the `Merge` method. The `MissingSchemaAction` enumeration determines the action to be taken when data is being added to the `DataSet` and a required `DataTable` or  `DataColumn` is missing.

Name	Description
<code>Merge(DataSet)</code>	Used to merge a specified <code>DataSet</code> and its schema into the current <code>DataSet</code>
<code>Merge(DataSet, Boolean)</code>	Used to merge a specified <code>DataSet</code> and its schema into the current <code>DataSet</code> Retains or discards any changes in the current <code>DataSet</code> depending on the given argument
<code>Merge(DataRow(), Boolean, MissingSchemaAction)</code>	Used to merge an array of <code>DataRow</code> objects into the current <code>DataSet</code> Retains or discards changes in the <code>DataSet</code> and handling schema mismatch depending on the given arguments
<code>Merge(DataSet, Boolean, MissingSchemaAction)</code>	Used to merge a specified <code>DataSet</code> and its schema with the current <code>DataSet</code> Retains or discards changes in the current <code>DataSet</code> and handling schema mismatch depending on the given arguments

Table 16.13: Overloads of the `Merge` method

Table 16.14 lists the various values that the `MissingSchemaAction` parameter can have.

Property	Description
<code>Add (default)</code>	Adds all the columns of the source <code>DataSet</code> to the target <code>DataSet</code> and populated with data
<code>AddWithKey</code>	Adds all the columns and primary key settings of the source <code>DataSet</code> to the target <code>DataSet</code>
<code>Ignore</code>	Ignores the schema variations between the source and target <code>DataSets</code>
<code>Error</code>	Throws an exception, namely <code>InvalidOperationException</code> , when the schemas in the source and target <code>DataSets</code> do not match

Table 16.14: Members of `MissingSchemaAction`

Consider that a DataSet named `StudentDataSet` has been created to store and manipulate student data records. Now, a new DataSet named `NewStudentDataSet` is created which has similar structure as `StudentDataSet` but with one or two different columns. You want to merge the contents of these DataSets. Code Snippet shows how to achieve this.

**Code Snippet:**

```
...
NewStudentDataSet.Merge(StudentDataSet, true,
    MissingSchemaAction.AddWithKey);
```

Here, the contents of `StudentDataSet` are merged into the contents of `NewStudentDataSet`. Any missing columns are added to the `NewStudentDataSet` along with primary key.

## Knowledge Check 1

1. Which of the following statements about merging two **DataSet**s are true?

(A)	It is possible to update the state of records in a <b>DataSet</b> based on another <b>DataSet</b> using the <b>Merge()</b> method of <b>DataSet</b> class.
(B)	The <b>MissingSchemaAction</b> interface determines the action to be taken during merge operations.
(C)	The <b>Ignore</b> option of <b>MissingSchemaAction</b> ignores the schema variations between the source and target <b>DataSets</b> .
(D)	The <b>Exception</b> option of <b>MissingSchemaAction</b> throws an exception, namely <b>InvalidOperationException</b> , when the schemas in the source and target <b>DataSets</b> do not match

2. Which one of the following codes will merge **tlstrpB** into **tlstrpA**?

(A)	<pre>private void btnMerge_Click(object sender, EventArgs e) {     ToolStripManager.Merge(tlstrpA, tlstrpB); }</pre>
(B)	<pre>private void btnMerge_Click(object sender, EventArgs e) {     ToolStripManager.Merge(tlstrpB, tlstrpA); }</pre>
(C)	<pre>private void btnMerge_Click(object sender, EventArgs e) {     ToolStripManager.MergeToolStrip(tlstrpB, tlstrpA); }</pre>
(D)	<pre>private void btnMerge_Click(object sender, EventArgs e) {     ToolStripManager.MergeToolStrip(tlstrpA, tlstrpB); }</pre>

3. Which of the following are valid **MergeAction** values for merging **ToolStrips**?

(A)	Append
(B)	Remove
(C)	MatchOnly
(D)	Match

4. In the snippet given, can you fill in the blank with the right code to display the **NotifyIcon** **nicVirusCheck** for 8 seconds?

```
nicVirusCheck.BalloonTipIcon = System.Windows.Forms.ToolTipIcon.Info;
nicVirusCheck.BalloonTipText = "Warning";
nicVirusCheck.BalloonTipTitle = "Message";
nicVirusCheck.Text = "nicVirusCheck";
nicVirusCheck.Visible = true;
```

(A)	nicVirusCheck.ShowBalloon(8);
(B)	nicVirusCheck.ShowTip(8);
(C)	nicVirusCheck.ShowBalloonTip(8);
(D)	nicVirusCheck.Show(8);

5. Which one of the following properties of the **TabControl** enables you to specify the collection of TabPage controls that will be contained within the **TabControl**?

(A)	TabPage
(B)	TabPages
(C)	TabPageCollection
(D)	TabControlPages

6. The range of dates selected in a **MonthCalendar** control should be less than a specified number of days that is assigned in a property of the **MonthCalendar** control.

Which among the following is that property?

(A)	MaxCount
(B)	SelectionCount
(C)	MaxDays
(D)	MaxSelectionCount

7. Which of the following statements about custom dialog boxes are true?

(A)	You can set the ParentForm property of the dialog box with the ShowDialog method by specifying the parent form as a parameter.
(B)	The FormStyle property of the form should be set to FixedDialog.
(C)	The ControlBox, MinimizeBox, and MaximizeBox properties should be set to False.
(D)	Information can be retrieved from the parent form of a custom dialog box by casting the Form property to the appropriate type and then, reading its properties.

8. Which of the following terms denotes a process by which users can drag and drop a tool strip from one edge of the container to another?

(A)	Rafting
(B)	Drawing
(C)	Dragging
(D)	Moving



## Module Summary

- Windows Forms supports creation of nonrectangular forms.
- The TabControl control is a container control that is used to group controls on a form into tabs.
- The TrackBar control provides a slider-like interface to set a value by scrolling through the slider using either mouse or keyboard.
- The NotifyIcon component represents an icon in the system tray and is usually used with applications that run in the background.
- Windows Forms supports specialized ToolStripItem controls such as ToolStripLabel, ToolStripButton, and ToolStripTextBox.
- You can merge ToolStrip controls at runtime into a single tool strip provided that their AllowMerge property is True.
- You can create custom dialog boxes in Windows Forms by creating a new form and configuring its properties.
- The MonthCalendar control allows a user to select a range of dates, in addition to selecting a single date.
- The Merge() method of DataSet class allows to merge contents of two DataSets.

**GROWTH**  
**RESEARCH**  
**OBSERVATION**  
**UPDATES**  
**PARTICIPATION**



# Module - 17

## Exploring Windows Forms (Lab)

In this Module, you will learn about:

- Create nonrectangular Windows Forms
- Use TabControl and TrackBar controls
- Use ToolStripContainer and ToolStripItems
- Use NotifyIcon Component
- Create and use custom dialog boxes

## Part I – 60 Minutes

### Exercise

GoodReads is a popular book store in Surrey, UK. As the number of books in the store has increased in recent times, the proprietors find it difficult to organize and manage them properly. Hence, they have decided to opt for a software application that will make their day-to-day tasks easier.

A database containing all the book records needs to be created and the software application must provide the means to connect to the database and interact with it. Title, author name, publisher, price, and purchase date of each book will be maintained in the database.

The software application should also have the following features:

- An unusual eye catching entry screen, preferably such as an oval
- A neat user interface, preferably containing tabbed views
- Facility to view all the records from the database through the software application
- Search feature that can locate books based on one of three criteria: title, author, or publisher
- Ability to add new records

Assume that you are hired by the GoodReads proprietors to develop this application. You must use Visual Studio 2008, Windows Forms, and SQL Server 2008 to build this application.

### Solution:

#### Design the GoodReads database and Books table

The details of books must be stored in a database. Hence, create a database named GoodReads by performing the following steps:

1. Open Microsoft SQL Server Management Studio. Enter the appropriate login details.
2. Create a new database named GoodReads.
3. Design a table named Books with the structure specified in table 17.1.

Column Name	DataType	Width	Primary Key
BookID	int		Yes
Title	nchar	40	
Author	nchar	40	
Publisher	nchar	60	
Price	money		
PurchaseDate	date		

Table 17.1: Books table

**4. Insert a record in the table using INSERT command as follows:**

```
INSERT INTO [GoodReads].[dbo].[Books]
( [BookID]
, [Title]
, [Author]
, [Publisher]
, [Price]
, [PurchaseDate])
VALUES (101, 'The Secret', 'Rhonda Bryne', 'Hatchett', 280.00, '2010-10-10')
GO
```

**5. Similarly, insert a few more records into the table.**

### Creating nonrectangular Windows Forms

In order to create an unusual eye catching entry screen such as an oval, the application will utilize nonrectangular Windows Forms.

**1. Launch Visual Studio 2008.**

**2. Create a Windows Forms application named BookStore as shown in figure 17.1.**

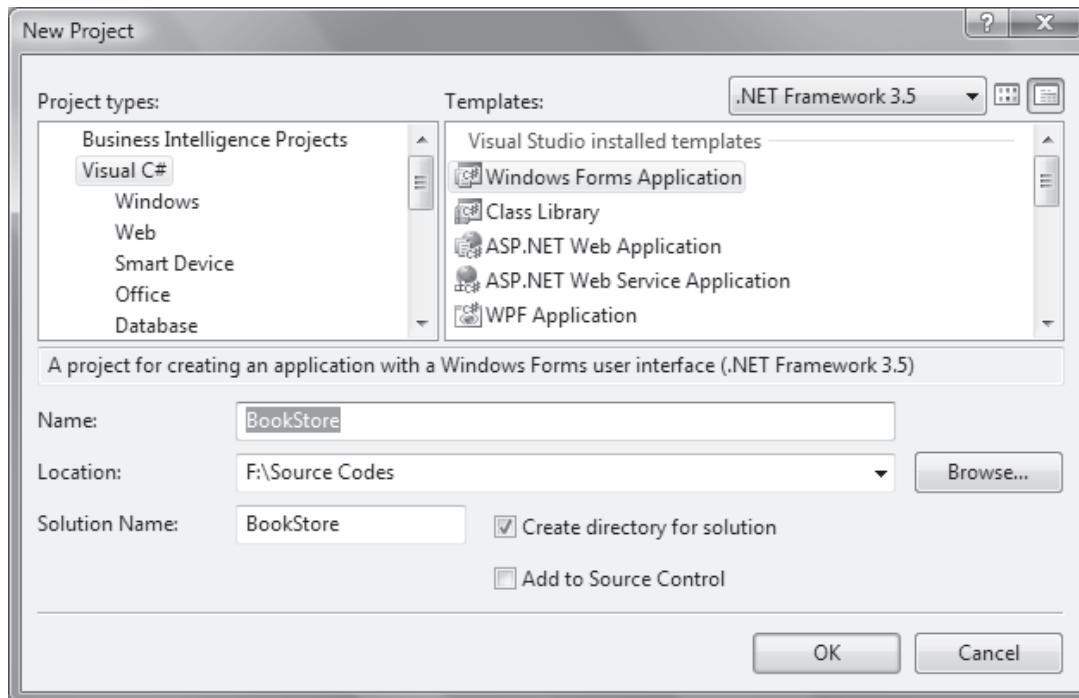


Figure 17.1: Creating the BookStore project

3. Rename the default form to MainForm.cs and within the class, rename the classname, Form1, to frmMainForm.
4. Set the properties of frmMainForm as shown in table 17.2.

Property	Value
Name	frmMainForm
Text	Main Form
FormBorderStyle	None
Size	457, 241

Table 17.2: Form Properties

5. Add a Label and two Buttons to the form. Set their properties as shown in the following table 17.3.

Control	Property	Value
Label	Name	lblHeading
	Text	GoodReads BookStore Admin System
	Font	Times New Roman 14.25F
Button	Name	btnExit
	Text	Exit
Button	Name	btnEnter
	Text	Enter

Table 17.3: Control Properties

6. Center the controls such that the User Interface should resemble as shown in figure 17.2.



Figure 17.2: User Interface of Main Form

7. Switch to the code view. Declare the following private variables in the class frmMainForm:

```
private int _top, _left, _height, _width;
```

These variables will be used to store the top , left, height, and width values of the form.

8. **Switch to design view and double-click the form. This results in the Load event handler being auto-generated.**
9. **Revert to the code view. Add the following using statement which is required for drawing functionality.**

```
using System.Drawing.Drawing2D;
```

10. **Add the following code in the Load event handler.**

```
private void frmMainForm_Load(object sender, EventArgs e)
{
    // Create and add an ellipse to the form graphics path
    GraphicsPath path = new GraphicsPath();
    path.AddEllipse(0, 0, this.Width, this.Height);
    this.Region = new Region(path);
    this.BackColor = Color.LightGoldenrodYellow;

    // Retrieve the dimensions of the ellipse in order
    //to draw a border around it
    _top = 0;
    _left = 0;
    _height = this.Height;
    _width = this.Width;
}
```

In the code, an instance of `GraphicsPath` is created. This class is used to represent a graphical path on the screen. Applications normally use a path to draw and fill shapes. The `AddEllipse()` method is called to draw an ellipse on the path. This path is then used to create a region which is assigned to the form using the `Region` property of the form. The background color of the form is set to `LightGoldenrodYellow`. The ellipse that is drawn will not have any border, hence, in order to draw a border, the coordinates of the ellipse are stored in four variables.

11. **Generate an event handler for Paint event of the form through the Properties window. Add code to event handler to create a border.**

```
private void frmMainForm_Paint(object sender, PaintEventArgs e)
{
    // Draw a border around the ellipse
    GraphicsPath innerPath = new GraphicsPath();
    innerPath.AddEllipse(_left, _top, _width, _height);
    e.Graphics.DrawPath(new Pen(Color.Black, 5), innerPath);
}
```

There is no border property or method to draw a border. Hence, to simulate a border like effect, another ellipse with black outline is drawn over the existing ellipse in the `Paint()` method.

- 12. Add code to the Click event of the button, btnExit. This code confirms from the user whether to exit or not. Depending on the button clicked by the user in the message box, appropriate action takes place.**

```
private void btnExit_Click(object sender, EventArgs e)
{
    DialogResult dr = MessageBox.Show("This will exit the application.", "Exit",
        MessageBoxButtons.OKCancel);
    if (dr.ToString().Equals("OK"))
    {
        Application.Exit();
    }
}
```

- 13. Save, build, and test to view the nonrectangular form.**

The output is shown in figure 17.3.

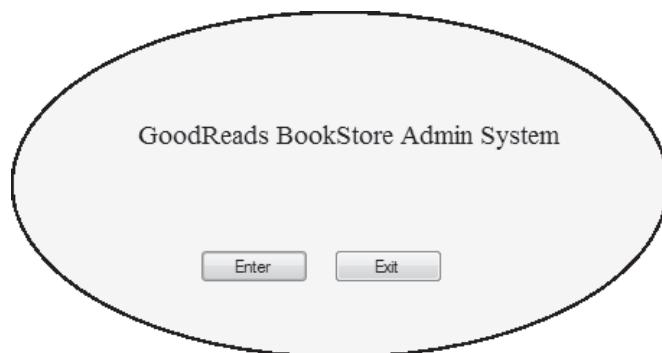


Figure 17.3: Output Showing Nonrectangular form

### Creating Tabbed Views in the Application

Tabbed Views can be created by adding a `TabControl` and two or more `TabPage`s within it.

The steps to create tabbed views for the application are as follows:

1. Add a new Windows Form to the project by selecting Project ➔ Add Windows Form.
2. Rename the form as `Admin.cs` and the class as `frmAdmin`.
3. Set its `Size` property to 676, 507.
4. Set the `Text` property of the form to Admin Screen.
5. Drag the `TabControl` from the Toolbox and drop it on the form.
6. Set its `Name` property as `tbcAdmin` and `Size` as 640, 378.
7. Add a `TabPage` in addition to the default two `TabPage`s. Use the `TabPage`s property on the `TabControl` to add the new page.
8. Set the `Name` and `Text` properties of the `TabPage`s as shown in table 17.4.

Control	Property	Value
TabPage	Name	<code>tbgAdd</code>
	Text	Add Books
TabPage	Name	<code>tbgView</code>
	Text	View Books
TabPage	Name	<code>tbgSearch</code>
	Text	Search

Table 17.4: `TabPage` properties

9. In the first `TabPage`, `tbgAdd`, add six `Label` and `TextBox` controls. Also, add a `PictureBox` and a `Button` control to the `TabPage`. Set the properties of all these controls as shown in the following table 17.5.

Control	Property	Value
Label	Name	<code>lblBookID</code>
	Text	BookID:
	Font	Bold
Label	Name	<code>lblTitle</code>
	Text	Title:
	Font	Bold
Label	Name	<code>lblAuthor</code>
	Text	Author:
	Font	Bold

Control	Property	Value
Label	Name	lblPublisher
	Text	Publisher:
	Font	Bold
Label	Name	lblPrice
	Text	Price:
	Font	Bold
Label	Name	lblDate
	Text	Purchase Date:
	Font	Bold
TextBox	Name	txtBookID
	ReadOnly	True
TextBox	Name	txtTitle
TextBox	Name	txtAuthor
TextBox	Name	txtPublisher
TextBox	Name	txtPrice
TextBox	Name	txtDate
Button	Name	btnSave
	Text	Save
PictureBox	Name	picBooks

Table 17.5: Properties of controls on first TabPage

10. Design the layout of these controls as shown in figure 17.4. To set the picture, import an image of books into the solution folder and then, set the Image property of the PictureBox.

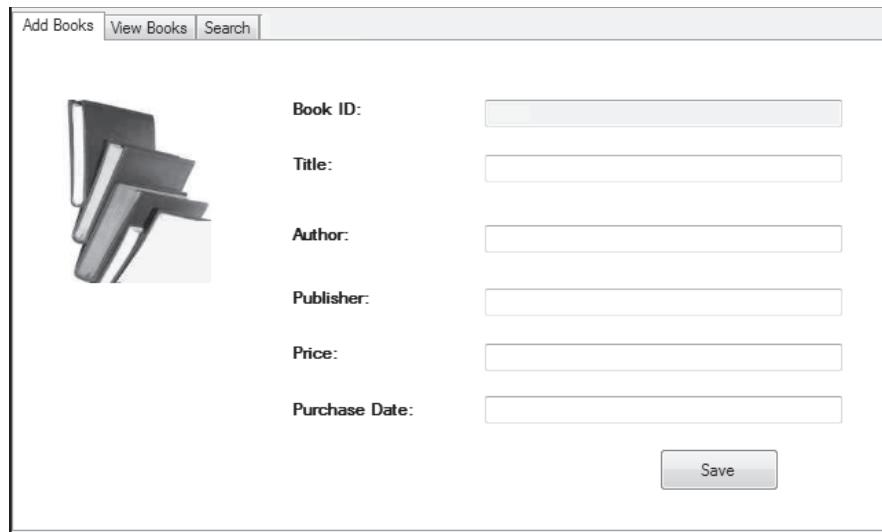


Figure 17.4: TabPage with Controls

### Adding a data source

1. Add a data source to the project by selecting Data→Add New Data Source.

This displays the **Data Source Configuration Wizard** as shown in figure 17.5.

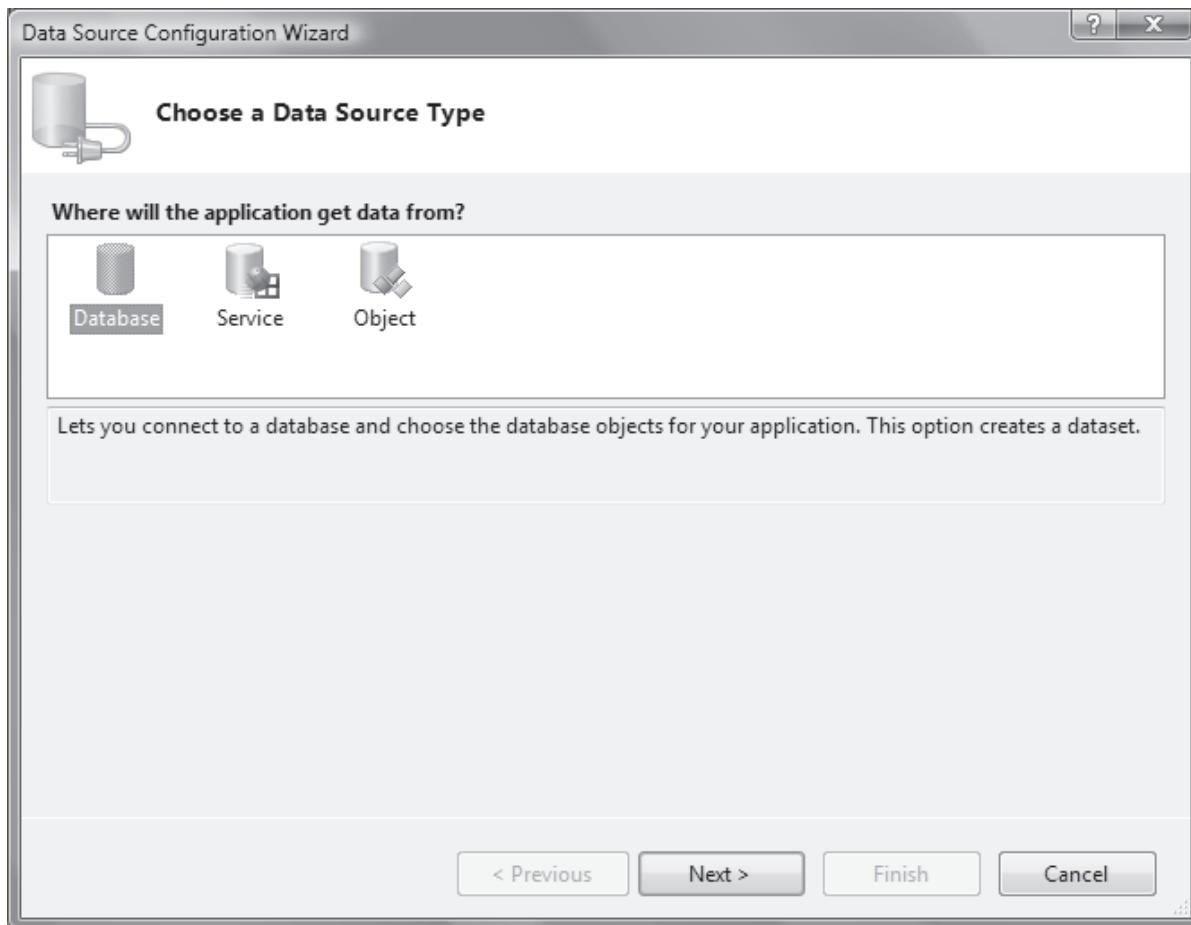


Figure 17.5: Data Source Configuration Wizard

2. Select Database and click Next.

This displays the **Choose Your Data Connection** page of the wizard as shown in figure 17.6.

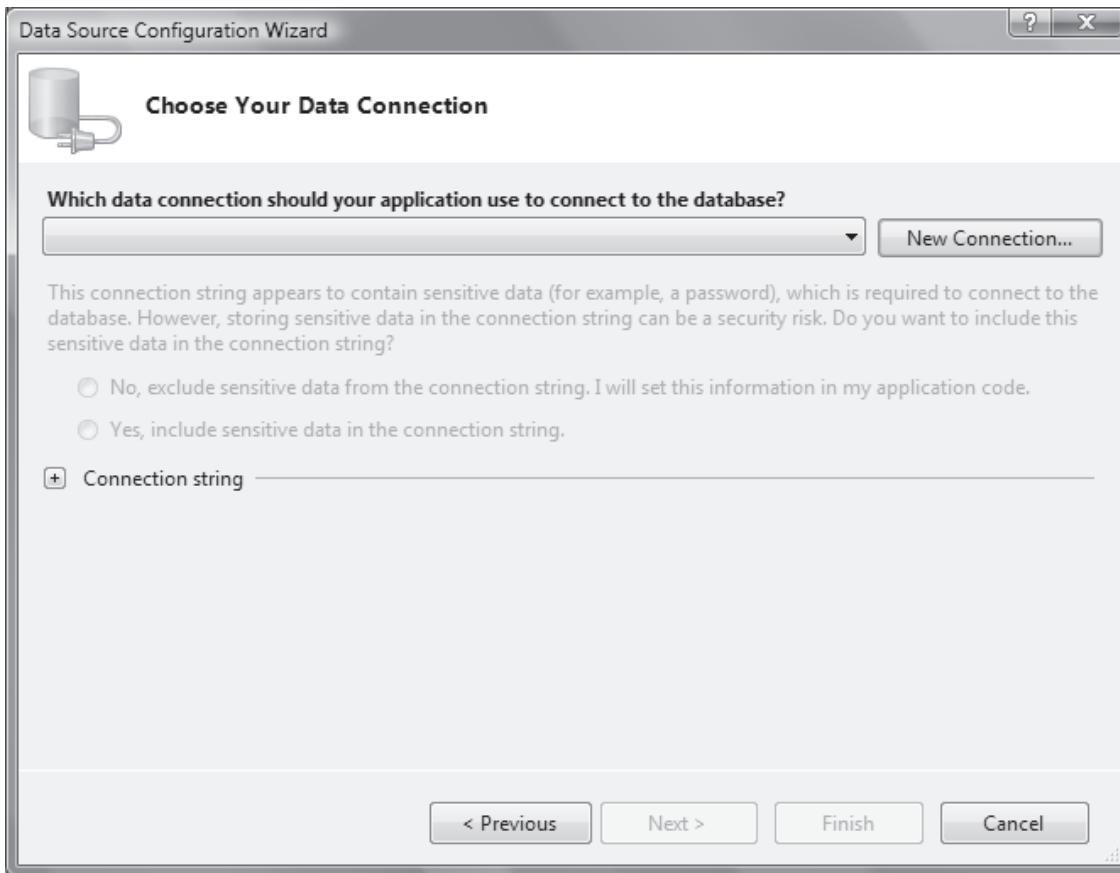


Figure 17.6: Choose Your Data Connection page

**3. Click New Connection.**

This displays the **Add Connection** page of the wizard.

4. Specify appropriate connection details to the SQL Server and select the database GoodReads as shown in figure 17.7.

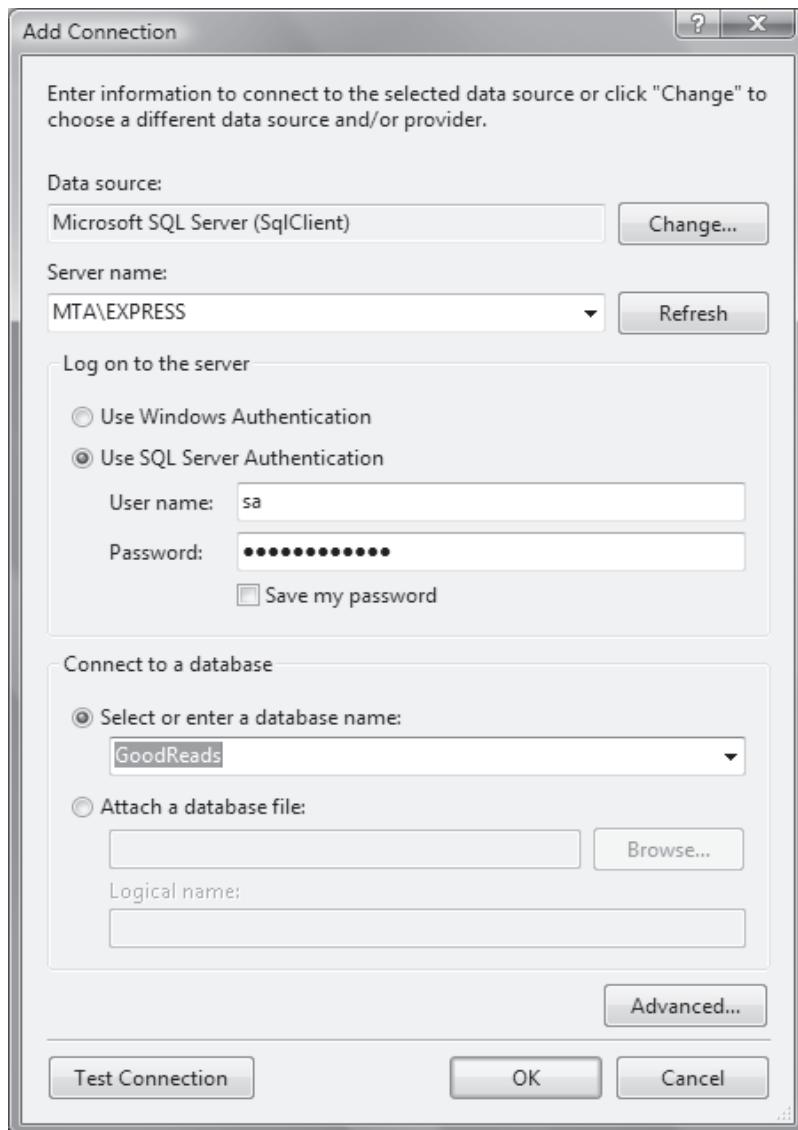


Figure 17.7: Add Connection page

5. Click OK. This will bring you back to the Choose Your Data Connection page of the wizard.

6. Select the Yes, include sensitive data in the connection string option button as shown in figure 17.8.

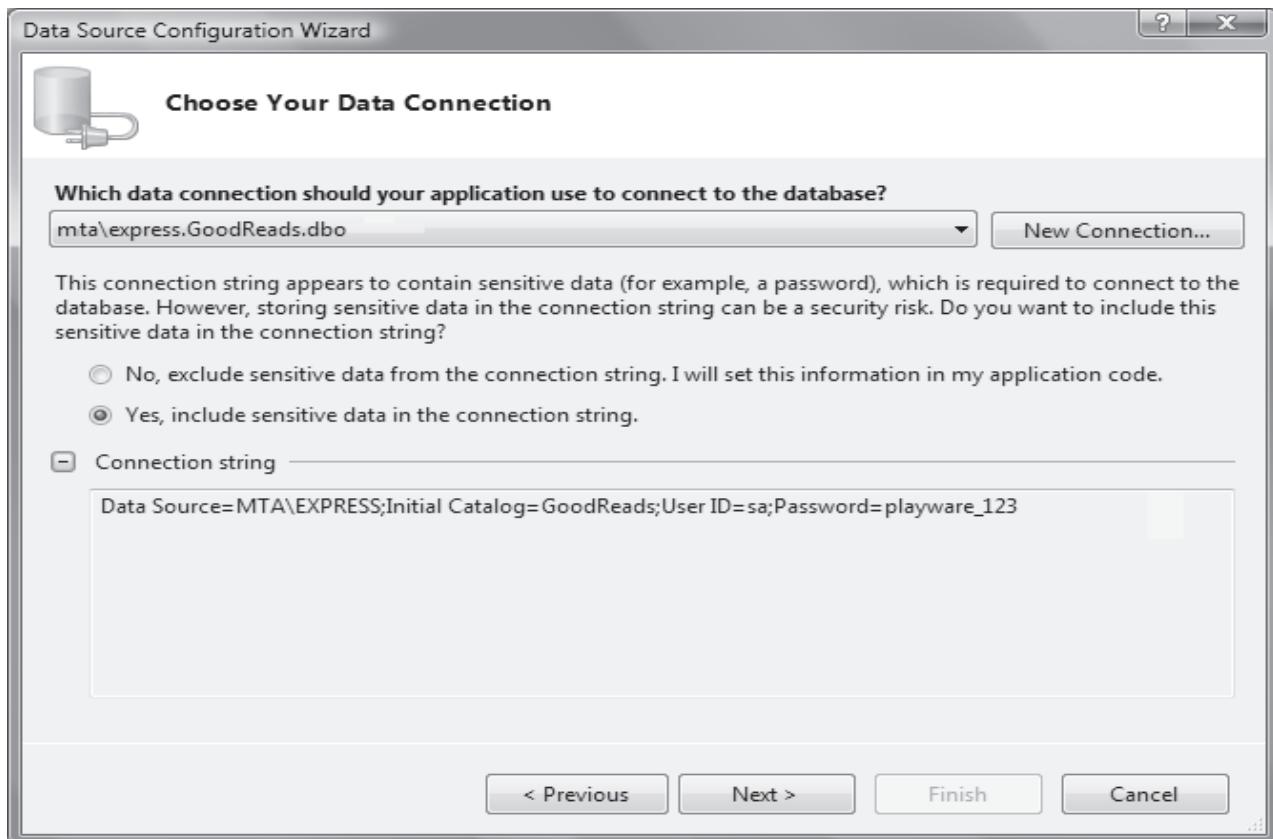


Figure 17.8: Choose Your Data Connection page

7. Click Next.

This displays a page to confirm if you want to save the connection string to an application file as shown in figure 17.9.

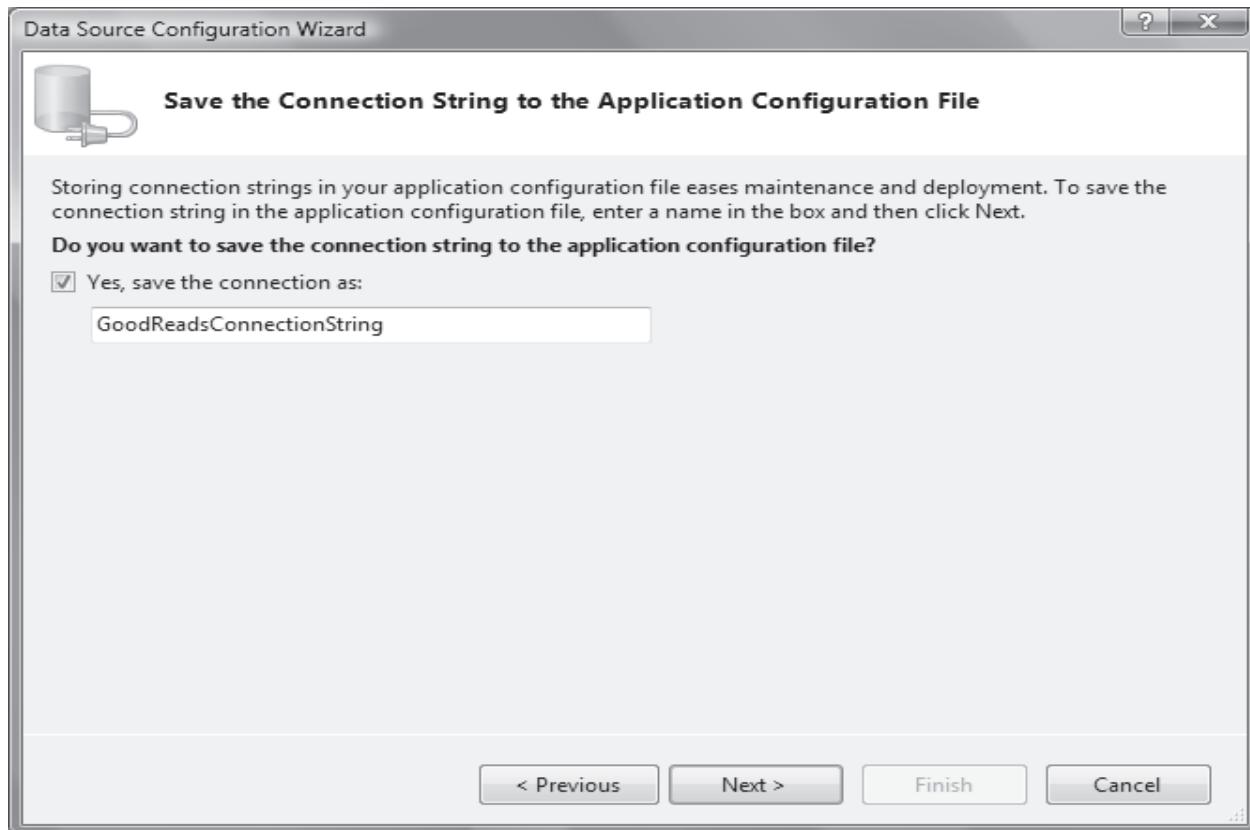


Figure 17.9: Save Connection string page

8. Click Next. This displays the Choose Your Database Objects page of the Wizard.

9. Select the table Books in Choose Your Database Objects page of the Wizard as shown in figure 17.10.

The typed DataSet that will be generated will have the name GoodReadsDataSet.

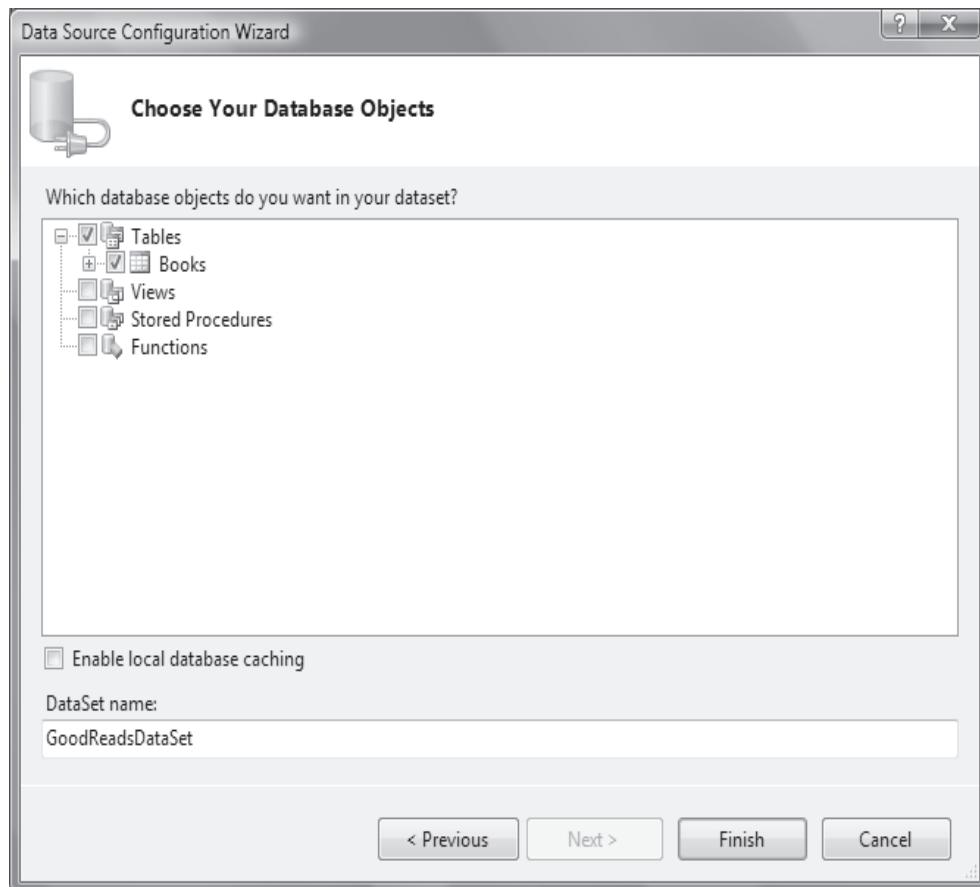


Figure 17.10: Choose Your Database Objects

10. Click Finish. This will result in the typed DataSet, GoodReadsDataSet.

#### Adding data handling code

1. Add a new class to the project by selecting Project→Add Class.
2. Rename the class to DataAccess.cs.
3. Add the following using statements to the class:

```
using System.Data.SqlClient;
using System.Data;
using System.Collections;
```

These are needed to handle data access operations and also to create collections such as ArrayList.

**4. Declare the following variables in the class.**

```
private int _maxid;
public static string error;
```

**5. Declare and initialize the DataSet and Adapter at the class level as follows:**

```
// Declare and initialize the dataset and adapter
GoodReadsDataSet ds = new GoodReadsDataSet();
GoodReadsDataSetTableAdapters.BooksTableAdapter bta = new
    GoodReadsDataSetTableAdapters.BooksTableAdapter();
```

**6. Add the following code in the constructor of the class.**

```
public DataAccess()
{
    // Load the Connection String
    bta.Connection.ConnectionString =
        "Server=MTA\\EXPRESS;Database=GoodReads;UID=sa;
        Pwd=playware_123";
    bta.Fill(ds.Books);
}
```

Here, the connection string is set programmatically within the constructor of `DataAccess`. The `ConnectionString` property of the `TableAdapter`'s connection is used to set the connection string. The SQL Server name, initial database, user id, and password are included in the connection string.

**7. Create a method to retrieve all the records from the Books table.**

```
public DataTable GetAllData()
{
    try
    {
        // Populate data into the DataSet
        bta.Fill(ds.Books);
        return ds.Books;
    }
}
```

```

        catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        return null;
    }
}

```

The `GetAllData()` method populates the table in the `DataSet` with records using the adapter's `Fill()` method. The method returns the records as a `DataTable`. The code in the method is enclosed in a `try-catch` block in order to handle any exceptions.

#### **8. Create a method to generate the BookID.**

```

public int GenerateBookID()
{
    try
    {
        if (bta.Connection.State == ConnectionState.Closed)
        {
            bta.Connection.Open();
        }

        // Get the max book id and calculate new book id
        string cmd = "SELECT Max(BookID) FROM Books";
        bta.Adapter.SelectCommand = new SqlCommand(cmd, bta.Connection);
        _maxid = (int)bta.Adapter.SelectCommand.ExecuteScalar();

        // Increment to get the new id
        _maxid++;
        return _maxid;
    }

    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
    }
}

```

The code calculates a new value for BookID by retrieving the most recent (maximum) BookID and adding 1 to it.

Before you execute any queries or commands, the database connection must be open. Hence, you must first check see if the state is closed. If yes, then, the connection is opened. If the connection is already open, you simply proceed. Then, a SELECT query is constructed as a string and passed to the SelectCommand of the TableAdapter. The connection is also passed as a parameter.

The ExecuteScalar() command executes the query and returns a single scalar value, which is the maximum book id. This id value is incremented by 1 to generate the latest book id and this value is returned back. All the code is enclosed in a try-catch block in order to handle any exceptions.

The catch block includes a statement return -1 because the method is declared to return an int value.

**9. Create a method SaveData() that will save records into the database.**

```
public int SaveData(string[] data)
{
    string cmd;
    int result;
    try
    {
        if (bta.Connection.State == ConnectionState.Closed)
        {
            bta.Connection.Open();
        }
        _maxid = GenerateBookID();

        // Construct command to insert new record
        cmd = "INSERT INTO Books VALUES(" + _maxid + "," + data[0] + ",," +
        + data[1] + ",," + data[2] + ",," + data[3] + ",," + data[4] + ")";
        bta.Adapter.InsertCommand = new SqlCommand(cmd, bta.Connection);
        result = bta.Adapter.InsertCommand.ExecuteNonQuery();
        return result;
    }
}
```

```

    catch (Exception ex)
    {
        Console.WriteLine(ex.Message);
        error = ex.Message;
        return -1;
    }
}

```

In the code, the state of the connection is checked. If the state is closed, then, the connection is opened. The method constructs an `INSERT` command. Values from the string array named `data` are included in the `INSERT` command. The command is then executed through the `ExecuteNonQuery()` method of the adapter.

The number of records successfully inserted is stored into a variable `result`. This value is then returned from the `SaveData()` method. All the code is enclosed in a `try-catch` block in order to handle any exceptions. The `catch` block includes a statement `return -1` because the method is declared to return an `int` value.

#### 10. Save the class.

#### Adding new records

- Open `frmAdmin` in design view and double-click the form. This results in the `Load` event handler being auto-generated.**
- Revert to the code view. Add the following code in the `Load` event handler.**

```

private void frmAdmin_Load(object sender, EventArgs e)
{
    PopulateID();
}

```

Since the `BookID` field will be read-only on the form, the id must be generated through code and set on the form. For this purpose, a method named `PopulateID()` is invoked.

**3. Create a method named PopulateID() as follows:**

```
private void PopulateID()
{
    // Set the auto-generated Book ID
    DataAccess da = new DataAccess();
    int id = da.GenerateBookID();
    if (id == 0)
    {
        MessageBox.Show(DataAccess.error, "Error");
    }
    else
    {
        txtBookID.Text = id.ToString();
    }
}
```

The `DataAccess` class contains the logic to generate the id using a method named `GenerateBookID()`. This method is invoked through an instance of `DataAccess` named `da`.

If there is no error, the id is assigned to the text box for `BookID`.

- 4. Revert to Design view and generate an event handler for `txtDate _ Validating`.**
- 5. Add the following code in the `txtDate _ Validating` event handler.**

```
private void txtDate_Validating(object sender, CancelEventArgs e)
{
    if (DataValid())
    {
        btnSave.Enabled = true;
    }
}
```

As the Purchase Date field is the last field in the form, validation for the form data is performed in this event handler. The code first checks to see if the input data is valid by calling a method `DataValid()`. If the data is valid, the `Save` button is enabled.

**6. Write a method DataValid() to validate the input data.**

```
private bool DataValid()
{
    // Check for validations
    bool errorflag = false;
    foreach (Control c in tbpgAdd.Controls)
    {
        Control ctrl = (Control)c;
        if (ctrl is TextBox)
        {
            if ((ctrl.Name != "txtBookID") && (String.IsNullOrEmpty(ctrl.Text)))
            {
                MessageBox.Show(ctrl.Name.Substring(3) + " Cannot be
empty", "Error");
                errorflag = true;
            }
        }
    }
    // Finally, check if any of them caused error
    if (errorflag)
    {
        return false;
    }
    else
    {
        return true;
    }
}
```

The method `DataValid()` defines validations to check if any of the text boxes (except `txtBookID`) are empty. If they are empty, appropriate error messages are displayed.

You can also add code to implement other validations if required.

7. Revert to Design view and double-click the Save button. This will generate an event handler `btnSave_Click`.
8. Write the following code in the `btnSave_Click` event handler.

```
private void btnSave_Click(object sender, EventArgs e)
{
    DataAccess da = new DataAccess();

    string[] data = { this.txtTitle.Text, this.txtAuthor.Text,
                      txtPublisher.Text, txtPrice.Text, txtDate.Text };

    // Call the SaveData method
    int result = da.SaveData(data);

    // Check if insertion was successful
    if (result == 1)
    {
        MessageBox.Show("Record Saved", "Success");
    }
    else
    {
        MessageBox.Show("Error in Record Data. Could not be Saved",
                       "Error");
    }

    // Prepare the controls for the next new record
    foreach (Control c in tbpgAdd.Controls)
    {
        Control ctrl = (Control)c;
        if (ctrl is TextBox)
        {
            ctrl.Text = "";
        }
    }
}
```

```
    }  
}  
PopulateID();  
}
```

A string array is constructed using the input values from the form and passed to the `SaveData()` method of `DataAccess` class. This method is called through an instance of `DataAccess` named `da`.

Appropriate success or error messages are displayed depending on successful insert or error. Then, the existing values in the text boxes are cleared so that a new record can be inserted when this tab is clicked again. The `PopulateID()` method is called to set the next value of `BookID`.

9. Declare a variable of type DialogResult at the class level.

```
private DialogResult dr;
```

## **10. Revert to Design view of frmAdmin.**

## 11. Generate an event handler for the FormClosing event.

**12.** Write the following code in the event handler of FormClosing.

```
private void frmAdmin_FormClosing(object sender, FormClosingEventArgs e)
{
    // Check if user had already tried to exit before
    if (_dr.ToString() != "OK")
    {
        _dr = MessageBox.Show("This will exit the application.",
                             "Exit", MessageBoxButtons.OKCancel);
    }

    // Check if user clicked OK to confirm the exit
    if (_dr.ToString() == "OK")
    {
        Application.Exit();
    }
    else
    {

```

```

        e.Cancel = true;
    }
}

```

- 13.** Open MainForm.cs and write the following code to handle the Click event of btnEnter. This will link the entry form, frmMainForm, to frmAdmin.

```

private void btnEnter_Click(object sender, EventArgs e)
{
    // Redirect to the next screen
    frmAdmin admin = new frmAdmin();
    admin.Show();
    this.Hide();
}

```

- 14.** Save, build, and test the functionality of adding new records. A sample new record is shown in figure 17.11.

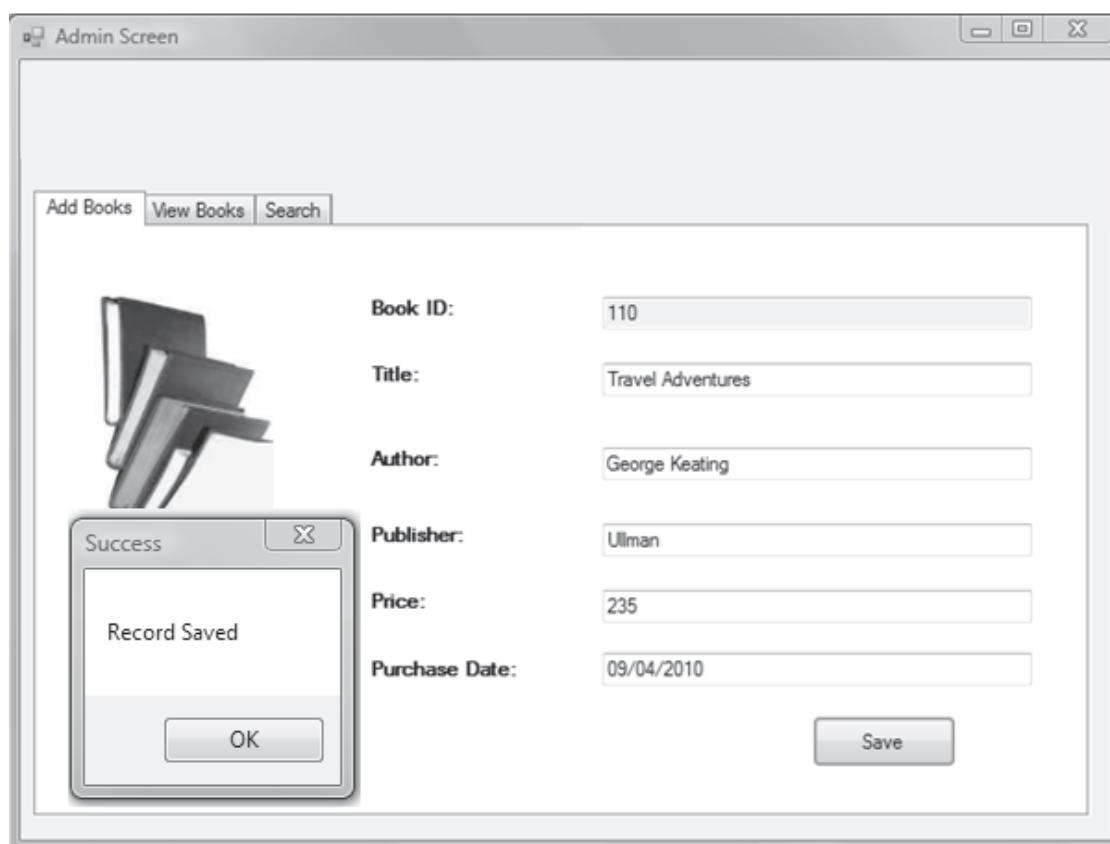


Figure 17.11: Adding new record

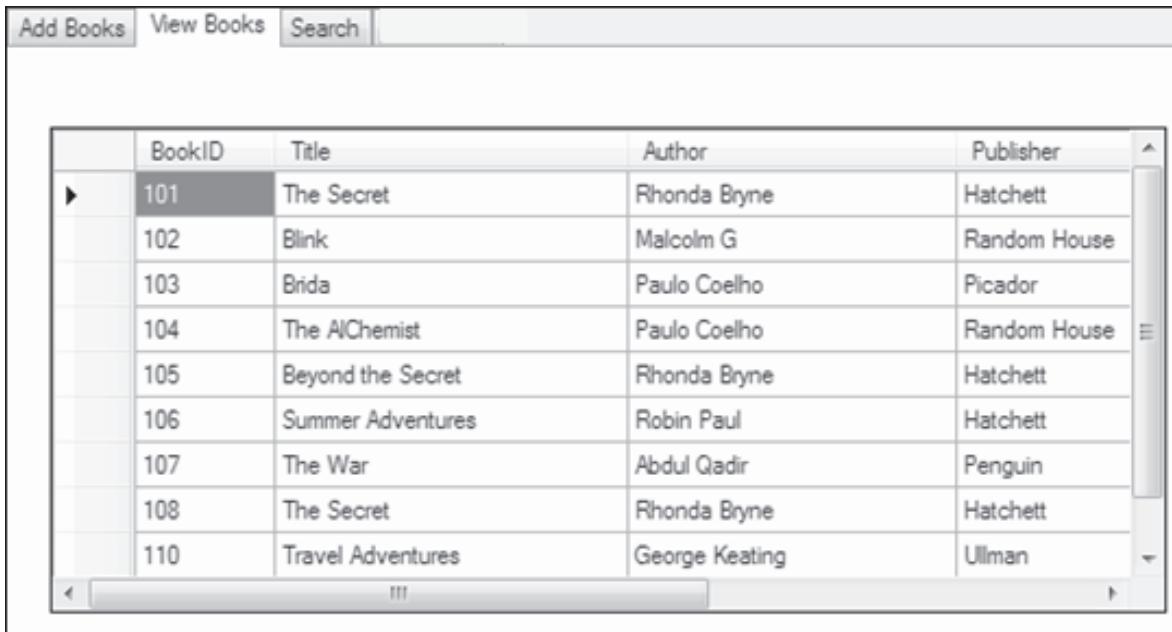
### Viewing Records

The steps to add functionality for tbpgView are as follows:

1. Add a DataGridView to the Tab Page, tbpgView.
2. Rename the DataGridView to dgvwRecords.
3. Generate an event handler for SelectedIndexChanged event of the TabControl.
4. Add the following code to the SelectedIndexChanged event handler. The GetAllData() method was written earlier in the DataAccess class.

```
private void tbcAdmin_SelectedIndexChanged(object sender, EventArgs e)
{
    if (tbcAdmin.SelectedTab.Name == "tbpgView")
    {
        DataAccess da = new DataAccess();
        dgvwRecords.DataSource = da.GetAllData();
        if (dgvwRecords.DataSource == null)
        {
            MessageBox.Show(DataAccess.error, "Error");
        }
    }
}
```

5. Save, build, and test the functionality of viewing records. An example of the output is shown in figure 17.12.



	BookID	Title	Author	Publisher
▶	101	The Secret	Rhonda Byrne	Hachett
	102	Blink	Malcolm G	Random House
	103	Brida	Paulo Coelho	Picador
	104	The AlChemist	Paulo Coelho	Random House
	105	Beyond the Secret	Rhonda Byrne	Hachett
	106	Summer Adventures	Robin Paul	Hachett
	107	The War	Abdul Qadir	Penguin
	108	The Secret	Rhonda Byrne	Hachett
	110	Travel Adventures	George Keating	Ullman

Figure 17.12: Viewing records

### Searching for particular data

1. Create a method in the **DataAccess** class to retrieve data belonging to a particular column as specified by the user. This method will be used later in the form to populate the combo box.

```
public ArrayList GetData(string criteria)
{
    ArrayList data = new ArrayList();
    try
    {
        bta.Fill(ds.Books);
        // Retrieve the data only from given column
        foreach (DataRow dr in ds.Books.Rows)
        {
            if (criteria.Equals("Author"))
            {
                data.Add(dr["Author"].ToString());
            }
        }
    }
```

```
else if (criteria.Equals("Title"))
{
    data.Add(dr["Title"].ToString());
}
else
{
    if (criteria.Equals("Publisher"))
    {
        data.Add(dr["Publisher"].ToString());
    }
}
return data;
}
catch (Exception ex)
{
    Console.WriteLine(ex.Message);
    error = ex.Message;
    return null;
}
```

- 2. Create a method in the DataAccess class to retrieve the records from the table based on a given criteria.**

```
public SqlDataReader SearchData(string criteria, string column)
{
try
{
if (bta.Connection.State == ConnectionState.Closed)
{
    bta.Connection.Open();
}
```

```

// Search for specific records based on given criteria

string cmd = "SELECT * FROM Books WHERE " + column + "=" + criteria
+ "'";

bta.Adapter.SelectCommand = new SqlCommand(cmd, bta.Connection);

SqlDataReader dreader = bta.Adapter.SelectCommand.ExecuteReader();

return dreader;

}

catch (Exception ex)

{

    Console.WriteLine(ex.Message);

    error = ex.Message;

}

}

```

3. Revert to Design view of Admin.cs.
4. Add a GroupBox, three RadioButton, a Label, a ComboBox, and a DataGridView to the TabPage, tbpgSearch. Arrange them as shown in figure 17.13.

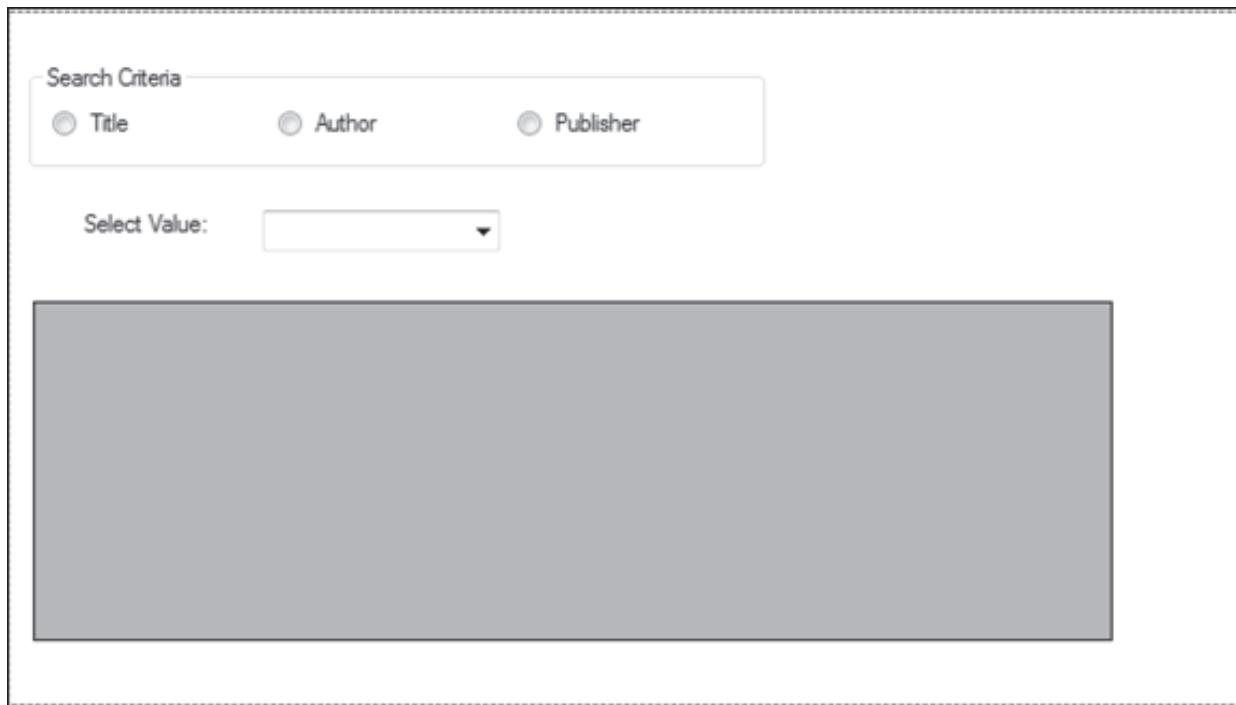


Figure 17.13: Search TabPage

5. Set the properties of the controls as shown in the following table 17.6

Control	Property	Value
GroupBox	Name	grpCriteria
	Text	Search Criteria
RadioButton	Name	radPublisher
	Text	Publisher
RadioButton	Name	radAuthor
	Text	Author
RadioButton	Name	radTitle
	Text	Title
Label	Name	lblSelect
	Text	Select Value
ComboBox	Name	cboCriteria
DataGridView	AutoSizeColumnsMode	AllCells
	Name	dgvwSearchData

Table 17.6: Properties of controls on Search TabPage

6. Switch to the code view of frmAdmin. Create a method named **Populate()** to add items to the combo box.

```
private void Populate()
{
    cboCriteria.Items.Clear();
    DataAccess da = new DataAccess();
    if (_selected != null)
    {
        System.Collections.ArrayList data = da.GetData(_selected);
        if (data != null)
        {
            foreach (string item in data)
            {
                if (!cboCriteria.Items.Contains(item.Trim()))
                {
```

```

        cboCriteria.Items.Add(item.Trim());
    }

}

cboCriteria.Sorted = true;
cboCriteria.SelectedIndex = 1;
}

else { MessageBox.Show(DataAccess.error); }

}

```

- 7. Declare the following private variable at the class level to hold the selected value from the radio buttons.**

```
private string _selected;
```

- 8. Add code for the CheckedChanged event of the three radio buttons and associate it with respective event through Properties window.**

```

private void radTitle_CheckedChanged(object sender, EventArgs e)
{
    _selected = "Title";
    Populate();
}

private void radPublisher_CheckedChanged(object sender, EventArgs e)
{
    _selected = "Publisher";
    Populate();
}

private void radAuthor_CheckedChanged(object sender, EventArgs e)
{
    _selected = "Author";
    Populate();
}

```

**9. Add code for the SelectedIndexChanged event of the combo box.**

```
private void cboCriteria_SelectedIndexChanged(object sender, EventArgs e)
{
    string item = cboCriteria.SelectedItem.ToString();
    DataAccess da = new DataAccess();
    BindingSource bsrcData = new BindingSource();
    bsrcData.DataSource = da.SearchData(item, _selected);
    dgvwSearchData.DataSource = bsrcData;
}
```

**10. Save, build, and test the functionality of viewing records. An example of the output is shown in figure 17.14.**

The screenshot shows a Windows application window titled "Search". At the top left are three buttons: "Add Books", "View Books", and "Search". To the right of the buttons is a search bar. Below the search bar is a section labeled "Search Criteria" containing three radio buttons: "Title", "Author", and "Publisher". Underneath this is a dropdown menu labeled "Select Value" with the option "Random House" selected. At the bottom is a "DataGridView" displaying two rows of book data:

	BookID	Title	Author	Publisher
▶	102	Blink	Malcolm G	Random House
	104	The AlChemist	Paulo Coelho	Random House

Figure 17.14: Searching Records

### Using ToolStripContainer and ToolStripButtons

The steps to add and configure ToolStripContainer and ToolStripButtons are given as follows:

1. Add a ToolStrip from the Toolbox onto the form `frmAdmin`, over the TabControl.
2. Select its smart tag option and click Embed in ToolStripContainer as shown in figure 17.15.

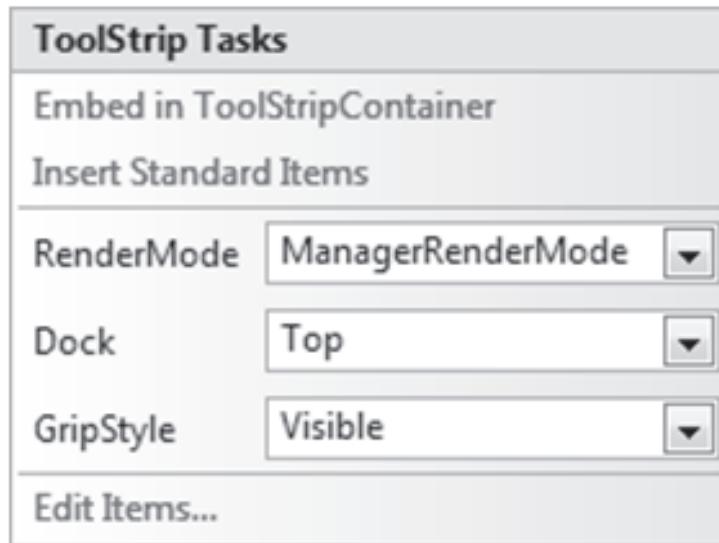


Figure 17.15: ToolStrip Tasks

This will automatically add a ToolStripContainer and place the ToolStrip inside it. Clicking the smart tag on the ToolStripContainer will display its tasks as shown in figure 17.16.

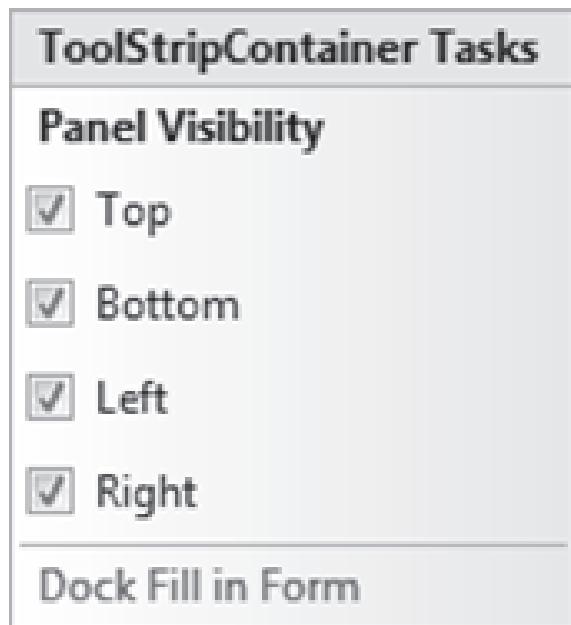


Figure 17.16: ToolStripContainer Tasks

3. Deselect all panels except the top panel in the ToolStripContainer.
4. Set the Name property of the ToolStripContainer to toolstrpctrBks. Similarly, set the ToolStrip name to toolstrpBks.
5. Add three ToolStripButton and two ToolStripSeparator controls on the ToolStrip.
6. Set their Name properties as shown in table 17.7:

Control	Name
ToolStripButton	viewToolStripButton
ToolStripButton	searchToolStripButton
ToolStripButton	newToolStripButton
ToolStripSeparator	toolstrpsepOne
ToolStripSeparator	toolstripsepTwo

Table 17.7: Name Properties of various ToolStripItems

7. Set the Image property of each ToolStripButton to an appropriate image. For example, you can create a ToolStrip with buttons as shown in figure 17.17:



Figure 17.17: ToolStripButtons

8. Add the following code to the respective Click event of each of the buttons.

```
private void newToolStripButton_Click(object sender, EventArgs e)
{
    // Display the first tab page
    tbcAdmin.SelectedIndex = 0;
}

private void viewToolStripButton_Click(object sender, EventArgs e)
{
    // Display the second tab page
    tbcAdmin.SelectedIndex = 1;
}
```

```

private void searchToolStripButton_Click(object sender, EventArgs e)
{
    // Display the third tab page
    tbcAdmin.SelectedIndex = 2;
}

```

When each ToolStrip button is clicked, the corresponding tab page will open. To achieve this, you set the SelectedIndex property to the corresponding tab page index.

## Part II - 60 minutes

1. In the application shown in Part I, add a new TabPage named 'Add Edition'. Add a DataGridView to this tab page. Create a new table in the SQL Server database and name it as Editions. It should have two columns as shown:

Column Name	DataType	Key
BookID	int	foreign key
Edition	nchar(20)	

On clicking the tab Add Edition at runtime, the DataGridView should display the merged contents of Books table and Editions table.

### Hints:

#### In the **DataAccess** class:

- ➔ Create a method `GetEditions()` that returns a `DataSet`.
- ➔ Create a new `DataSet` and populate it with data from the `Editions` table.
- ➔ At the time of filling the `DataSet`, make sure to name the table name within the new `DataSet` as `Books` (even though it is `Editions`) because, if not, the merging will not succeed.
- ➔ Invoke the `Merge()` method of `DataSet` on the new `DataSet` to merge it with existing `DataSet`. Use the parameter `MissingSchemaAction.Add`.

#### In `frmAdmin` class:

- ➔ In the appropriate event handler, call the `GetEditions()` method and assign the result to the `DataGridView`.

2. Create an application that will display an image in the center of the form. The user can change the size of the image using TrackBar control. When the user reaches the lower or upper limit of the TrackBar, a NotifyIcon should display the appropriate message to the user.

**Hint:**

Use the TrackBar control's ValueChanged event and LargeChange property.



## Try it Yourself

1. Create an application that will find a phrase or text in a given text file. The application should begin with an entry screen that is the shape of a polygon. The next screen of the application displays a FindText dialog box which will accept a phrase or text to be found and the name of the file in which to search. Implement this using a custom dialog box. After accepting the input through the dialog box, the program must search for the given text in the file.

WRITE-UPS BY

**EXPERTS AND LEARNERS**

TO PROMOTE NEW AVENUES AND  
ENHANCE THE LEARNING EXPERIENCE



FOR FURTHER READING, LOGIN TO

**[www.onlinevarsity.com](http://www.onlinevarsity.com)**

# Module - 18

## Windows Presentation Foundation

Welcome to the Module, **Windows Presentation Foundation**.

This session will explore the WPF architecture, controls in WPF, and graphics and animation features in WPF.

In this Module, you will learn about:

- Define Windows Presentation Foundation (WPF)
- Explain WPF architecture
- Describe XAML, its basic structure, and its role in WPF applications
- Describe the controls in WPF
- Describe the process of using styles and transforms in WPF applications
- Describe the graphics and animation features in WPF
- Explain WPF user controls

## 18.1 Introduction

The important part of every application is its user interface. The end-user expectation from user interfaces has advanced significantly.

Traditional applications use menu-driven graphical user interfaces (GUIs), but there is a need to use two-dimensional (2D) and three-dimensional (3D) graphics, display video, run animations, and work with various kinds of documents.

Windows has been providing all these ingredients of the user interface. For example, with Windows Forms, you can build a Windows-based GUI that can run in a .NET environment. You can create a Web browser interface with HTML, and perhaps Java applets or JavaScript code. The videos can be displayed using Windows Media Player or some other tool, while the document formats can be defined by Microsoft Word, PDF, or any other way. The challenge to create a consistent user interface for different kinds of clients using various technologies is not simple.

WPF, released with .NET Framework 3.0, is to address this challenge.

This session explores WPF technology, its structure, various controls in WPF, and hosting WPF controls in Windows Forms applications.

## 18.2 Introduction to Windows Presentation Foundation

WPF is basically a presentation subsystem in .NET Framework for creating, displaying, and manipulating user interfaces, documents, images, movies, and media. WPF supports two-dimensional and three-dimensional graphics, animation, video, and different kinds of documents. It gives a common foundation for desktop and browser clients; WPF makes it easier to develop applications that address both.

WPF has a rich set of Application Programming Interfaces (APIs) which are tied to a new markup language named Extensible Application Markup Language (XAML). This allows you to create a user interface based on markup and programming codes.

## 18.3 Features of WPF

WPF is a common technology for Windows-based and Web-based applications. WPF has a number of useful capabilities in comparison with older user interface engines. It provides a simple process to build visual effects and dynamic interfaces.

The features of WPF are as follows:

### → Resolution Independence

The size of the WPF window does not depend on Dots Per Inch (DPI) setting of the monitor. The WPF graphics engine provides scaling of any component without displaying it as individual pixels.

### → Animation

WPF provides animation features for an element such as rotating, moving, and changing its size.

→ **2D and 3D Graphics**

WPF provides graphics features such as shapes, image, and brushes. It supports creation of 2D and 3D vector graphics.

→ **New Programming Language – XAML**

XAML is a new markup language used to build WPF applications. It supports faster and easier GUI development through declarative markup.

→ **Data Binding**

Data binding connects a user interface with business logic and binds data with data sources.

→ **Video and Audio Elements**

The graphics capabilities of WPF provide support for multimedia, which includes audio and video.

Microsoft .NET Framework 3.5 Service Pack 1 (SP1) includes several significant improvements to WPF. Some of the graphics improvements include smoother animations, hardware accelerated rendering, text rendering speed improvements, 2D graphics improvements, layered window performance improvements, and several data binding and editing improvements.

The Microsoft .NET 3.5 SP1 provides support to integrate Direct3D directly into WPF.

## 18.4 WPF Architecture

WPF architecture spans across managed code and unmanaged code components. Figure 18.1 shows the core components in WPF.

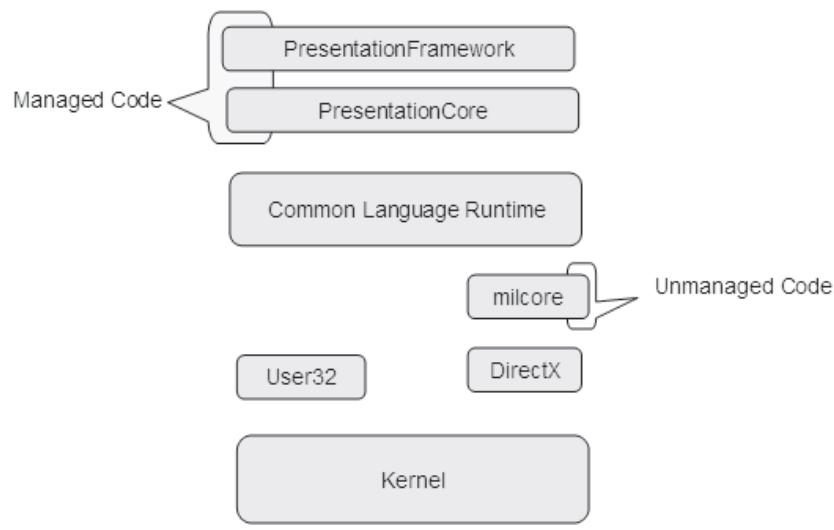


Figure 18.1: WPF Core Components

The programming model of WPF is exposed through three major components. They are as follows:

- **PresentationFramework**
- **PresentationCore**
- **milcore**

The **PresentationFramework** resides in `presentationframework.dll`. It implements presentation features for the end user such as shapes, imaging, animation, and themes.

The **PresentationCore** resides in `presentationcore.dll`. This assembly provides the features such as drawing, brushes, and layout for the UI elements.

The unmanaged code component is a composition engine used to render the WPF application. It is named as Media Integration Layer (MIL) and resides in `milcore.dll`. A `milcore.dll` has low level rendering code which interfaces directly with DirectX. This assembly provides support for 2D and 3D graphics, and animation services.

**Note - DirectX** is a set of APIs, which consists of commands and functions to create and manage graphical images and multimedia effects.

#### 18.4.1 Subsystems of WPF

The major subsystems or classes that form WPF are as follows:

- **System.Object**

The **Object** class is the base class for all .NET Framework classes and hence, is an important subsystem in WPF.

- **System.Threading.DispatcherObject**

The **DispatcherObject** subsystem is the basic message dispatching system that contains multiple prioritized queues. It handles messages from other objects. In WPF, most of the objects are derived from **DispatcherObject** class. It is the base class that handles messages from other objects.

- **System.Windows.DependencyObject**

The **DependencyObject** subsystem defines different properties for objects. This class allows two-way binding. By using binding objects, the properties of one object can be bound to the other object property.

- **System.Windows.Media.Visual**

Once the system is designed, the next step is to draw pixels. **Visual** is the main entry point to the WPF system which is connection between the managed code and **milcore**. The **Visual** class is the base class of all the visual elements in WPF.

→ **System.Windows.UIElement**

The `UIElement` subsystem provides layout, input, and events for UI components. Layout is the basic concept for position of any component. WPF introduces two- phase model of layout. These phases are Measure (to determine size) and Arrange (position of each child). `UIElement` also provides two phases for event handling – tunnel (Root to Target) and bubble (Target to Root).

→ **System.Windows.FrameworkElement**

The `FrameworkElement` subsystem is based on the `UIElement` system. This system provides features such as data binding, styles, and animation. This introduces a set of events, properties, and methods for WPF elements.

→ **System.Windows.Controls.Control**

The core components of GUI, `Controls`, have properties such as background, foreground, and padding. `ControlTemplate` is used for customization. The control implementation provides a data model, integration model, and display model. The separation of data model (properties), interaction model (events and commands), and display model (template) gives the customization of a control's behavior and look.

## *18.5 Extensible Application Markup Language*

XAML is a markup language based on XML for declarative application programming. You can create your own UI definitions in XAML.

XAML has more benefits than older markup languages such as HyperText Markup Language (HTML), Extensible Hypertext Markup Language (XHTML), and Scalable Vector Graphics (SVG). The features of XAML are as follows:

→ **UI and Business Logic Separation**

This is one of the greatest benefits of XAML. It separates design and development from each other. This provides more collaboration and efficiency between developers and designers of an application.

→ **High User Experience**

XAML files are basically simple XML format files, so transferring user interfaces between platforms is easy. To design user interfaces using XAML is easier and also needs lesser code.

→ **Easier Extension**

In XAML, .NET classes are placed in a hierarchical manner, where each element is the equivalent of a Common Language Runtime (CLR) class. So, extension of the .NET classes will be easier.

→ **Easier to implement Styles for UI**

XAML makes the development of any user interface much faster and easier. It provides features such as creating layout, applying styles, and templates for the UI application.

### 18.5.1 Basic Structure of XAML

A WPF stand-alone application contains windows or pages. A Window is a top-level window with the `<Window>` tag, whereas the Page is a browser-hosted page with the `<Page>` tag in an XAML file. Apart from Window and Page, XAML has ResourceDictionary and Application root elements for specifying the external dictionary and application definition.

The syntax for a Window is:

**Syntax:**

```
<Window x:Class="WpfApplication1.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="Window1" Height="300" Width="300">
    <Grid>
        .....
        .....
    </Grid>
</Window>
```

where,

`Window`: One of commonly used root element which contains other elements

`xmlns`: Namespace declared specifically for WPF

`xmlns:x`: Namespace with keywords and markup extensions in XAML. It includes mapping with the `x:` prefix.

Some commonly used prefixes are as follows:

`x:Class` – To associate with code-behind file

`x>Type` – To specify the type

`x:Null` – To assign a null value

In the syntax shown here,

`x:Class:` Specifies the related code-behind file. Here, `WpfApplication1` is the name of an application and `Window1` is the name of the class that binds the XAML file with the related code-behind file

`Title:` The title of the window

`Grid:` Displays tabular data in a row and column format

### 18.5.2 Window Class

The .NET Framework Class Library is a collection of classes, interfaces, and value types. The `System.Windows` namespace is one of the most useful namespaces in WPF. This namespace provides base element classes.

The `Window` class is present in the `System.Windows` namespace. This class enables a user to create, design, show, and handle the lifetime of windows and dialog boxes.

A `Window` is a point of communication between an application and a user. The `Window` class also provides services for appearance, behavior of window, and dialog boxes.

The `Window` class can be created using markup, markup and code-behind file, or only code.

The following code snippet shows the use of `Window` class using markup in XAML code.

#### Code Snippet:

```
<Window
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    Title="Window" Height="300" Width="300">
</Window>
```

### 18.5.3 Application Class

The `Application` class, a member of `System.Windows` namespace, specifies the WPF application functionality. It manages navigation, property, lifetime, and resources of an application. The services for startup and exit of an application are provided by this class.

A WPF application can be created using markup, markup and code-behind file, or only code.

The following code snippet shows the use of `Application` class in a WPF program.

#### Code Snippet:

```
<Application
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"/>
```

### 18.5.4 Attributes and Elements

The XAML syntax describes properties and objects. It also defines the relationship between them. The extension for an XAML file is .xaml. The XAML syntax element is divided into two parts. They are as follows:

#### → Attributes

The attribute element assigns the names for an event or value to the property. The attribute is mentioned using attribute name, an assignment operator, and the value of the attribute in quotation marks (""). The attributes are of two types - Property Attribute, which defines properties for the element and Event Attribute, which specifies handler for the element.

##### Syntax:

```
<...attribute_name="value"...>
```

where,

attribute\_name: Name of the property or a name of an event

value: A string value for attribute

#### → Elements

An XAML element instantiates a .NET CLR class. The syntax of declaring elements is the same as element syntax of markup languages such as HTML. One of the two types, property element, enables to assign other element as a value of a property. The second type, event element, handles an event of the control.

##### Syntax:

###### Property Elements

```
<typeName.propertyName>...</typeName.propertyName >
```

where,

typeName: Name of the element

propertyName: Property of an element

###### Event Elements

##### Syntax:

```
<typeName event="event handler">...</ typeName>
```

where,

typeName: Name of the element

event: Name of the event

The following example demonstrates the use of attributes and elements in a WPF application. Example 1 creates a button and a text box with some properties set. The Click event and the event handler (`btnTest_Click`) of the button are specified in the markup, whereas the code that implements the handler is defined in the code-behind.

### Example:

The steps to create a WPF application are listed as follows:

1. Create a new Visual Studio project by selecting **WPF Application** from **Visual Studio installed templates**.
2. Name the application and click **OK**.

Visual Studio creates a project and automatically adds files namely `Window1.xaml` and `App.xaml` to the solution.

Figure 18.2 shows `Window1.xaml` file in the WPF designer.

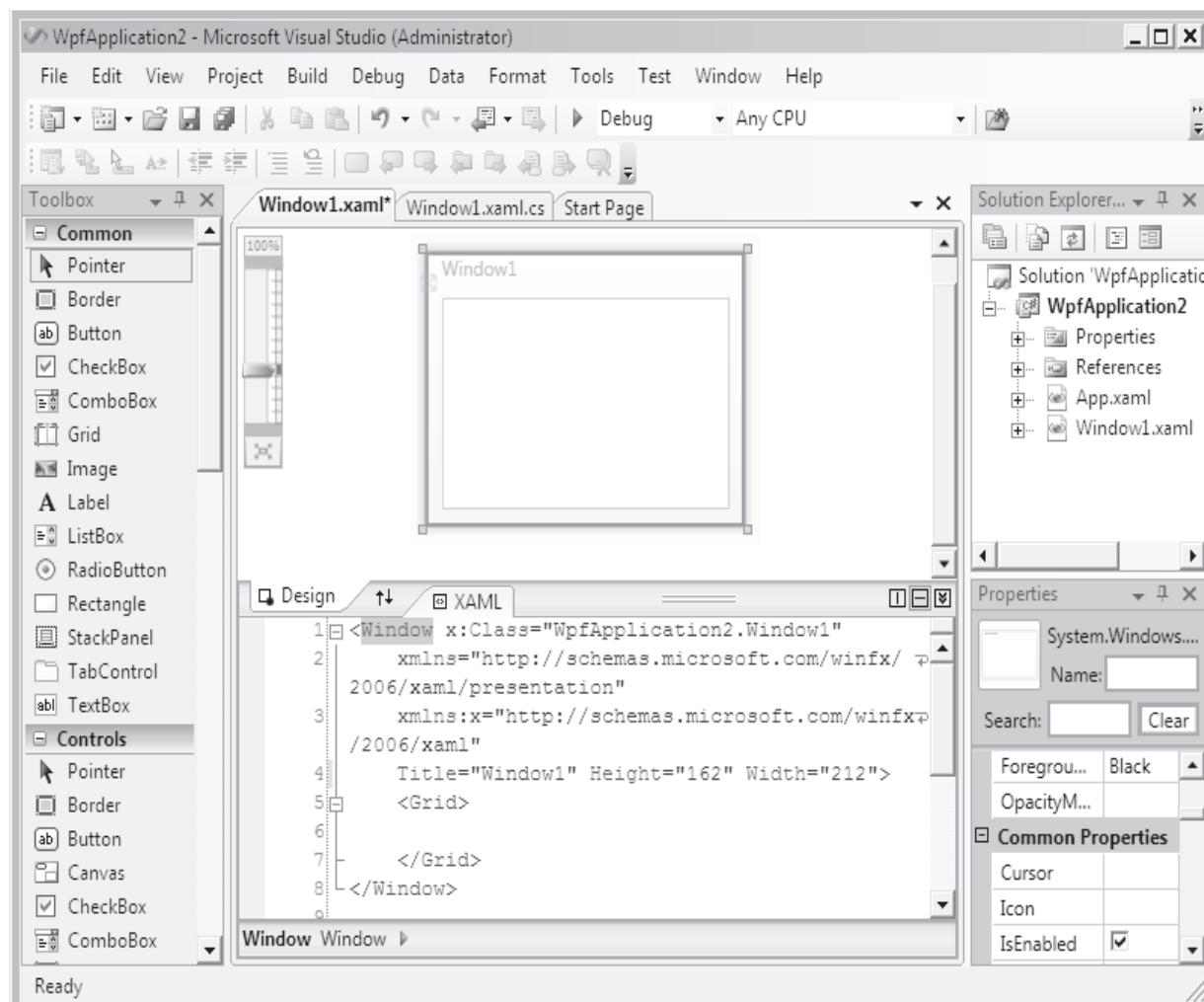


Figure 18.2: `Window1.xaml` in WPF Designer

The code for the WPF user interface is defined in Window1.xaml file. The App.xaml file defines a WPF application and its application resources. In this file, you can also specify the UI that automatically shows when the application starts.

3. Add the following code in Window1.xaml file.

#### XAML Code

```
<Window x:Class="XAMLApplication.TestWindow"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="TestWindow" Height="300" Width="300">
    <Grid>
        <TextBox Name="txtName" >
            <TextBox.VerticalAlignment>Top</TextBox.VerticalAlignment>
            <TextBox.Margin>70, 80</TextBox.Margin>
            <TextBox.Height>30</TextBox.Height>
            <TextBox.Width>150</TextBox.Width>
        </TextBox>
        <Button Name="btnTest" Click="btnTest_Click" >
            <Button.Height>30</Button.Height>
            <Button.Width>100</Button.Width>
            <Button.Content>Click Here</Button.Content>
        </Button>
    </Grid>
</Window>
```

In the code, `Height="300"`, `Width="300"` are the attributes defined for the Window. The property elements defined to the text box are `<TextBox.VerticalAlignment>`, `<TextBox.Margin>`, `<TextBox.Height>`, and `<TextBox.Width>` and the property elements for the button are `<Button.Height>`, `<Button.Width>`, and `<Button.Content>`. The handler for the button is defined as `Click="btnTest_Click"`. This has to add in Window1.xaml.cs file.

**C# Code**

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;

namespace XAMLApplication
{
    /// <summary>
    /// Interaction logic for TestWindow.xaml
    /// </summary>
    public partial class TestWindow : Window
    {
        public TestWindow()
        {
            InitializeComponent();
        }

        private void btnTest_Click(object sender, RoutedEventArgs e)
        {
            MessageBox.Show(txtName.Text, "WPF");
        }
    }
}
```

```
}
```

```
}
```

```
}
```

In the code, the `Click` event of the button is handled. When the user clicks the button, a message box with the text box content is displayed.

The output of the program is displayed as shown in figure 18.3.

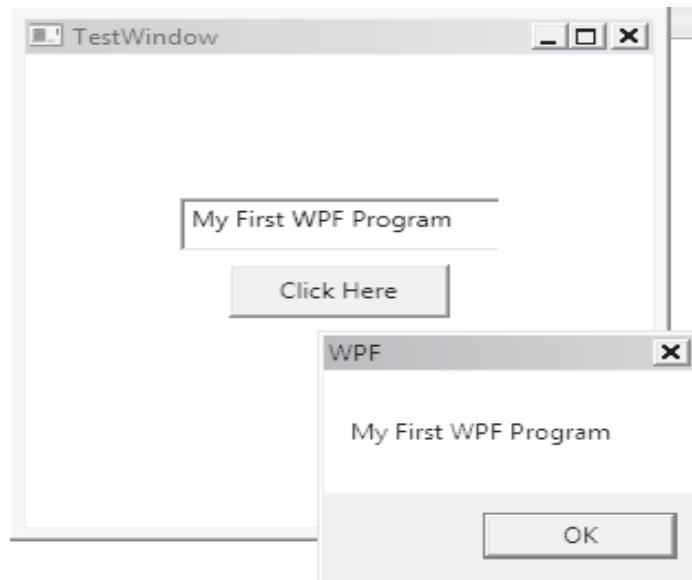


Figure 18.3: WPF Program

### 18.5.5 Routed Events

The WPF user interface is constructed in a layered approach, where each visual element can have zero or more child elements. A visual tree in WPF is the hierarchy of visible layers in a user interface, whereas a logical tree describes the relations between elements of the user interface.

Consider the following snippet:

#### Code Snippet:

```
<Window>
    <Grid>
        <Button Content="Button" />
        <Label Content="Label" />
    </Grid>
</Window>
```

The logical and visual trees for the snippet are represented in figure 18.4.

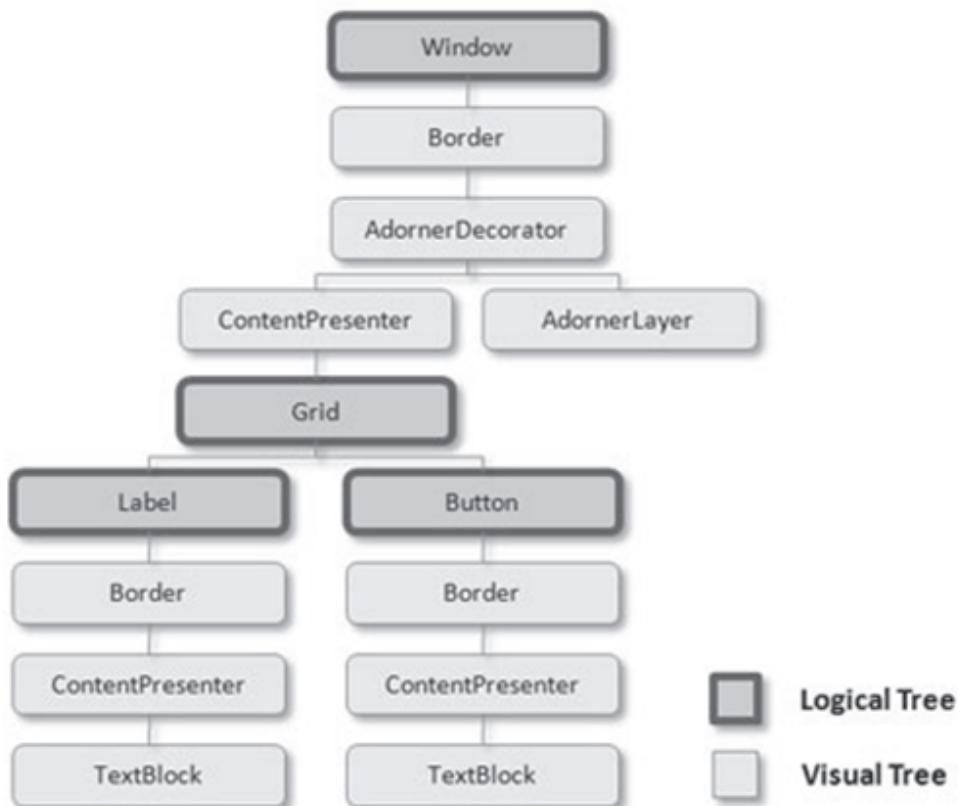


Figure 18.4: Logical and Visual Trees

Routed events in WPF are a new model in which events are tunneled down in the visual tree to reach the target element, or they can bubble up to the root element.

The three types of routed events are as follows:

#### → **Direct Events**

These are simple events that are raised on a single source. They are closest to standard .NET events, however, they are registered with the WPF routed-event system.

#### → **Tunneling Events**

These events travel from the root of the visual tree to the target element.

#### → **Bubbling Events**

These events travel from the target element to the root element.

Figure 18.5 shows the routed events in a visual tree.

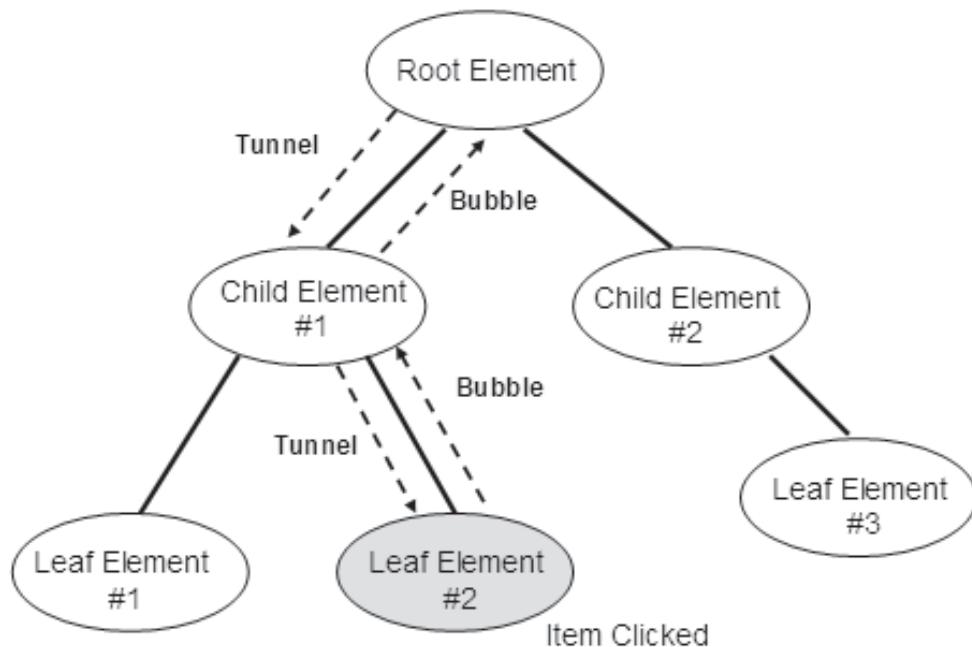


Figure 18.5: Routed Events

The event handler signature for routed events has the same pattern of .NET event handlers. The first parameter named `sender` is an instance of `System.Object`, and the second parameter named as `e` is a derived class of `System.EventArgs`. The `e` parameter is an instance of `RoutedEventArgs` class which is a subclass of `EventArgs` class.

The properties exposed by `EventArgs` class are tabulated in table 18.1.

Property	Description
<code>Source</code>	An element that raised the event from the logical tree
<code>OriginalSource</code>	An element that raised the event from the visual tree
<code>Handled</code>	A Boolean value to handle the event. This can be used to halt tunneling or bubbling.
<code>RoutedEventArgs</code>	The routed event object

Table 18.1: EventArgs Properties

The properties, `Source` and `OriginalSource`, will help you to work with logical tree and visual tree.

The following example demonstrates the use of bubbling event. The program associates an event handler to a Window's MouseRightButtonDown event.

**Example:**

**XAML Code**

```
<Window x:Class="WPFExample.SummaryDialog" MouseRightButtonDown="SummaryDialog_MouseRightButtonDown"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Routed Events" Height="300" Width="300">
    <StackPanel>
        <Label>Michigan Enterprises Limited</Label>
        <ListBox>
            <ListBoxItem>Windows</ListBoxItem>
            <ListBoxItem>Web</ListBoxItem>
        </ListBox>
        <StackPanel Orientation="Horizontal" HorizontalAlignment="Center">
            <Button MinWidth="75" Margin="10">OK</Button>
            <Button MinWidth="75" Margin="10">Cancel</Button>
        </StackPanel>
        <StatusBar>Copyright (c) Michigan Enterprises Ltd.</StatusBar>
    </StackPanel>
</Window>
```

**C# Code**

```
public partial class SummaryDialog : Window
{
    public SummaryDialog()
    {
        InitializeComponent();
    }
}
```

```
// Define the Mouse Event

private void SummaryDialog_MouseRightButtonDown(object sender,
MouseButtonEventArgs e)
{
    // Display information about this event

    this.Title = "Source = " + e.Source.GetType().Name + ", OriginalSource = " +
e.OriginalSource.GetType().Name + "@" + e.Timestamp;

    // Derive all possible sources from Control in this example
    Control source = e.Source as Control;

    // Toggle the border on the source control
    if (source.BorderThickness != new Thickness(5))
    {
        source.BorderThickness = new Thickness(5);
        source.BorderBrush = Brushes.Black;
    }
    else
        source.BorderThickness = new Thickness(0);
}
}
```

The output of the program is displayed as shown in figure 18.6.

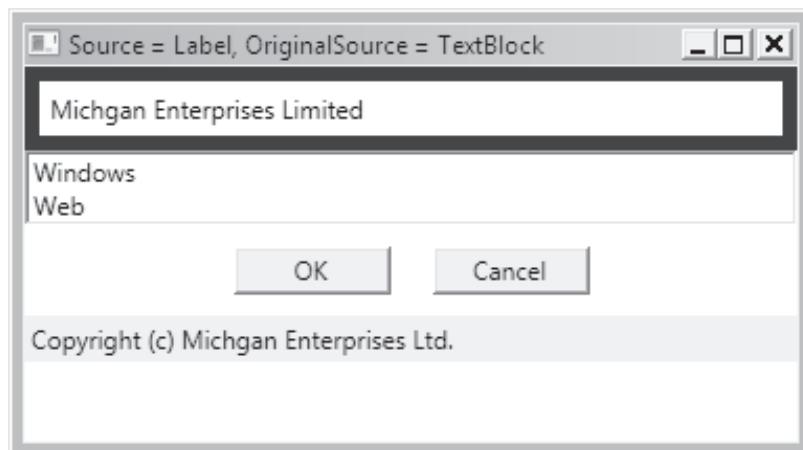


Figure 18.6: Output with the first label click

The event handler, `SummaryDialog_MouseRightButtonDown` performs two actions when a right-click bubbles up to the Window. It displays the source and original source of the element in the title bar and it adds (and then subsequently removes) a thick blue border around the element in the logical tree that was right-clicked. In this case, the user has right-clicked the first label, a blue border appears around the label and the title bar is displayed with the `Source` as `Label` and `OriginalSource` as `TextBlock`.

When the `ListBoxItem` is right-clicked, the `Window` will not receive the `MouseRightButtonDown` event because the event is internally handled by `ListBoxItem`.

When the button is right-clicked, `Window` receives the `MouseRightButtonDown` event, but there is no visual effect on `Button`'s `Border` property. This is because `Button` has no `Border` element in visual tree, whereas elements such as `Window`, `Label`, `ListBox`, `ListBoxItem`, and `StatusBar` has the `Border` element.

### 18.5.6 Role of XAML in WPF

The foundation for WPF applications is XAML. The XAML syntax is almost identical to HTML and ASP.NET. XAML defines the UI elements, database binding, and other features for WPF.

There are different tools that can be used to create and edit XAML files for WPF applications. They are as follows:

- Microsoft Expression Blend
- Microsoft Visual Studio 2008 or higher
- Standard text editors
- Developer tool - XAMLPad

In WPF, XAML describes the visual UI which defines 2D and 3D objects, and animations.

XAML files can be compiled into Binary Application Markup Language (BAML). This BAML is stored as a resource into .NET Framework assembly. At runtime, when a resource is requested, the `.baml` file is extracted and the Framework engine parses it and creates WPF visual tree.

Figure 18.7 shows the compilation process of a WPF application.

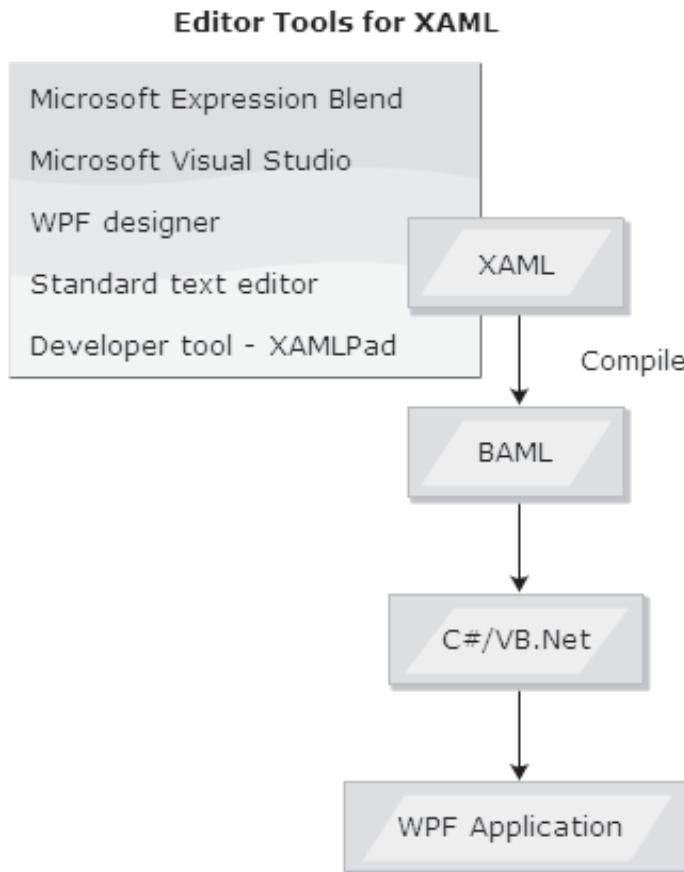


Figure 18.7: Compilation Process

## 18.6 Controls in WPF

WPF has a rich set of UI controls. These controls are grouped into various categories depending on their functionality, as shown in table 18.2.

Category	Controls
Layout	Border, Canvas, DockPanel, Grid, GroupBox, Panel, ResizeGrip, Separator, ScrollBar, ScrollViewer, StackPanel, Thumb, Viewbox
Buttons	Button, RepeatButton
Menus	ContextMenu, Menu, ToolBar
Selection	CheckBox, ComboBox, ListBox, ListView, TreeView, RadioButton
Navigation	Frame, Hyperlink, Page, NavigationWindow, TabControl
Input	TextBox, RichTextBox, PasswordBox

Category	Controls
User Information	AccessText, Label, Popup, ProgressBar, StatusBar, TextBlock, ToolTip
Media	Image, MediaElement, SoundPlayerAction

Table 18.2: Controls in WPF

### 18.6.1 Basic Controls

WPF has some basic controls that are common to Windows Forms. They are as follows:

→ **Button**

A `Button` is a basic UI control which contains content such as text, image, or panel. A `Button` raises a `Click` event to give response to input from the user. The user input can come from mouse, keyboard, or other input devices. The `Button` control is inherited from the `ContentControl` class.

→ **TextBox**

The user can enter or edit text using the `TextBox` control. It also provides formatting to text input.

→ **Label**

The `Label` control is used to display information in the UI. It is inherited from the `ContentControl` class.

→ **Image**

The `Image` control allows you to display images on the user interface. It supports image files such as `.bmp`, `.gif`, `.jpg`, and so on.

The following example creates a Label, TextBox, Button, and an Image in the user interface. Assume that an image, test.jpg, is placed in C drive.

**Example:**

```
<Grid>
    <Label Content="Name:" Margin="13,31,0,0" Height="28"
        HorizontalAlignment="Left" VerticalAlignment="Top" Width="52"/>

    <TextBox Name="txtName" Margin="69,33,113,0" Height="30"
        VerticalAlignment="Top">
        This is a TextBox.
    </TextBox>

    <Button Name="btnClick" Content="Click Me"
        Margin="69,85,113,0" Height="30" VerticalAlignment="Top" />

    <Image Name="imgPhoto" Source="c:\\test.jpg"
        Margin="95,126,126,100">
    </Image>
</Grid>
```

The output of the code is displayed as shown in figure 18.8.

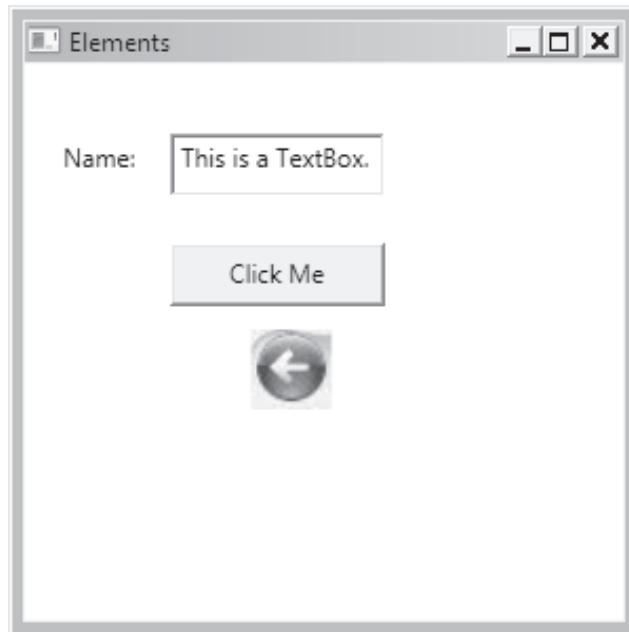


Figure 18.8: Basic Elements

### 18.6.2 New Controls in WPF

WPF provides some controls which were not available for earlier Windows-based applications. They are as follows:

→ **TextBlock**

A **TextBlock** is used to display a string or a small amount of flow content elements such as **Bold**, **Italic**, and **Underline** through its content property named **Inlines**. This property is used to add elements in the **TextBlock**.

→ **Slider**

The **Slider** control is used to select a value from a range. This control has properties such as **Maximum** and **Minimum** which define a range of the values represented by the control. The **Value** property is used to hold current selected value. An **Orientation** property is used to select vertical or horizontal modes.

→ **TabControl**

A **TabControl** is used to hold multiple controls that share same screen space. A **TabItem** is an object which represents each control. At a time, only one **TabItem** is displayed.

→ **GridSplitter**

A **GridSplitter** control is used to change size of rows or columns of a grid. You can define horizontal and vertical mode of a **GridSplitter** by using properties **HorizontalAlignment** and **VerticalAlignment** respectively.

**Example:**

The following code creates a **TextBlock**, **Slider**, **TabControl**, and a **GridSplitter** in a **Grid** control.

```
<Grid>
    <TextBlock Name="txtBlock" Margin="11,14,116,0" Height="27"
    VerticalAlignment="Top">
        This is a TextBlock control.
    </TextBlock>

    <Slider Margin="21,47,34,0" Height="25" VerticalAlignment="Top"/>
    <TabControl Name="tabControl1" Margin="24,84,41,125">
        <TabItem>
            <TabItem.Header>
```

```
<StackPanel Orientation="Horizontal">
    <TextBlock>Tab 1</TextBlock>
</StackPanel>
</TabItem.Header>
<StackPanel Height="47" Width="215">
    <TextBox Name="textBox1" Width="156" Height="25">
        This is First Tab
    </TextBox>
</StackPanel>
</TabItem>
<TabItem Header="Tab 2">
</TabItem>
</TabControl>

<Grid Height="66" Margin="37,0,41,27" VerticalAlignment="Bottom"
Background="AliceBlue">
    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="86*" />
        <ColumnDefinition Width="114*" />
    </Grid.ColumnDefinitions>
    <GridSplitter Grid.Column="0"
Background="Blue" Width="5" HorizontalAlignment="Right"
VerticalAlignment="Stretch"/>
</Grid>
</Grid>
```

The output of the code is displayed as shown in figure 18.9.

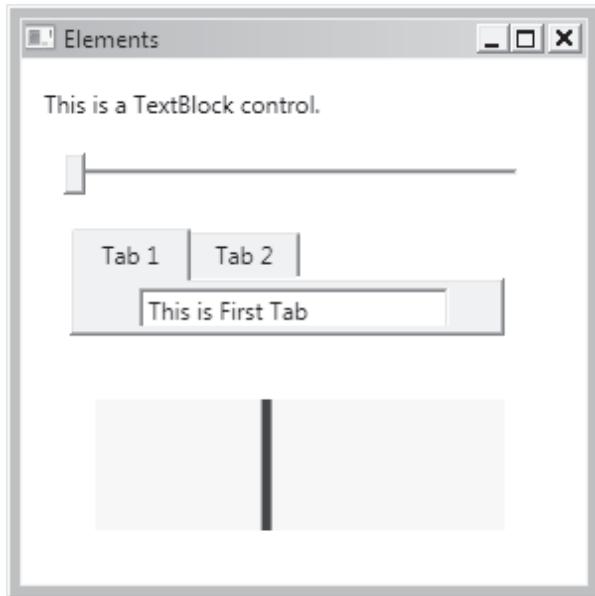


Figure 18.9: More Elements

### 18.6.3 Container Controls

WPF provides the user with dynamic layout functionality. It provides container controls to achieve this. The layout and positioning properties are provided by the container controls. `Canvas`, `Grid`, and `DockPanel` are some of the container controls in WPF.

The `Canvas` control is a simple container control. It handles absolute `x` and `y` positions and layout for its contained controls. Each contained control has four attached properties – `Top`, `Bottom`, `Left`, and `Right`. In a canvas, contained controls can have a position at an offset from any corner of the panel.

Another property called as the `ZIndex` is used to avoid overlapping of the contained controls. This property decides the order of contained controls in the canvas.

#### Example:

The example will create a canvas containing two child controls – `Rectangle` and `Ellipse`. An ellipse will be placed before the rectangle because its `ZIndex` value is greater.

```
<Canvas Margin="0,0,0,0" Background="White">
    <Rectangle Fill="Red" Width="100" Height="100" Canvas.ZIndex="0" Canvas.Left="22" Canvas.Top="20"/>
    <Ellipse Fill="Blue" Width="100" Height="100" Canvas.ZIndex="1" Canvas.Left="80" Canvas.Top="53"/>
</Canvas>
```

The output of the code is displayed as shown in figure 18.10.

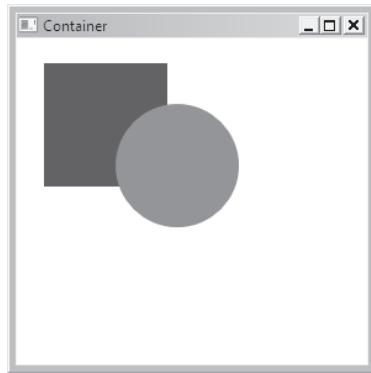


Figure 18.10: Container Controls

#### 18.6.4 Grid Control

The `Grid` control is the default container control. This is like the HTML table control, in which you can have rows and columns, and cells. In a `Grid` control, a single cell can contain more than one control.

The `Grid` control allows placing of a child control in a specified cell. When the Window is resized, controls contained in cells maintain a fixed margin between control edges and cell edges.

The four main properties of the `Grid` control are `Row`, `Column`, `RowSpan`, and `ColumnSpan`. `Row` and `Column` properties of any control define the cell they belong to within the grid. `RowSpan` and `ColumnSpan` specify a number of rows or columns of a cell. For example, a `RowSpan` of 3 means the cell spans 3 rows.

Other properties such as `RowDefinitions` and `ColumnDefinitions` are used to specify layout of the rows and columns respectively.

##### Example:

The given snippet will create a grid control with `RowDefinitions` and `ColumnDefinitions`. The first Rectangle renders in row 0, column 0 and second one renders in the same row, column 1 of the grid.

```
<Grid Width="264" Height="108" Background="Wheat">
<Grid.RowDefinitions>
    <RowDefinition Height="Auto"/>
    <RowDefinition Height="Auto"/>
</Grid.RowDefinitions>
<Grid.ColumnDefinitions>
    <ColumnDefinition Width="75" />
    <ColumnDefinition Width="175"/>
</Grid.ColumnDefinitions>
```

```
<Rectangle Fill="BlueViolet"
    Grid.Row="1" Height="21" VerticalAlignment="Top" Margin="0,13,0,0"/>
<Rectangle Fill="Plum" Grid.Column="1" Height="30"
    HorizontalAlignment="Left" Grid.Row="1" VerticalAlignment="Bottom"
    Width="134" Margin="10,0,0,-38" />
</Grid>
```

The output of the code is displayed as shown in figure 18.11.



Figure 18.11: Grid Control

### 18.6.5 DockPanel Control

The `DockPanel` control is useful to place a child control relative to other child controls in the same container. This provides docking for the easy placing of toolbars or main content to the side of the panel.

The `DockPanel` control has two main properties: `DockStyle` and `LastChildFill`. `DockStyle` property defines the position for the contained control. When `LastChildFill` property is set to True, it will make the last added control fill the remaining space of the container.

The `Dock` property determines the position of the child control.

#### Example 7:

The given code creates two `Rectangle` controls and places them within `DockPanel`. The `LastChildFill` property is set to True; this will make the second rectangle fill the remaining space.

```
<Grid>
    <DockPanel LastChildFill="True">
        <Rectangle Fill="Blue" Height="20" DockPanel.Dock="Top"/>
        <Ellipse Fill="Black" DockPanel.Dock="right" Height="70" Width="98" />
        <Rectangle Fill="PaleGoldenrod" DockPanel.Dock="Left"/>
    </DockPanel>
</Grid>
```

The output of the code is displayed as shown in figure 18.12.

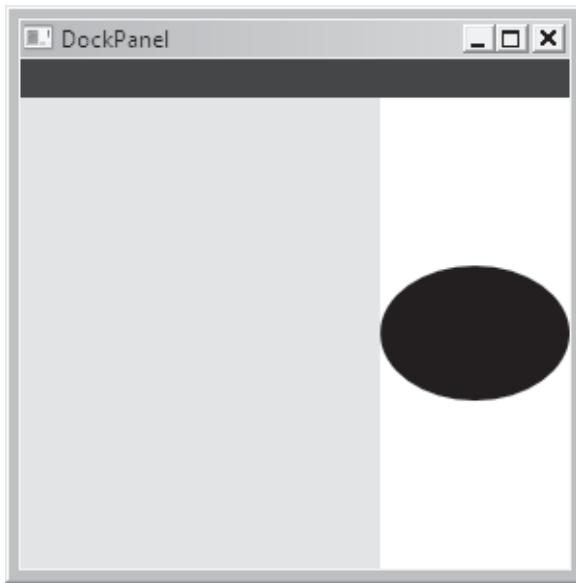


Figure 18.12: DockPanel Control

## 18.7 Styles in WPF

In WPF, styles have the same capabilities as that of Cascading Style Sheets (CSS) for Web-based development. A style is used to set properties for several controls in an organized manner. Table 18.3 shows the various properties of styles.

Property	Purpose
BasedOn	Specifies or retrieves a base style.
Dispatcher	Retrieves the Dispatcher inherited from <code>DispatcherObject</code> .
IsSealed	Used to check whether the style is read-only or not.
Resources	Specifies or retrieves the collection of resources that can be used within the scope of this style.
TargetType	Specifies or retrieves the type for which this style is intended.
Setters	Retrieves a collection of <code>Setter</code> and <code>EventSetter</code> objects.

Property	Purpose
Triggers	Retrieves a collection of TriggerBase objects that apply property values based on given conditions.

Table 18.3: Style Properties

### 18.7.1 Types of Styles

Styles can be applied to a group of controls or to an individual control. There are three types of styles namely inline, named, and element.

#### → **Inline Styles**

Inline styles apply a unique style to a control. Hence, each control in WPF for which you can apply styles has a Style property.

##### **Code Snippet:**

The following code applies a style to a TextBox.

```
<TextBox>
    <TextBox.Style>
        <Style><Setter Property="FontSize" Value="15" />
    </Style>
</TextBox.Style>
</TextBox>
```

#### → **Named Styles**

Named styles are defined by adding inline styles into resources. A named style has a name which is later used in control instances. Using this style, you can apply a set of properties to a specific control.

##### **Example:**

The following code will create a key, named stylebutton and this key will be used further to apply style on any Button.

```
<Window.Resources>
    <Style x:Key="stylebutton">
        <Setter Property="Control.FontSize" Value="30" />
        <Setter Property="Control.Foreground" Value="Blue" />
    </Style>
</Window.Resources>
```

```
<Button Style="{StaticResource stylebutton}" Height="55" Width="201"
x:Name="btnTest" Content="Test"/>
```

In this code, a static resource has been defined. Resources are used to store commonly used objects and values within a collection. `Window.Resources` supports the WPF resources. `StaticResource` is a markup extension provides value for XAML property attribute by referring defined resource.

The output of the code is displayed as shown in figure 18.13.

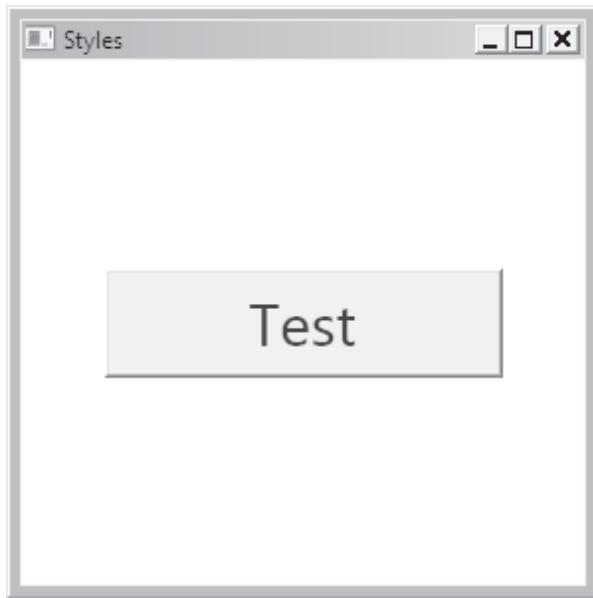


Figure 18.13: Named Styles

#### → Element Styles

The `Element` styles allow you to apply a style consistently to all instances of a particular type of control.

#### **Example:**

The following code will create a group of styles that can be applied to all instances of a `TextBlock` control.

```
<Window.Resources>
  <!-- no Key -->
  <Style TargetType="{x:Type Button}">
    <Setter Property="FontSize" Value="32" />
    <Setter Property="Foreground" Value="Black" />
    <Setter Property="Background" Value="AliceBlue" />
  </Style>
</Window.Resources>
```

```
</Style>
</Window.Resources>

<Button x:Name="btnTest" Width="214" Height="78" Content="Click"/>
```

The button will be displayed with the respective properties.

### 18.7.2 Transforms

Transforms provides different alteration to the object. A transform provides mapping of object points from one coordinate space to another. This allows objects to rotate, move, and scale by using transforms classes.

TransformGroup is used to group multiple transforms for a single object. The different types of transforms are as follows:

#### → Rotation Transforms

The RotateTransform class rotates an object in X-Y positions. You can rotate an object by specifying its angle of rotation in clockwise degrees. This also allows to set specific X and Y points. The three properties are – CenterX, CenterY, and Angle.

The RenderTransform property gives information about rendering position of an object. The RenderTransformOrigin property specifies origin point of an object after rendering.

#### Example:

The following code snippet demonstrates use of RotateTransform. An image will rotate by 50 degrees anticlockwise from its upper-left corner. Here, the starting point of an object is specified in CenterX and CenterY properties.

```
<Grid>
    <Image Name="imgPhoto" Source="c:\\test.jpg" Margin="25,15,0,0"
    Height="49" VerticalAlignment="Top" HorizontalAlignment="Left"
    Width="56">
        </Image>
    <Image Name="imgPhoto1" Source="c:\\test.jpg" Margin="0,19,59,0"
    Height="49" VerticalAlignment="Top" HorizontalAlignment="Right"
    Width="56">
        <Image.RenderTransform>
            <RotateTransform Angle="-50" CenterX="50" CenterY="50" />
        </Image.RenderTransform>
    </Image>
</Grid>
```

```
</Image.RenderTransform>  
</Image>  
</Grid>
```

Figure 18.14 shows the output that displays two different images. The first image shows the output before applying the rotation effects. The second image shows the output after applying the rotation effects.

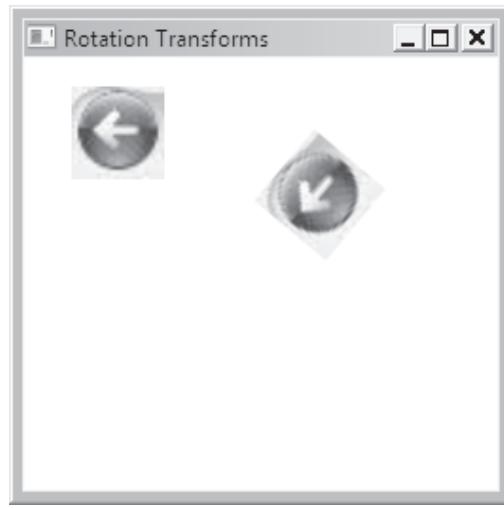


Figure 18.14: Rotation Transforms

#### → Scale Transforms

The `ScaleTransform` class is used to scale an object starting from specific center point. You can also specify different values using the properties – `ScaleX` and `ScaleY`.

##### Example:

The following example demonstrates the use of `ScaleTransform` effect.

```
<Grid>  
  
    <Image Source="C:\\test.jpg" Height="57" HorizontalAlignment="Left"  
          VerticalAlignment="Top" Width="71" Margin="21,36,0,0">  
        </Image>  
  
        <Image Source="C:\\test.jpg" Margin="120,40,45,0" Height="70"  
              VerticalAlignment="Top">  
  
            <Image.RenderTransform>  
              <ScaleTransform ScaleX="0.50" ScaleY="0.75"/>
```

```

</Image.RenderTransform>
</Image>
</Grid>

```

Figure 18.15 shows the output of the code. The first image shows the actual image and the second one shows the output after applying the scale transform effect.



Figure 18.15: Scale Transforms

#### → Skew Transforms

The `SkewTransform` class skews (or slants) an object by defines x-axis and y-axis angle. The two properties are – `AngleX` and `AngleY`.

#### Example:

The following example demonstrates the use of `SkewTransform` for an image. This will transform image by 20 degrees in X axis and 20 degrees in Y axis.

```

<Grid>
    <Image Source="c:\\Test.jpg" Margin="16,37,0,105"
        HorizontalAlignment="Left" Width="66"></Image>

    <Image Source="c:\\Test.jpg" Margin="106,35,75,104">
        <Image.RenderTransform>
            <SkewTransform AngleX="20" AngleY="20"/>
        </Image.RenderTransform>
    </Image>
</Grid>

```

Figure 18.16 shows the output of the code. The first image is the original image and the second

image shows the output after applying the skewed transform effect.



Figure 18.16: Skewed Tranforms

#### → Translate Transforms

The `TranslateTransform` class is used to translate the WPF object in the two-dimensional plane. The two properties to specify amount in pixel to translate an object are – `X` and `Y`.

##### Example:

The following example uses a `TranslateTranform` effect to move the image 50 pixels right and 50 pixels down.

```
<Grid>

    <Image Name="imgPhoto" Source="c:\\Test.jpg" Margin="10,11,0,0"
Height="66" VerticalAlignment="Top" HorizontalAlignment="Left"
Width="55" />

    <Image Name="imgPhoto1" Source="c:\\Test.jpg" Margin="91,7,98,0"
Height="66" VerticalAlignment="Top">
        <Image.RenderTransform>
            <TranslateTransform X="50" Y="50" />
        </Image.RenderTransform>
    </Image>
</Grid>
```

Figure 18.17 shows the output of the code.



Figure 18.17: Translate Tranforms

The first image shows the image in the original location and the second image shows the output after applying the `TranslateTransform` effect.

## 18.8 Animations in WPF

Animation is one of the most useful abilities of the WPF technology. An object is animated by actions such as rotating, moving, changing size and color, and compressing. This makes the user interface more attractive.

WPF also allows you to create your own animation by inheriting animation base classes, called custom animation. To make an object animated, animation is applied to its individual properties.

Each property can have animation ability if it is a dependency property.

The class of the property must be inherited from `DependencyObject` and it should implement the `IAnimatable` interface which provides animation.

### 18.8.1 Types of Animations

In WPF, animation is applied using a `Storyboard` class. Every animation is a part of the `Storyboard` class. This class has two properties which are used to identify the object and its property for animation. These properties are as follows: `TargetName` and `TargetProperty`. `TargetName` retrieves the object name which is to be animated. `TargetProperty` sets or gets the property which is to be animated.

The different types of animations are as follows:

- `<Type>Animation`
- `<Type>AnimationUsingKeyFrames`

**→ <Type>AnimationUsingPath****→ <Type>AnimationBase**

<Type> is the type of property value which is animated by a class. Some of the types are Color, Double, Byte, Decimal, and Point.

**→ <Type>Animation**

This animation changes the object from the specified starting value to the ending value within a defined duration. This animation is also called as From/To/By animation.

The starting value is set by using From property and ending value is set by using To property. The By property is used to specify offset value.

**→ <Type>AnimationUsingKeyFrames**

These animations create transitions among any number of ending values. Ending values can be specified using key frames objects.

An animation's interpolation methods which define transition process can be controlled. The interpolation methods are as follows: discrete, linear, and spline.

**→ <Type>AnimationUsingPath**

This animation allows you to produce animated values using geometric path.

**→ <Type>AnimationBase**

This is a base animation class that allows you to create your own custom animation. This abstract class is the base for <Type>AnimationUsingKeyFrames and <Type>Animation.

### 18.8.2 Graphics

WPF provides new graphics features which were not available earlier for Windows developers. WPF supports creation of 2D or 3D vector graphics. By using geometries, you can create custom shapes.

The different graphics tools are as follows:

**→ Bitmap Effects**

Bitmap effects are used to apply visual effects on rendered WPF content. These tools provide pixel processing operations. An input and output of BitmapEffects is BitmapSource which is a set of pixels with a certain size and resolution. Visuals provide rendering support which is rendered in software.

**→ Brushes**

These tools provide different types of brushes to paint user interface objects. Every brush type

produces different types of output. Some of the types are as follows: `SolidColorBrush`, `LinearGradientBrush`, `RadialGradientBrush`, `ImageBrush`, and so on.

#### → **Drawings**

These tools allow drawing shapes, images, text, and media. WPF provides `MediaElement` to add video in an application. Drawings provide different types of objects. These are `GeometryDrawing`, `ImageDrawing`, `GlyphRunDrawing`, `VideoDrawing`, and `DrawingGroup`.

#### → **Geometries**

A geometry object is used to define the region for 2D graphics, clipping, and animating. The three categories for geometries classes are `Simple`, `Path`, and `Composite`. The `Simple` geometry provides basic shapes such as lines, rectangles, and circles. The `Path` geometry provides shapes with arcs and curves. The `Composite` geometry combines two geometry areas.

#### → **Images**

WPF Imaging provides a new set of components to use images within an application. It also supports different image formats, codec extensibility. This imaging enables user to rotate, crop, and convert images.

#### → **Shapes**

WPF offers a number of basic shape objects such as `Rectangle`, `Ellipse`, `Line`, `Path`, `Polygon`, and `Polyline`. The shape object allows user to draw basic shape to the screen. Every shape object has three properties: `Stroke`, `StrokeThickness`, and `Fill`.

**Note -** The `System.Windows.Media.Media3D` namespace defines 3D graphics primitives, transformations, and animations that can be used to create 3-D controls and graphics.

## 18.9 Hosting a WPF Control in Windows Form

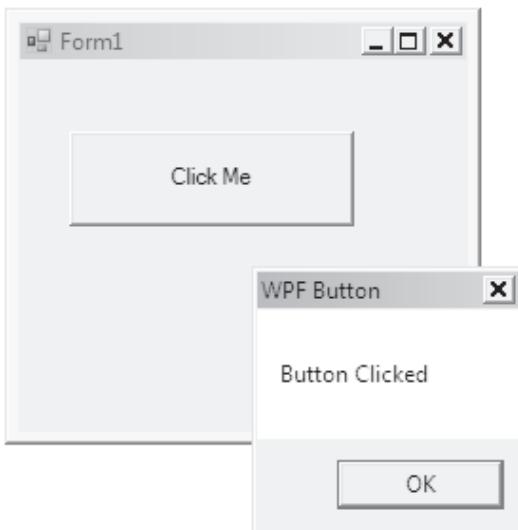
You can host a WPF control in Windows Forms by putting an `ElementHost` control to your Windows Forms application. You need to add references to `PresentationCore`, `PresentationFramework`, `WindowsBase`, and `WindowsFormsIntegration` assemblies. The following example creates a Windows Forms application that hosts a WPF `TextBox` control with some text in it. The program shows a `MessageBox` whenever the user clicks the `Button` element.

### Example:

```
public partial class Form1 : Form
{
    public Form1()
    {
        InitializeComponent();
        // Adding a WPF Button
        System.Windows.Controls.Button wpfButton = new System.Windows.Controls.Button();
        wpfButton.Name = "btnTest";
        wpfButton.Content = "Click Me";
        // Setting the Click Event
        wpfButton.Click += new System.Windows.RoutedEventHandler(btnTest_Click);
        ElementHost elementHost = new ElementHost();
        elementHost.Dock = DockStyle.None;
        elementHost.Width = 150;
        elementHost.Height = 50;
        elementHost.Child = wpfButton;
        pnlTest.Controls.Add(elementHost);
    }
    // Click Event Handler
    void btnTest_Click(object sender, System.Windows.RoutedEventArgs e)
    {
```

```
MessageBox.Show("ButtonClicked", "WPF Button");
}
}
```

The output of the program is displayed as shown in figure 18.18.



**Figure 18.18: WPF Button in Windows Form**

The form contains a panel control that holds the ElementHost control. The main code for creating the WPF Button and setting the properties and event handler is added in the form initialization. The ElementHost object is created and then, adds the Button as its Child property. Finally, the ElementHost control is added to the panel. When the user clicks the Button, a MessageBox is displayed that shows the text 'Button Clicked'.

### 18.9.1 Hosting a WPF User Control in Windows Form

WPF user controls can be hosted in Windows Forms applications. Hosting a WPF user control consists of two steps. The first step is to create a WPF user control and the second step is to add the user control in Windows Forms application. Example 15 shows the creation of a WPF user control.

#### Example 15:

The user control program uses a Grid element with a Label, TextBox, and two Button controls. The steps to create a WPF user control is shown as follows:

1. Create a new project in Microsoft Visual Studio 2008.
2. Select **Visual C# → Windows** from **Project types** and select **WPF User Control Library** template.
3. Name the project as and create the project.
4. Inherit the **UserControl1** from **Grid** element instead of **UserControl1**.

Make the change in both the files, **UserControl1.xaml** and **UserControl1.xaml.cs**.

Here the **Grid** element is chosen because in Windows Forms, you can host only a WPF element, not a control.

5. Include a **Label**, **TextBox**, and two **Button** controls in the user control. Arrange them on the grid appropriately.
6. Build the project.

You need to make sure that the following references are included in the user control project.

- **System**
- **PresentationCore**
- **PresentationFramework**
- **WindowsBase**

The second step is to create a Windows Forms application and add the user control in it. Example 16 shows the creation of a Windows Forms application with the WPF user control in it.

**Example:**

The steps to create a Windows Forms application is shown as follows:

1. Create a Windows Forms Application in Visual Studio 2008.
2. Name the project as **WinFormWPFUserCtrlHost** and create the project.
3. Drop an **ElementHost** control to the form. If this element is not seen in the Toolbox by default, add it by right-clicking the Toolbox and selecting 'Choose Items' from the list displayed.
4. Add the **WPFUserControl** project created in example 15.
5. Add references to the following libraries.
  - WindowsBase
  - WindowsFormsIntegration

The code for the application is as follows:

```
public partial class WinFormWPFUserCtrlHost : Form
{
    WPFUserControl.UserControl1 WPFUserControl = null;

    public WinFormWPFUserCtrlHost()
    {
        InitializeComponent();

        elementHost1.Dock = DockStyle.Fill;
        WPFUserControl = new WPFUserControl.UserControl1();
        elementHost1.Child = WPFUserControl;
    }
}
```

6. Compile and run the application. The output of the program is shown in figure 18.19.

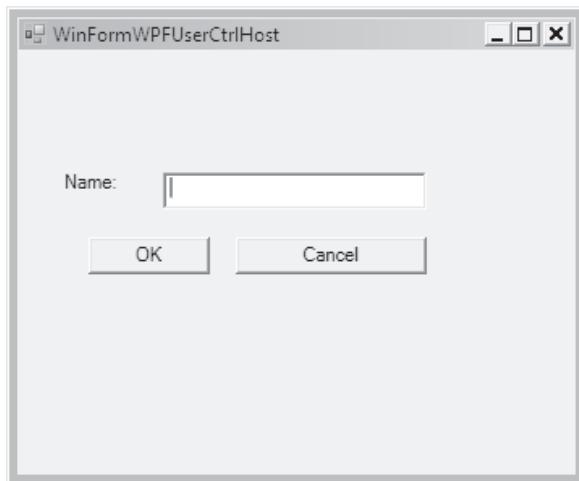


Figure 18.19: WPF User Control in Windows Forms

## Knowledge Check 1

1. Which of the following statements about WPF are true?

(A)	WPF 3.5 supports a number of international languages.
(B)	WPF provides a common platform for 2D/3D graphics, animation, video, and documents.
(C)	WPF supports video and audio handling.
(D)	WPF is a complete replacement for Windows Forms.

2. Which class is the base class of all the visual elements in WPF?

(A)	Control
(B)	Shape
(C)	Visual
(D)	Window

3. Which of the statements related to container controls are true?

(A)	The ZIndex property of the Canvas control is used to avoid overlapping of the contained controls.
(B)	The DockStyle property defines the position for the parent control.
(C)	The Canvas control is the default container control.
(D)	In a Grid control, a single cell can contain more than one control.

4. Match the transforms of WPF against their corresponding properties?

(A)	Scale Transform	1.	CenterX, CenterY, Angle
(B)	Translate Transform	2.	ScaleX, ScaleY
(C)	Rotation Transform	3.	AngleX and AngleY
(D)	Skew Transform	4.	RenderTransformOrigin
(E)	Render Transform	5.	X and Y

5. Which of the following statements about XAML are true?

(A)	XAML is XHTML-DOM based markup language.
(B)	XAML provides user interface and business logic separation.
(C)	XAML files are compiled into Byte Application Markup Language (BAML).
(D)	XAML describes visual UI which defines 2D and 3D objects, and animations.

6. Which of the statements about the Window class and the Application class are true?

(A)	Application class provides services for appearance, behavior of window and dialog boxes.
(B)	Window class and Application class are present in a System.Windows namespace.
(C)	Application class manages lifetime, navigation, property, and resource of an application.
(D)	Window class can be implemented using markup, markup and code-behind file, or only code.

7. Which of these statements about Styles are true?

(A)	Styles can be applied to only an individual control.
(B)	Inline styles apply a uniform style to all controls.
(C)	A Style is used to set properties for several control in an organized manner.
(D)	BasedOn property is used to get or set a base style.

8. Match the category of WPF controls with their descriptions.

(A)	To manage position, size, and dimensions of an element	1.	Menus
(B)	To select one or more than one options	2.	Buttons
(C)	To enable user to enter content or text	3.	Layout
(D)	To group related tasks	4.	Selection
(E)	To perform some task on clicking	5.	Input

9. Identify the correct code to host a WPF User Control in Windows Forms.

```
(A) public partial class WinFormWPFUserCtrlHost : Form
{
    string WPFUserControl = null;
    public WinFormWPFUserCtrlHost()
    {
        InitializeComponent();
        WPFUserControl = new UserControl.UserControl1();
        elementHost1.Child = WPFUserControl;
    }
}
```

(B)	<pre>public partial class WinFormWPFUserCtrlHost : Form {     UserControl.UserControl1 WPFUserControl = null;     public WinFormWPFUserCtrlHost()     {         InitializeComponent();         WPFUserControl = new UserControl();         elementHost1.Child = WPFUserControl;     } }</pre>
(C)	<pre>public partial class WinFormWPFUserCtrlHost : Form {     UserControl.UserControl1 WPFUserControl = null;      public WinFormWPFUserCtrlHost()     {         InitializeComponent();         WPFUserControl = new UserControl.UserControl1();     } }</pre>
(D)	<pre>public partial class WinFormWPFUserCtrlHost : Form {     UserControl.UserControl1 WPFUserControl = null;      public WinFormWPFUserCtrlHost()     {         InitializeComponent();         WPFUserControl = new UserControl.UserControl1();         elementHost1.Child = WPFUserControl;     } }</pre>



## Module Summary

- WPF provides a common platform for 2D/3D graphics, animation, video, and documents which was not provided by Windows Forms or ASP.NET.
- XAML is the foundation for WPF applications.
- Transforms allow objects to rotate, move, and scale by using transforms classes.
- The Window class enables user to create, design, show, and handle the lifetime of windows and dialog boxes.
- The Application class provides functionality for WPF application.
- Interoperability in WPF enables users to add a WPF control into a Windows Forms application and vice versa.
- You can use the ElementHost control to host a WPF user control into Windows Forms.

WRITE-UPS BY

**EXPERTS AND LEARNERS**TO PROMOTE NEW AVENUES AND  
ENHANCE THE LEARNING EXPERIENCE

FOR FURTHER READING, LOGIN TO

**[www.onlinevarsity.com](http://www.onlinevarsity.com)**

# Module - 19

## Windows Presentation Foundation (Lab)

In this Module, you will learn about:

- Create a WPF application
- Create and use WPF user control

## Part I – 60 Minutes

### Exercise

The Federation of International Football Association (FIFA) World Cup is a football tournament contested by the men's national team of each country. The matches are organized by FIFA every four years.

The 2010 World Cup involves 32 teams competing for the title for about a month. The World Cup football is the most viewed sporting event, an estimated 716 million people watched the final match of the 2006 World Cup. The association felt a need for creating an application that shows all the details of a team.

The association has asked Michigan Corporation to create an application that gives details of each team such as the country, coach, popular player, team photo, history, and so on.

Consider that you are one of the programmers in the development team and are assigned the following tasks:

- Create a database that stores the details of each team
- Design an interface for the application using WPF controls
- Create a class for defining properties and database connection
- Write code to display the details of each team

### Solution:

#### Design the WorldCupFootball database and Teams table

The details of each team needs to be stored in a database. To create a database named **WorldCupFootball**, perform the following steps:

1. Open Microsoft SQL Server Management Studio. Enter the appropriate login details.
2. Create a new database named **WorldCupFootball**.
3. Design a table named, **Teams** with the structure specified in table 19.1.

Column Name	Data Type	Width
Country	NCHAR	15
Coach	NCHAR	20
PopularPlayer	NCHAR	20
WorldCupMatches	INT	
Summary	VARCHAR	1200
Photo	VARBINARY	MAX

Table 19.1: Teams Table Structure

**4. Insert a record in the table using INSERT command.**

```
INSERT INTO [Teams]
(
    [Country]
    , [Coach]
    , [PopularPlayer]
    , [WorldCupMatches]
    , [Summary]
    , [Photo]
)
SELECT 'Brazil' AS COUNTRY,
    'Carlos Dunga' AS COACH,
    'Ricardo Izecson' AS PLAYER,
    18 AS APPEARANCES,
```

'The Brazil national football team represents Brazil in international Association football and is controlled by the Brazilian Football Confederation. They are the reigning South American champions, successfully defending their title in the 2007 Copa América. Brazil is the most successful national football team in the history of the World Cup, with five championships. Brazil, along with Argentina, are the only teams to win a World Cup outside their continental zone and the Verde-Amarela is the only team to have won the championship in four different continents; once in Europe (1958 Sweden), once in South America (1962 Chile), twice in North America (1970 Mexico and 1994 USA) and once in Asia (2002 S. Korea-Japan). A common quip about football is: "Os ingleses o inventaram, os brasileiros o aperfeiçoaram" ("The English invented it, the Brazilians perfected it"). Currently ranked first by FIFA, Brazil is consistently considered the strongest football nation in the world, and has also been marked as one of the most competitive teams of each decade since the 1960s. Brazil is the only national team to have played in every World Cup.'

```
* FROM OPENROWSET (BULK 'c:\brazil.gif', SINGLE_BLOB) as PHOTO
```

Here, the team photo is inserted into the Photo column using the OPENROWSET statement with BULK OLEDB provider. The OPENROWSET statement with the provider is used to insert documents and images in SQL Server.

**5. Insert more records and display the records in the table.**

```
USE WorldCupFootball
```

```
GO
```

```
SELECT * FROM TEAMS
```

Figure 19.1 displays the records of the Teams table.

	Country	Coach	Popular...	WorldC...	Summary	Photo
1	Argentina	Diego M...	Lionel ...	14	The Argentina natio...	0x474946383961C201F500F7000002020380000000800080...
2	Germany	Joachim ...	Philipp ...	16	The German nation...	0x474946383961C201F600F700000202028000000080008C...
3	Brazil	Carlos D...	Ricard...	18	The Brazil national f...	0x474946383961C301F500F7000002020280000000800082...

Figure 19.1: Teams Table

SQL Server stores the image in the `Photo` column as binary data.

#### Design the interface for the application

You need to create a WPF application to display the details of each team. The following steps should be performed to create the application:

- 1. Open Microsoft Visual Studio 2008 IDE. Click File → New → Project command from the menu. The New Project dialog box is displayed.**
- 2. Select WPF Application from Templates section. Name the project as WorldCupFootball and specify the location as shown in figure 19.2.**

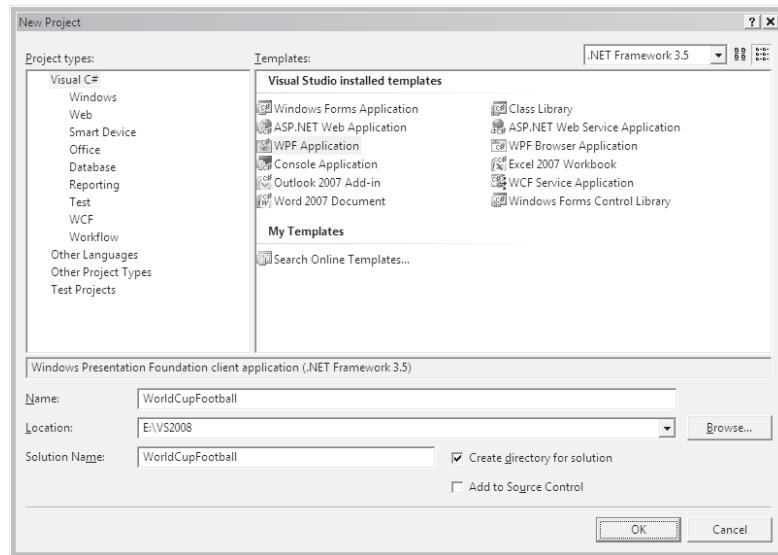


Figure 19.2: New Project Dialog Box

3. Click OK to create the application.

Figure 19.3 shows the interface of the application with Design, XAML, Solution Explorer, and Properties Windows.

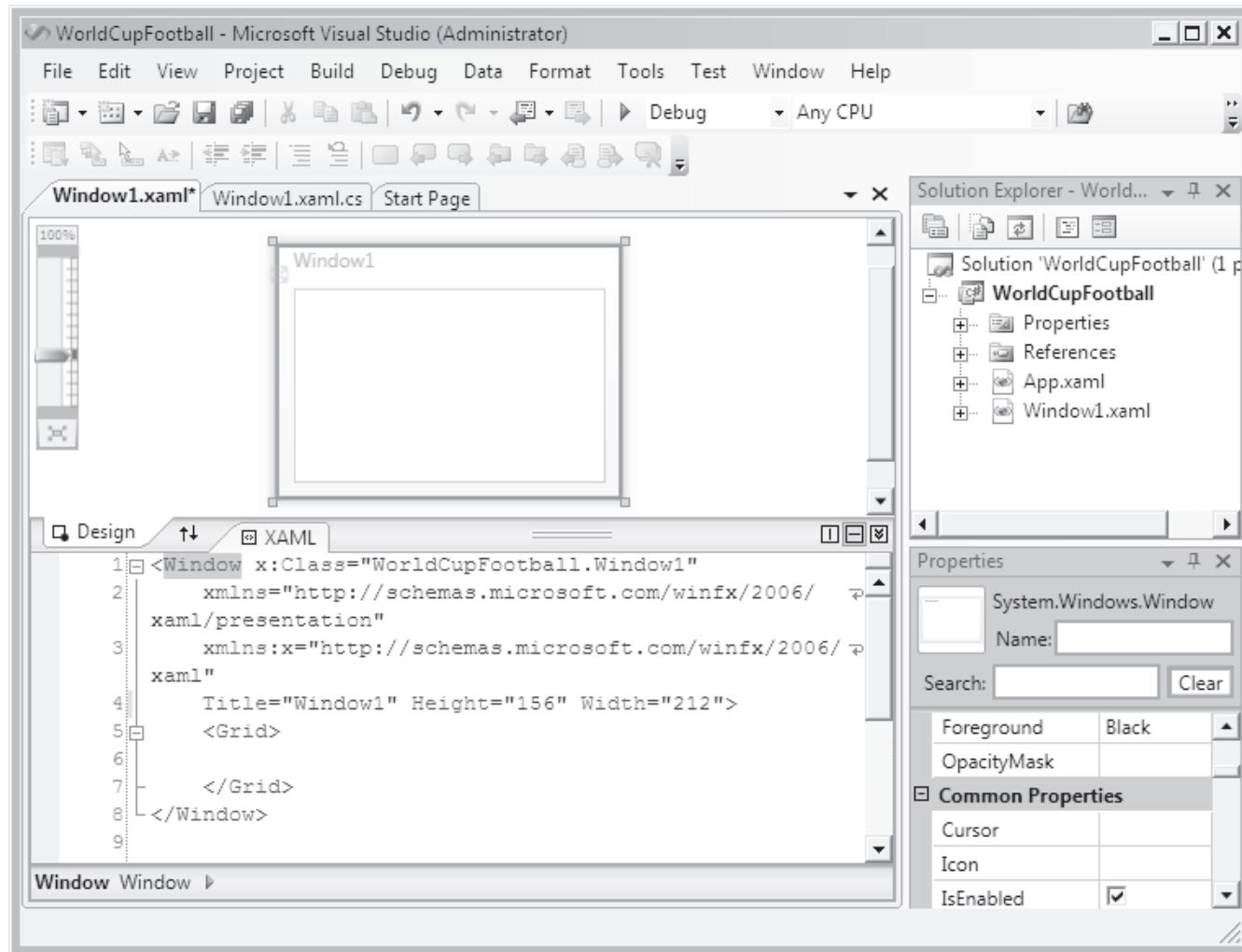


Figure 19.3: Application Interface

When you create a WPF application, two files named **Window1.xaml** and **App.xaml** are automatically created for you. **Window1.xaml** is the WPF form file and it contains a code-behind file named **Window1.xaml.cs**. The user interface of the application is designed through **Window1.xaml** file.

The **App.xaml** file is called the application file that defines the application-level resources such as styles. You can define global event handlers for the application using its code-behind file, **App.xaml.cs**.

4. Rename Window1.xaml file to Teams.xaml. Navigate to App.xaml file and change the StartupUri property to Teams.xaml.

```
<Application x:Class="WorldCupFootball.App"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    StartupUri="Teams.xaml">
    <Application.Resources>
        </Application.Resources>
</Application>
```

With StartupUri property, you can specify an XAML file for the application to load initially.

The default XAML code for Teams.xaml is shown as follows:

```
<Window x:Class="WorldCupFootball.Window1"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Title="Window1" Height="156" Width="212">
    <Grid>
    </Grid>
</Window>
```

The root element of an XAML document can contain certain elements such as Window, panels, or canvases.

The code `xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"` is mandatory for the XAML root element. The `<Window>` element represents the Window that replaces Windows Forms or ASP.NET Web Page in the previous versions of Visual Studio.

The `x:Class` attribute generates a class definition for this XAML file and it is derived from the type of the root element. The `<Grid>` element represents a Grid control in which the data can be displayed in rows and columns.

5. In Teams.xaml, in the XAML pane, in the `<Window>` element, add or change the values given in table 19.2.

Property	Value
<code>x:Class</code>	WorldCupFootball.Team
<code>Title</code>	Team Details

Property	Value
Height	517
Width	741
Name	TeamDetails
Topmost	True
BorderBrush	Chocolate

Table 19.2: &lt;Window&gt; Element Properties

Here, the class name is specified as Team and the Window name is specified as TeamDetails.

#### 6. Modify the <Grid> element to create two columns and eight rows.

```
<Grid x:Name="grTeam" Height="487" Width="721">

    <Grid.ColumnDefinitions>
        <ColumnDefinition Width="195*" />
        <ColumnDefinition Width="526*" />
    </Grid.ColumnDefinitions>

    <Grid.RowDefinitions>
        <RowDefinition Height="68*" />
        <RowDefinition Height="51.766*" />
        <RowDefinition Height="56.807*" />
        <RowDefinition Height="53*" />
        <RowDefinition Height="53*" />
        <RowDefinition Height="41*" />
        <RowDefinition Height="59*" />
        <RowDefinition Height="65*" />
    </Grid.RowDefinitions>

</Grid>
```

The <Grid.ColumnDefinitions> and <Grid.RowDefinitions> tags are used to define a collection of columns and rows in a <Grid> element respectively. The <ColumnDefinition> and <RowDefinition> tags are used to define column and row elements respectively. The name of the <Grid> element is specified as grTeam.

7. Add a `<Label />` control as a child of `<Grid>`, after the `</Grid.RowDefinitions>`, and then, add the values to the `<Label />` control given in table 19.3.

Property	Value
Content	Teams
Name	lblHeading
Margin	101,0,126,32
HorizontalContentAlignment	Center
Grid.ColumnSpan	2
Background	FloralWhite
FontWeight	Bold
FontFamily	Verdana
FontSize	20
Foreground	DarkBlue

Table 19.3: Label Properties

8. Add a `<Border />` after the `<Label />`, and then, add the values in table 19.4 to the `<Border />` element.

Property	Value
Name	br1
Height	3
Margin	-9,0,-11,23
VerticalAlignment	Bottom
Grid.ColumnSpan	2
BorderThickness	2
BorderBrush	BurlyWood
ClipToBounds	True

Table 19.4: Border Properties

**9. Add another five <Label /> controls after the <Border /> element.**

```
<Label Margin="35,1,69,21" Name="lblCountry" Grid.Row="1" Height="30"
Content="Country: " />
```

```
<Label Margin="35,3,0,23.734" Name="lblCoach" Grid.Row="2" Content
="Coach: " />
```

```
<Label Margin="35,1,0,21.932" Name="lblPopularPlayer" Grid.Row="3"
Content="Popular Player: " />
```

```
<Label Margin="35,2,0,20.932" Name="lblAppearances" Grid.Row="4"
Content="World Cup Appearances: " />
```

```
<Label Margin="35,2,22,9" Name="lblAbout" Grid.Row="5" Content="About:
" />
```

**10. Add a <TextBox /> for the Country label and add the values in table 19.5 to the <TextBox> element.**

Property	Value
Name	txtCountry
Grid.Column	1
Grid.Row	1
HorizontalAlignment	Left
Margin	0,3.163,0,23.163
Width	FloralWhite
Height	120
FontFamily	30

Table 19.5: TextBox Properties

- 11. Add another four <TextBox /> controls for Coach, Popular Player, World Cup Appearances, and About.**

```
<TextBox Grid.Column="1" Grid.Row="2" Margin="1,3,0,23.734"
Name="txtCoach" HorizontalAlignment="Left" Width="120" Height="30" />
```

```
<TextBox Grid.Column="1" Grid.Row="3" HorizontalAlignment="Left" Margin="1,5.368,0,22.3" Name="txtPlayer" Width="120" />
```

```
<TextBox Grid.Column="1" Grid.Row="4" HorizontalAlignment="Left" Margin="1,6.368,0,21.3" Name="txtAppearance" Width="120" />
```

```
<TextBox Grid.Column="1" Grid.Row="5" Margin="2,8,14,53" Name="txtSummary"
Grid.RowSpan="3" Background="Linen" ScrollViewer.VerticalScrollBarVisibility="Auto"
ScrollViewer.HorizontalScrollBarVisibility="Auto" ScrollViewer.CanContentScroll="True"
TextWrapping="WrapWithOverflow" />
```

- 12. Add a <Image /> control after the <TextBox /> control XAML code, and add the values given in table 19.6 to the <Image> control.**

Image Properties	
Name	imgCountry
Grid.Column	1
Grid.Row	1
Grid.RowSpan	4
Margin	137,1,13,8
Stretch	Fill

Table 19.6: Image Properties

- 13. Add a <Border /> control after the <Image> XAML code.**

```
<Border BorderBrush="BurlyWood" BorderThickness="2"
ClipToBounds="True" Grid.ColumnSpan="2" Margin="0,24,-11,0" Height="3"
VerticalAlignment="Top" Grid.Row="7" Name="br2" />
```

- 14. Add a <Button/> control after the second <Border>, and then, add the values given in table 19.7 to the <Button> element.**

Property	Value
Name	btnLoad
Content	Load
Grid.Column	1
Grid.Row	7
HorizontalAlignment	Left
Width	75
Margin	5, 34, 0, 9

Table 19.7: Button Properties

15. Add two more `<Button />` controls after the first `<Button>` with the names as `btnUpdate` and `btnExit`.

```
<Button Grid.Column="1" Grid.Row="7" Margin="94, 34, 0, 9" Name="btnUpdate"
HorizontalAlignment="Left" Width="75" Content="Update" />
```

```
<Button Grid.Column="1" Grid.Row="7" Margin="178, 34, 0, 9" Name="btnExit"
HorizontalAlignment="Left" Width="84" Content="Exit" />
```

### Defining and Applying Styles to Label and Button Controls

You can apply a style to an individual control or a group of controls. To define a named style, perform the following steps:

1. In Teams.xaml, before the `<Grid>` element add the following code.

```
<Window.Resources>
    <Style x:Key="CommonStyle">
        <Setter Property="Control.FontSize" Value="12" />
        <Setter Property="Control.FontWeight" Value="Bold" />
        <Setter Property="Control.Foreground" Value="Brown" />
    </Style>
</Window.Resources>
```

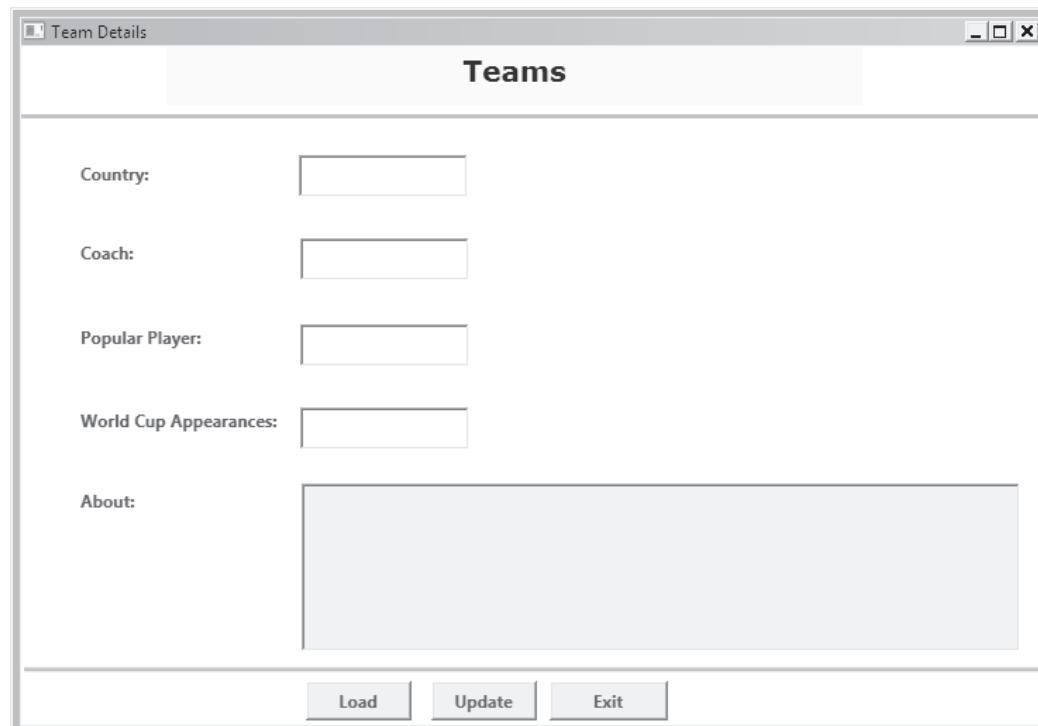
Here, a general style is defined with the name as `CommonStyle`. The `Control` prefix is used before the properties so that it can be used with any control. If this style is applied, the control's font size, font weight, and foreground color will be 12, Bold, and Brown respectively.

- 2. To apply the named style, CommonStyle to Label and Button controls, add the code, Style="{StaticResource CommonStyle}" to the required Label and Button controls. The following code applies the style to Country Label and Load Button.**

```
<Label Margin="35,1,69,21" Name="lblCountry" Grid.Row="1" Height="30" Content="Country: " Style="{StaticResource CommonStyle}"/>

<Button Grid.Column="1" Grid.Row="7" HorizontalAlignment="Left" Name="btnLoad" Width="75" Margin="5,34,0,9" Content="Load" Style="{StaticResource CommonStyle}"/>
```

Figure 19.4 shows the user interface after applying the named style, CommonStyle, to specific labels and buttons.



**Figure 19.4: User Interface after applying the style**

The style, CommonStyle, is applied to Coach, Popular Player, and World Cup Appearances. The same style is also applied to About Label controls, and Update and Exit Button controls.

#### Defining Properties for the Columns

You need to define properties for the columns in the Teams table. To define properties, perform the following steps:

1. Add a new class file named TeamInformation.cs to the project. Implement the INotifyPropertyChanged interface in the TeamInformation class.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ComponentModel;
namespace WorldCupFootball
{
    class TeamInformation : INotifyPropertyChanged
    {
        }
}
```

The INotifyPropertyChanged interface implemented in this class generates an event every time the value in the object property changes.

2. Define a property named Country in TeamInformation.cs file.

```
private string _Country;
public string Country
{
    get
    {
        return _Country;
    }
    set
    {
        _Country = value;
        OnPropertyChanged("Country");
    }
}
```

The class raises the `PropertyChanged` event when the value in the `Country` property is changed.

3. **Similarly, define properties named `Coach`, `PopularPlayer`, `WorldCupMatches`, `Summary`, and `Photo`.**

```
private string _Coach;
public string Coach
{
    get
    {
        return _Coach;
    }

    set
    {
        _Coach = value;
        OnPropertyChanged("Coach");
    }
}

private string _PopularPlayer;
public string PopularPlayer
{
    get
    {
        return _PopularPlayer;
    }

    set
    {
        _PopularPlayer = value;
        OnPropertyChanged("PopularPlayer");
    }
}
```

```
}
```

```
private int _WorldCupMatches;
```

```
public int WorldCupMatches
```

```
{
```

```
    get
```

```
    {
```

```
        return _WorldCupMatches;
```

```
    }
```

```
    set
```

```
    {
```

```
        _WorldCupMatches = value;
```

```
        OnPropertyChanged("WorldCupMatches");
```

```
    }
```

```
}
```

```
private string _Summary;
```

```
public string Summary
```

```
{
```

```
    get
```

```
    {
```

```
        return _Summary;
```

```
    }
```

```
    set
```

```
    {
```

```
        _Summary = value;
```

```
        OnPropertyChanged("Summary");
```

```
    }
```

```
}
```

```
private byte[] _Photo;
```

```
public byte[] Photo
```

```
{
```

```
    get
```

```
    {
```

```
        return _Photo;
```

```
    }
```

```
    set
```

```
    {
```

```
        _Photo = value;
```

```
        OnPropertyChanged("Photo");
```

```
    }
```

```
}
```

The `Photo` property is returning a `byte[]` array because it retrieves and stores the team photo.

4. To implement the `INotifyPropertyChanged` interface, right-click `INotifyPropertyChanged` from the class definition `TeamInformation`, and select **Implement Interface** → **Implement Interface**. This generates the `PropertyChanged` event.

```
#region INotifyPropertyChanged Members
```

```
public event PropertyChangedEventHandler PropertyChanged;
```

```
#endregion
```

The `PropertyChanged` event is generated. Next step is to generate the method for `PropertyChanged` event. To generate the method stub for `PropertyChanged`, perform the following steps:

- 5. Right-click any `OnPropertyChanged()` statement and select Generate Method Stub. The system automatically creates the `OnPropertyChanged()` method.**

```
private void OnPropertyChanged(string p)
{
    throw new NotImplementedException();
}
```

- 6. Modify the `OnPropertyChanged` method as shown in the code to include validation.**

```
private void OnPropertyChanged(string propertyName)
{
    if (this.PropertyChanged != null)
        PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
}
```

In the class, all the properties are defined and the `OnPropertyChanged()` statement is included with all the properties. Whenever the value in the property changes, it calls the `OnPropertyChanged()` method and it passes the current object along with the changed property name.

### Connecting to database

Next task is to create a method that connects to the database and retrieve data.

- 1. Define a method named `Load()` and write the statements to connect to the `WorldCupFootball` database.**

```
public void Load()
{
    SqlConnection objConnection = new SqlConnection("SERVER = MY SERVER\\
SQLEXPRESS; DATABASE=WorldCupFootball; UID=sa; pwd=sa");

    objConnection.Open();
    string str = "SELECT * FROM Teams WHERE Country = @Country";

    SqlCommand objCommand = new SqlCommand(str, objConnection);
    objCommand.CommandType = CommandType.Text;
    objCommand.Parameters.Add(new SqlParameter("@Country", _Country));
    SqlDataReader objReader = objCommand.ExecuteReader();
```

```

if (objReader.Read())
{
    _Coach = objReader["Coach"].ToString();
    _PopularPlayer = objReader["PopularPlayer"].ToString();
    _WorldCupMatches = (int)objReader["WorldCupMatches"];
    _Summary = objReader["Summary"].ToString();
    _Photo = (byte[])objReader["Photo"];

    OnPropertyChanged("Coach");
    OnPropertyChanged("PopularPlayer");
    OnPropertyChanged("WorldCupMatches");
    OnPropertyChanged("Summary");
    OnPropertyChanged("Photo");

    objCommand.Dispose();
    objConnection.Close();
    objConnection.Dispose();
}

}

```

The database connection is established by specifying the server and user details. Then, a `SELECT` command is issued to get all records from the `Teams` table based on the `Country` parameter. The records are stored in the `objReader` object. Then, the records are retrieved one by one and stored in the corresponding property elements. Next, the `OnPropertyChanged()` method of each property is called. Finally, all the resources related to the database are released.

The next step is to use WPF data binding syntax. You can use the `Text` property of text boxes and `Source` property of Image control for data binding.

## 2. Add the Binding statements to text boxes Text property and image's Source property.

```
<TextBox Grid.Column="1" Grid.Row="1" HorizontalAlignment="Left" Margin="0,3.163,0,23.163" Name="txtCountry" Width="120" Height="30" Text="{Binding Path=Country}" />
```

```

<TextBox Grid.Column="1" Grid.Row="2" Margin="1,3,0,23.734"
Name="txtCoach" HorizontalAlignment="Left" Width="120" Height="30"
Text="{Binding Path=Coach}"/>

<TextBox Grid.Column="1" Grid.Row="3" HorizontalAlignment="Left" Margin="1,5.368,0,22.3"
Name="txtPlayer" Width="120" Text="{Binding Path=PopularPlayer}"/>

<TextBox Grid.Column="1" Grid.Row="4" HorizontalAlignment="Left" Margin="1,6.368,0,21.3"
Name="txtAppearance" Width="120" Text="{Binding Path=WorldCupMatches}"/>

<TextBox Grid.Column="1" Grid.Row="5" Margin="2,3,14,58" Name="txtSummary"
Grid.RowSpan="3" Background="Linen" ScrollViewer.VerticalScrollBarVisibility="Auto"
ScrollViewer.HorizontalScrollBarVisibility="Auto" ScrollViewer.CanContentScroll="True" TextWrapping="WrapWithOverflow"
Text="{Binding Path=Summary}"/>

```

The {Binding Path=...} statements in the XAML code retrieve values from the bound object or resource and place them in the target controls.

### 3. Attach the Loaded event to the <Window> element.

```

<Window x:Class="WorldCupFootball.Team"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Team Details" Height="517" Width="741" Name="TeamDetails"
        Topmost="True" BorderBrush="Chocolate" Loaded="TeamDetails_Loaded">

```

The Loaded event for the <Window> element is defined.

### 4. To generate the Loaded event handler, right-click the statement Loaded = 'TeamDetails\_Loaded' and select Navigate to Event Handler. The control will move to the Loaded() event handler of Teams.xaml.cs code-behind file.

```

private void TeamDetails_Loaded(object sender, RoutedEventArgs e)
{
}

```

- 5. In Teams.xaml.cs, instantiate an object of TeamInformation class.**

```
public partial class Team : Window
{
    TeamInformation objTeamInfo = new TeamInformation();
    .....
    .....
}
```

- 6. Assign the TeamInformation object, objTeamInfo to the Grid's DataContext property in TeamDetails\_Loaded event handler.**

```
private void TeamDetails_Loaded(object sender, RoutedEventArgs e)
{
    grTeam.DataContext = objTeamInfo;
}
```

The statement defines a data context for the Grid, the UI element that encloses all other UI controls. Once the data context is set in the form loaded event, the {Binding Path=...} statements in XAML retrieve the value from the bound object or resource and place them in the target controls.

- 7. In the Design Window of Teams.xaml, double-click the Load button to generate the btnLoad\_Click event handler.**

```
private void btnLoad_Click(object sender, RoutedEventArgs e)
{
}
```

- 8. Write the code to call the Load() method of TeamInformation class in the Click event of btnLoad() button.**

```
try
{
    objTeamInfo.Load();
}
```

```
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Error");
}
```

The `Load()` method of the `TeamInformation` class is called. If it fails, it throws an error message.

**9. Build and execute the application. The output is displayed as shown in figure 19.5.**

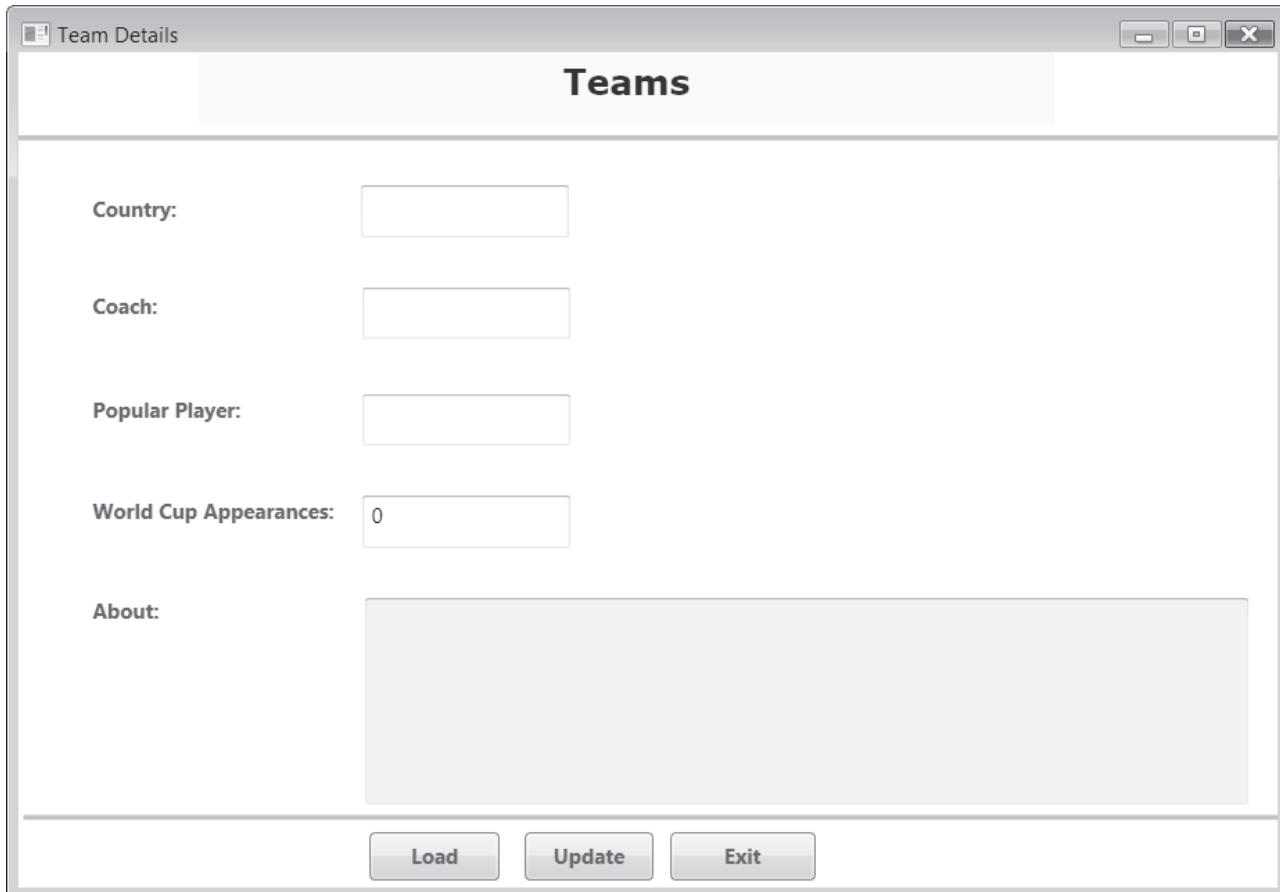


Figure 19.5: TeamDetails

10. Type the text Brazil in Country text box and click Load. The output will be displayed as shown in figure 19.6.



Figure 19.6: Output of TeamDetails

### Modifying the records

1. In TeamInformation.cs, add one more method named Update() to modify the records in the database.

```
public void Update()
{
    SqlConnection objConnection = new SqlConnection("SERVER = MYSERVER\\
SQLEXPRESS; DATABASE=WorldCupFootball; UID=sa; pwd=sa");

    objConnection.Open();
    string str = "";
    str += "UPDATE Teams SET Coach = @Coach ,";
    str += "PopularPlayer = @PopularPlayer,";
```

```

str += "WorldCupMatches = @WorldCupMatches,";

str += "Summary = @Summary";
str += " WHERE Country = @Country";
SqlCommand objCommand = new SqlCommand(str, objConnection);

objCommand.CommandType = CommandType.Text;

objCommand.Parameters.Add(new SqlParameter("@Country", _Country));
objCommand.Parameters.Add(new SqlParameter("@Coach", _Coach));
objCommand.Parameters.Add(new SqlParameter("@PopularPlayer", _Popular-
Player));
objCommand.Parameters.Add(new SqlParameter("@WorldCupMatches", _World-
CupMatches));
objCommand.Parameters.Add(new SqlParameter("@Summary", _Summary));

objCommand.ExecuteNonQuery();

objCommand.Dispose();
objConnection.Close();
objConnection.Dispose();
}

```

The modified text in the form is updated in the table, Teams by using the `ExecuteNonQuery()` method of `SqlCommand` class.

- 2. In Teams.xaml Design Window, double-click the Update button to generate the btnUpdate\_Click event handler.**

```

private void btnUpdate_Click(object sender, RoutedEventArgs e)
{
}

```

- 3. Add code to call the `Update()` method of TeamInformation.cs in the Click event handler of btnUpdate button.**

```
try
{
    objTeamInfo.Update();
    MessageBox.Show("Record Successfully updated !!!", "Update");
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message, "Error");
}
```

**4. Generate the Click event handler for Exit button in Teams.xaml file and add code to exit the application.**

```
private void btnExit_Click(object sender, RoutedEventArgs e)
{
    MessageBoxButtonResult objResult = MessageBox.Show("Exit the application?", "Exit", MessageBoxButton.YesNo, MessageBoxIcon.Stop);

    if (objResult == MessageBoxButtonResult.Yes)
        this.Close();
}
```

On clicking Exit, the user is prompted with a dialog box asking whether to exit the application. If Yes is selected, the application is terminated.

**5. Build and execute the application. Type the text Brazil in Country text box and click Load. The team details are displayed.**

6. Change the Popular Player name to Robinho and click Update. The message Record Successfully updated. is displayed as shown in figure 19.7.



Figure 19.7: Updating a Record

7. Click OK to close the dialog box.  
8. To exit the application, click Exit. The Exit dialog box is displayed as shown in figure 19.8.

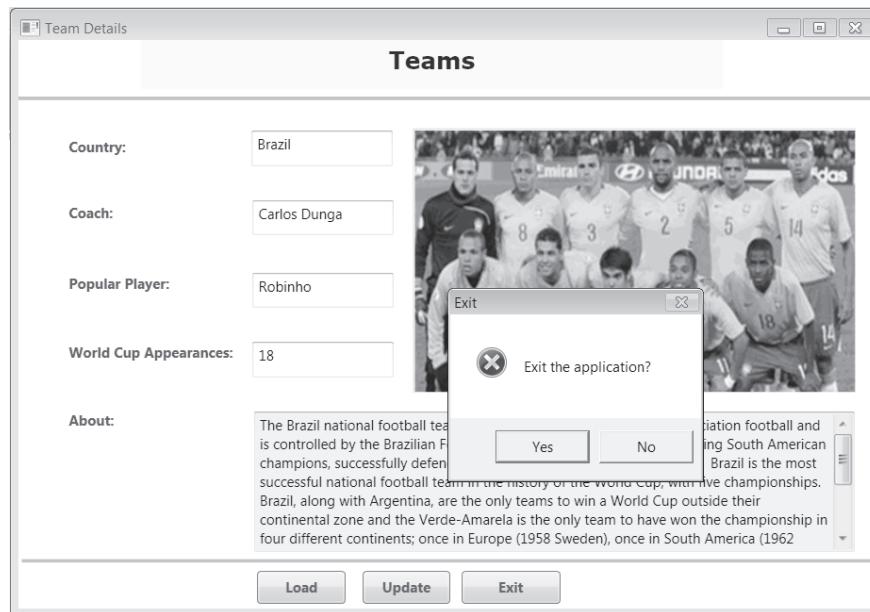


Figure 19.8: Exit Dialog Box

9. Click Yes to exit the application.

## Part II – 60 Minutes

1. Modify the application given in Part I to do the following:
  - Modify the Country text box to a combo box control so that the user can select a country from the list
  - Modify the `TeamInformation.cs` file to create a common method for loading the database for both Load and Update methods
  - Use the object transforms to change the look and feel of the image

**Hints:**

- Create an array for storing all the countries and then, populate the countries in the Country combo box
- Create a method named `DBConnection()` to specify the database connection. Then, call this method whenever you need the database
- Use rotation, scale, and skew transforms



## Try it Yourself

1. Create a WPF user control to validate the user credentials such as user id and password. The user details should be validated against the values stored in the database. Host the user control in a Windows Forms application. If the user details provided are correct, the application should display appropriate message.
2. Create a WPF application to display all the details of a student stored in SQL Server database. Create separate tabs to display the student details. The first tab will show icons such as Personal, News, Results, and Exam. When the Personal icon is clicked the personal information of such as name, age, family details, academic qualification, photo, and so on should be displayed in second tab. If the News icon is clicked, the university news will be displayed on third tab. When the Results tab is clicked, the academic performance of the student such as marks, grade, and so on will be displayed in the next tab. The university exam related details will get displayed in the last tab when the user clicks the Exam icon.

**GROWTH**  
**RESEARCH**  
**OBSERVATION**  
**UPDATES**  
**PARTICIPATION**



**[www.onlinevarsity.com](http://www.onlinevarsity.com)**

# Module - 20

## LINQ and ADO.NET Entity Framework

Welcome to the Module, **LINQ and ADO.NET Entity Framework**.

This session will explore the LINQ, ADO.NET Entity Framework, Entity Data Model, and data retrieval using the EDM.

In this Module, you will learn about:

- Define and describe Language Integrated Query (LINQ)
- List and describe the commonly used query operators
- Describe the use of LINQ to query different data sources
- Explain the ADO.NET Entity Framework
- Describe Entity Data Model
- Describe creation of a simple Entity Data Model
- Describe creation of a Windows Forms Data Source using EDM
- Describe data retrieval using the EDM

## 20.1 Introduction

Microsoft Visual Studio 2008 introduces many new features and enhancements. One of the most important and beneficial new feature is LINQ. LINQ is a set of technologies that simplifies data access present in various formats on different data sources. LINQ provides a consistent model to work with such data.

Another key feature is the ADO.NET Entity Framework which is integrated into Visual Studio 2008. Using ADO.NET Entity Framework, you can now build data access applications by interacting with a conceptual application model, instead of a relational database.

This session describes a brief introduction to LINQ and the ADO.NET Entity Framework.

## 20.2 LINQ

Applications often need to store, retrieve, and manipulate data in some form or another. In Windows-based .NET applications, such as Windows Forms applications, a technology called ADO.NET is extensively used to enable data handling. Data, however, is not always in the same format. While some applications store and manage XML data, some others may work with data in relational data sources. There are also applications that may use data from in-memory collections.

Retrieving and maintaining data requires creation of queries and statements. For each kind of data source, a developer may have to use a different approach. This is cumbersome and time consuming.

Another issue is that the Visual Studio environment has no provision to check for any syntax errors present in SQL queries written in C#. If any such errors exist or the query is malformed, it would be detected only at runtime.

To overcome all these issues, Microsoft introduced a new technology, Language-Integrated Query, LINQ.

LINQ is a set of features that adds query capabilities to .NET languages such as C#. LINQ enables you to query data from diverse data sources in an easy manner. The only condition is that these data sources must be LINQ-compatible, which means they must be supported by LINQ. LINQ can be used with SQL, XML files, objects (such as C# arrays and collections), and ADO.NET DataSets.

Figure 20.1 shows an overview of LINQ.

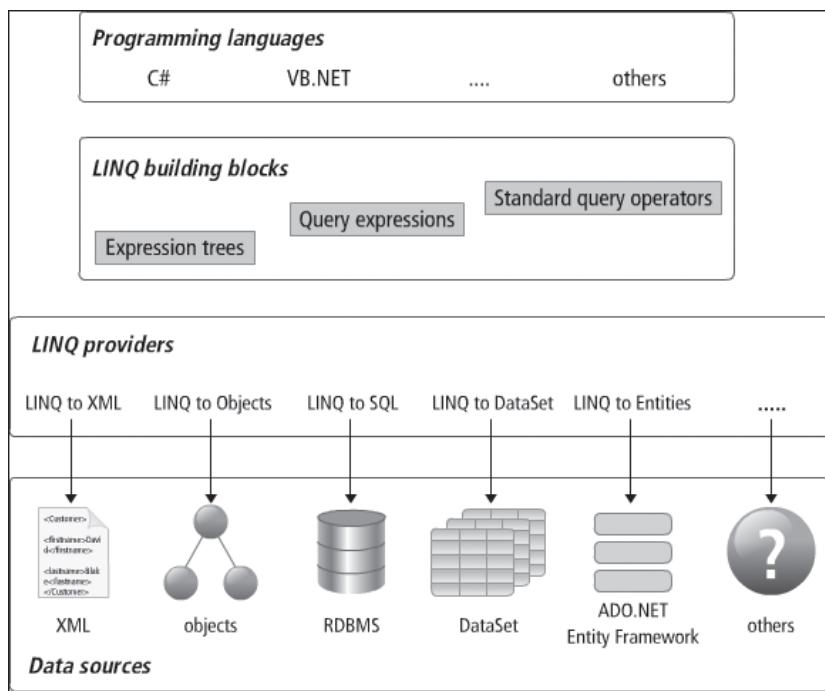


Figure 20.1: LINQ

Hence, LINQ gives you a consistent means to access data.

One of the key advantages of LINQ is that the syntax used to write queries is the same, regardless of which data source is used.

Through LINQ, developers can now work with queries as part of the C# language. Developers can create and use query expressions, which are used to query and transform data from a data source supported by LINQ. A query expression is a query that is written in query syntax using clauses such as `from`, `select`, and so forth.

Query expressions are written using a declarative query syntax introduced in C# 3.0.

A simple example of a query expression is shown. Consider that you have a `ListBox` named `1stResult`. Given an array of integers, you need to populate the `ListBox` with only those elements from the array that are greater than 50. The following code shows how to use a LINQ query expression to achieve this.

#### Code Snippet:

```
...
int[] points = { 15, 105, 89, 33, 60, 124, 68, 23, 58, 91 };
// Query expression
IQueryable<int> result =
from point in points
```

```

    where point > 50
        orderby point
        select point;
foreach (int i in result)
{
    lstResult.Items.Add(i);
}

```

In this code the query keywords such as `from`, `select`, and so forth are used.

A query expression must always start with a `from` clause. The `from` clause specifies the data source on which the query will execute and a range variable representing each element in the source sequence. Here, in this code, `points` is the data source and `point` is the range variable. The type of the range variable is inferred by the compiler based on the data source.

For example, in the code, the data source is an `int` array. Hence, the data type of `point` is inferred to be an `int`. The range variable is similar to an iteration variable in a `foreach` statement except that a range variable does not actually store any data.

The `where` clause is used to filter the data based on some criteria and the `select` clause returns the data that is being queried.

The output of the code is shown in figure 20.2. The `ListBox` contains only the points that are greater than 50.

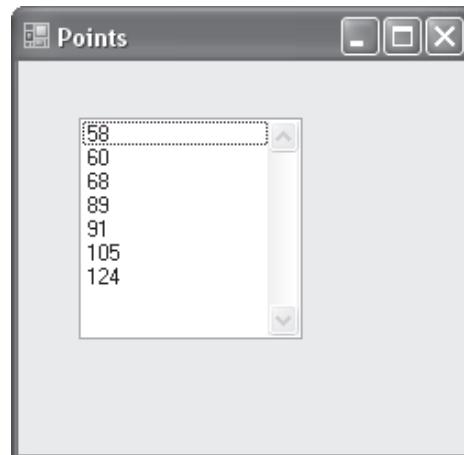


Figure 20.2: ListBox Output

The CLR in .NET does not understand LINQ query syntax; therefore, query expressions need to be translated to method calls at compile time. This will enable the CLR to understand and process it. These methods are called standard query operators. They have names such as `Select`, `Where`, `GroupBy`, `Sum`, `Average`, and so on. They can be called directly as methods instead of query syntax that was shown in the code.

## 20.3 Commonly used Standard Query Operators

Standard query operators represent the operations available in LINQ and used for tasks such as filtering, counting, aggregating, sorting, and so forth. They provide querying capability over the collection on which they operate. Standard query operators are the static methods of the static classes, System.Linq.Enumerable and System.Linq.Queryable.

Table 20.1 lists the commonly used standard query operators that are supported by C#.

Category	Operator	Description
Restriction Operators	Where	Filters values based on a given criteria
Projection Operators	Select	Presents the values based on a transform function
Join Operators	Join	Performs an inner join of two collections based on matching columns
Element Operators	First	Returns the first element of a collection
	Last	Returns the last element of a collection
	Single	Returns a single element of a collection
Concatenation Operators	Concat	Concatenates two collections to form one collection
Ordering Operators	OrderBy	Sorts or arranges a collection by one or more keys
Grouping Operators	GroupBy	Groups the elements of a collection
Aggregate Operators	Aggregate	Applies a function over a collection
	Average	Calculates the average of a collection of numeric values
	Count	Counts the number of elements in a collection
	Sum	Computes the sum of a collection of numeric values

Table 20.1: Commonly Used Standard Query Operators

The following code shows how to use the Count operator to count the items in a string array.

### Code Snippet:

```
// Declare and initialize an array of strings
string [] books = {"Perl in 24 hours", "Java Study Guide", "C#", "LINQ", "Exploring
ASP.NET", "Beginning JavaScript"};
MessageBox.Show(books.Count().ToString(), "Number of Books",
    MessageBoxButtons.OK, MessageBoxIcon.Information);
```

When you use books.Count(), you are making a method call to the standard query operator Count, which is a static method.

The following code shows how to use the `OrderBy` operator with the lambda operator to sort a list of books and then, add it to a `ListBox` named `lstBooks`.

**Code Snippet:**

```
// Declare and initialize an array of strings
string [ ] books = {"Perl in 24 hours", "Java Study Guide", "C#", "LINQ", "Exploring
ASP.NET", "Beginning JavaScript"};
foreach (string book in books.OrderBy(name => name))
{
    lstBooks.Items.Add(book);
}
```

The code sorts the list of book titles in the string array alphabetically and then, adds them into the `ListBox` one by one. It makes use of the standard query operator `OrderBy` along with the lambda operator. The following code shows how to use the `Sum` operator.

**Code Snippet:**

```
List<float> prices = new List<float> { 45.50F, 1.25F, 58.0F, 68.50F };
float sum = prices.Sum();
MessageBox.Show(sum.ToString(), "Total Price",
    MessageBoxButtons.OK, MessageBoxIcon.Information);
```

## 20.4 Querying different data sources using LINQ

One of the key features of LINQ is that it enables you to query data from different types of data sources. LINQ can be categorized into various flavors as shown in figure 20.3.

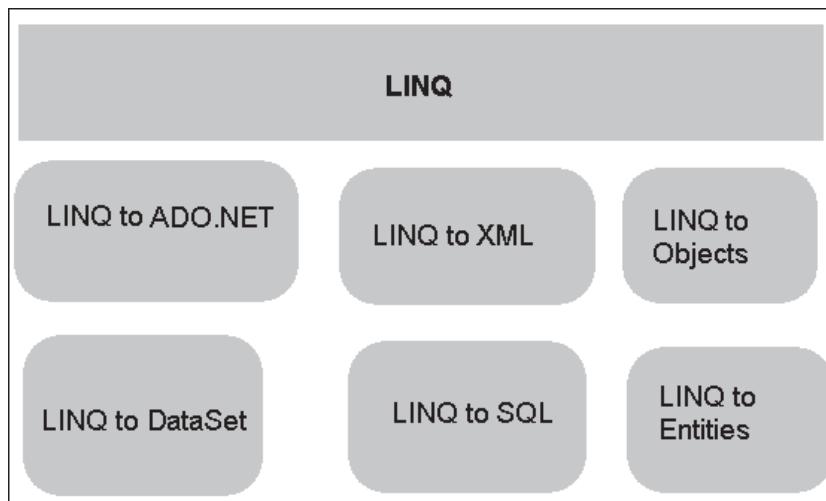


Figure 20.3: Different flavors of LINQ

### 20.4.1 LINQ to SQL

LINQ to SQL allows you to query and manipulate data from SQL Server databases in a .NET language of your choice.

LINQ to SQL requires a reference to the `System.Data.Linq` namespace. You must also map your data object model to your database using the `Table` and `Column` attributes to point to the respective database tables and columns.

The LINQ to SQL object that contains the information to connect to the database is the `DataContext` object. In LINQ to SQL, the `DataContext` class is the means by which objects are retrieved from the database and changes are sent back. The `DataContext` converts your requests for objects into SQL queries against the database and then, converts the results into objects.

Consider that you have created a SQL Server database named `MovieDB`. The database contains a table `Movies` with structure as shown in figure 20.4.

	Column Name	Data Type	Allow Nulls
1	MovieID	int	<input type="checkbox"/>
	MovieName	nchar(40)	<input checked="" type="checkbox"/>
	Category	nchar(20)	<input checked="" type="checkbox"/>

Figure 20.4: Movies Table

Figure 20.5 shows the records present in the table `Movies`.

	MovieID	MovieName	Category
	109	Three Amigos	Comedy
	111	Casablanca	Romance
	222	The Italian Job	Thriller
	333	The Patriot	Epic
	444	The Green Mile	Drama
	555	The Hurt Locker	Drama
	896	The Blind Side	Drama
	888	Vertigo	Drama

Figure 20.5: Data in the Movies Table

The step by step procedure to query the table using LINQ to SQL is as follows:

1. Create a new Windows Forms application named `MovieApplication`. Set the `Name` and `Text` properties of the default form to `frmLinqToSql` and `LINQ to SQL` respectively.
2. Select **Project→Add Reference** and add a reference to `System.Data.Linq.dll`.
3. Add `using` statements for `System.Data.Linq` and `System.Data.Linq.Mapping` to the form. These namespaces are required to implement the LINQ to SQL functionality.
4. Add a `ComboBox` to the form and set its `Name` property to `cboCategory`.
5. Add a `DataGridView` to the form and set its `Name` property to `dgvwMovies`.

6. Add a **Button** to the form and set its **Name** and **Text** properties to **btnSave** and **Save** respectively.
7. Add the following code of to the form class:

**Code Snippet:**

```
[Table(Name = "Movies")]
public class Movies
{
    private int _MovieID;
    [Column(IsPrimaryKey = true, Storage = "_MovieID")]
    public int MovieID
    {
        get
        {
            return this._MovieID;
        }
        set
        {
            this._MovieID = value;
        }
    }
    private string _MovieName;
    [Column(Storage = "_MovieName")]
    public string MovieName
    {
        get
        {
            return this._MovieName;
        }
        set
        {
            this._MovieName = value;
        }
    }
}
```

```

    }

}

private string _Category;
[Column(Storage = "_Category")]
public string Category
{
    get
    {
        return this._Category;
    }
    set
    {
        this._Category = value;
    }
}
}

```

The **Table** attribute is used to associate the **Movies** class with your **Movies** table. Within this class, there are several property definitions with **Column** attributes. This attribute is used to map a property in a class to a column in the table. The data type of both the property and column must match.

- Add the code at the class level of **frmLinqToSql** to create the **DataContext** and a table against which you want to query.

**Code Snippet:**

```

DataContext dcontext = new DataContext ("Data Source = pop \\ express; Initial
Catalog=MovieDB; uid=sa; pwd=release; Integrated Security=True");
Table<Movies> Movies;

```

- Create an event handler for **Load** event of the form and add the code to query the **Movies** table.

**Code Snippet:**

```

private void frmLinqToSql_Load(object sender, EventArgs e)
{
    // Get a typed table to run queries
}

```

```
Movies = dcontext.GetTable<Movies>();  
  
// Retrieve a list of Categories using a LINQ to SQL query  
var query = from movie in Movies  
            select movie.Category;  
  
// Populate the categories into the combo box  
foreach (var category in query.Distinct())  
{  
    cboCategory.Items.Add(category.ToString());  
}  
  
// Set the default selection  
cboCategory.SelectedIndex = 0;  
}
```

Consider the following expression:

```
from movie in Movies  
select movie.Category;
```

Here, **movie** is defined as a range variable and **Movies** is the data source, in this case, the table.

The code retrieves all the categories from the Movies table and populates them into the combo box.

The sequence in which this snippet executes is described now. When the LINQ to SQL query expression is constructed, it is not immediately executed but saved in memory for later execution. This is called deferred execution. Almost all the flavors of LINQ are based on deferred execution. Only when the variable, **query**, is encountered in the foreach loop, the query expression is converted to SQL format and executed on the SQL Server.

10. Create an event handler for the **SelectedIndexChanged** event of **cboCategory** and add the following code:

**Code Snippet:**

```
private void cboCategory_SelectedIndexChanged(object sender, EventArgs e)  
{
```

```
string selected = cboCategory.SelectedItem.ToString().Trim();  
  
// Retrieve data using a LINQ to SQL query  
var movies = from movie in Movies  
             where movie.category == selected  
             select movie;  
  
dgvwMovies.DataSource = movies;  
}
```

This code is executed whenever a selection is made in the combo box. The code retrieves all the movies belonging to the category selected in the combo box and then, assigns this to the DataGridView.

Figure 20.6 shows the output.

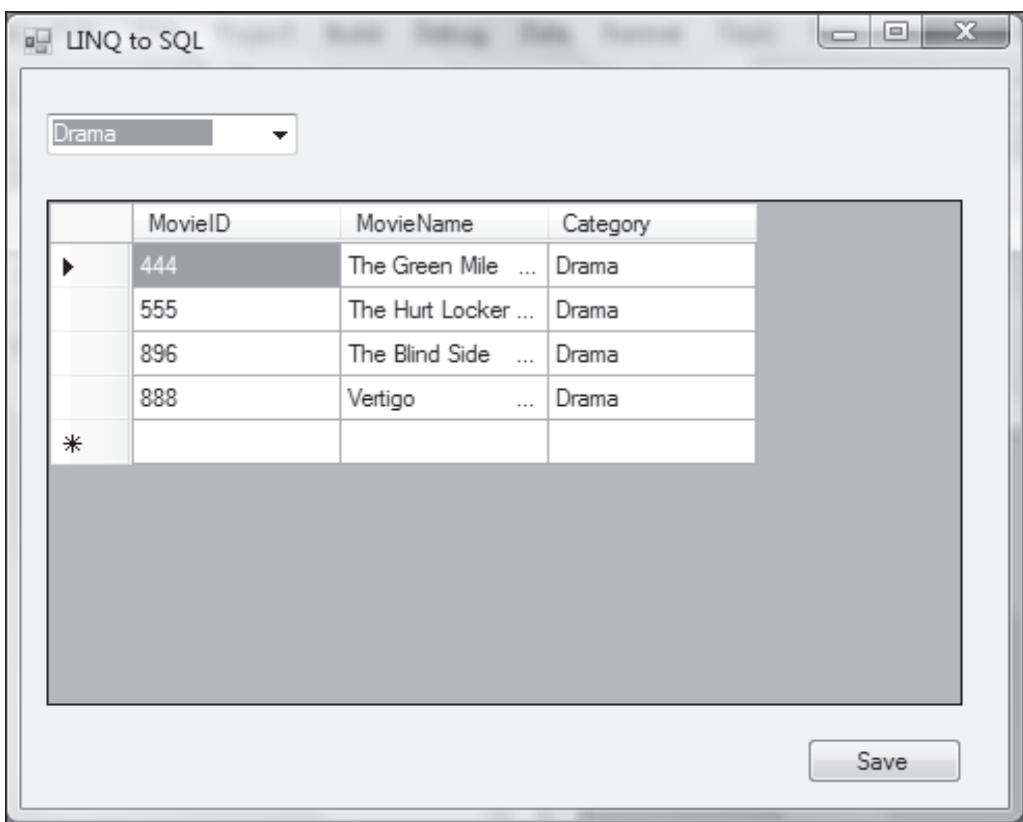


Figure 20.6: Output

11. Add the following code to the **Click** event of the Save button.

**Code Snippet:**

```
private void btnSave_Click(object sender, EventArgs e)
{
    try
    {
        dcontext.SubmitChanges();
        MessageBox.Show("Record Added");
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

The `SubmitChanges()` method of `DataContext` class enables you to submit changes from your application to the data source.

#### 20.4.2 LINQ to XML

LINQ to XML is an approach to create, store, and retrieve XML data in a .NET language of your choice. LINQ to XML simplifies working with XML data without requiring use of additional XML technologies such as XPath. However, LINQ to XML is not a replacement for any of XML class libraries or technologies.

Three important classes of LINQ to XML are, `XDocument`, `XElement`, and `XAttribute`. These classes are defined in the `System.Xml.Linq` namespace.

LINQ to XML makes it really easy to construct or retrieve elements and attributes from the XML file.

Consider an example.

Create a new Windows Forms application named **CustomerData**.

Add an XML file to the project and name it `Customers.xml`. Add the following content to it:

```
<?xml version="1.0" encoding="utf-8" ?>
<customers>
    <customer>
        <custId>120</custId>
        <firstName>Cathy</firstName>
```

```
<lastName>Baker</lastName>
</customer>
<customer>
    <custId>121</custId>
    <firstName>Jake</firstName>
    <lastName>Perry</lastName>
</customer>
<customer>
    <custId>101</custId>
    <firstName>Kim</firstName>
    <lastName>Faukner</lastName>
</customer>
<customer>
    <custId>111</custId>
    <firstName>Robin</firstName>
    <lastName>Stuart</lastName>
</customer>
</customers>
```

3. Save and close the XML file.
4. Right-click the Customers.xml in the Solution Explorer and select Properties. Set the **Copy To Output Directory** property to **Copy if newer**.
5. Set the **Name** and **Text** properties for the form to **frmCustomers** and **Customer Data** respectively.
6. Add a **ListBox** to the form and set the **Name** property to **lstCustomers**.
7. Add a **Button** to the form and set the **Name** and **Text** properties to **btnLoad** and **Load Customers** respectively.
8. Add a **Button** to the form and set the **Name** and **Text** properties to **btnGet** and **Get Customer Info** respectively.
9. Add two **TextBox** controls to the form and set their **Name** properties to **txtFirstname** and **txtLastname** respectively.
10. Add a **Label** at the top of the text box, **txtFirstname**, and set its **Name** and **Text** properties to **lblFirstname** and **First Name** respectively.

11. Add a **Label** at the top of the text box, `txtLastname`, and set its **Name** and **Text** properties to `lblLastname` and `Last Name` respectively.
12. Arrange the controls so that the form looks like figure 20.7.

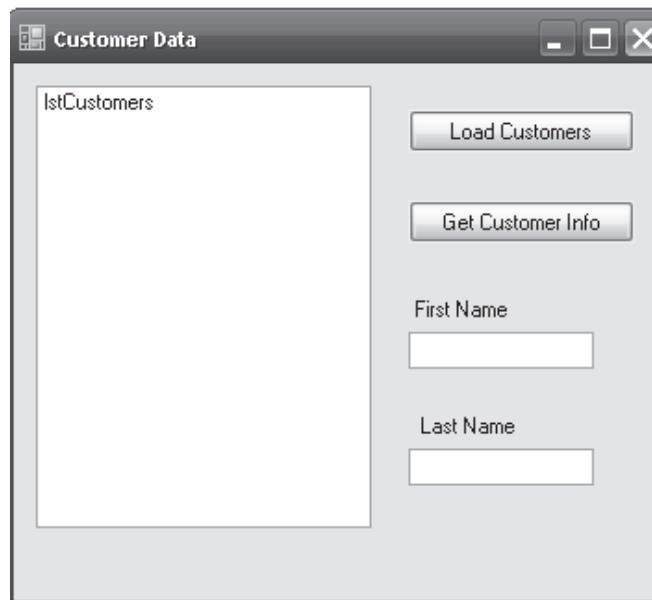


Figure 20.7: Customer Data Form

13. Double-click the **Load Customers** button. This will generate an event handler and automatically switch to code view.
14. Add a **using** statement for the `System.Xml.Linq` namespace in the **using** section of the code.
15. Add the following code in the event handler to load the customers list from the XML file:

**Code Snippet:**

```
private void btnLoad_Click(object sender, EventArgs e)
{
    // If already populated earlier, clear the list
    lstCustomers.Items.Clear();

    XElement root = XElement.Load("Customers.xml");
    IEnumerable<XElement> customer = from el in root.Descendants("customer")
        orderby
            el.Element("custId").Value
        select el;
```

```
{
    1stCustomers.Items.Add(el.Element("custId").Value);
}
}
```

The code retrieves all the customer ids, sorts them, and then, populates them into the `ListBox`.  
The `Descendants()` method of `XElement` class retrieves all the child elements.

16. Double-click the **Get Customer Info** button to generate an event handler.
17. Add the following code to load the customer details from the XML file:

**Code Snippet:**

```
private void btnGet_Click(object sender, EventArgs e)
{
    XElement root = XElement.Load("Customers.xml");
    IEnumerable<XElement> product =
        from el in root.Descendants("customer")
        where (el.Element("custId").Value ==
            1stCustomers.SelectedItem.ToString())
        select el;
    foreach (XElement el in product)
    {
        txtFirstname.Text = el.Element("firstName").Value;
        txtLastname.Text = el.Element("lastName").Value;
    }
}
```

The code retrieves all the customer ids, sorts them, and then, populates them into the `ListBox`.  
The `Descendants()` method of `XElement` class retrieves all the child elements.

16. Double-click the **Get Customer Info** button to generate an event handler.
17. Add the following code to load the customer details from the XML file:

**Code Snippet:**

```
private void btnGet_Click(object sender, EventArgs e)
{
}
```

```
XElement root = XElement.Load("Customers.xml");  
  
IEnumerable<XElement> product =  
    from el in root.Descendants("customer")  
    where (el.Element("custId").Value ==  
        lstCustomers.SelectedItem.ToString())  
    select el;  
  
foreach (XElement el in product)  
{  
    txtFirstname.Text = el.Element("firstName").Value;  
    txtLastname.Text = el.Element("lastName").Value;  
}  
}
```

The code searches for the customer details based on the customer id selected and populates the details into the text box.

18. Build and execute the application.
19. Click the **Load Customers** button. This populates all the customer ids in the list box. Select a customer id in the list. Click the **Get Customer Info** button. Verify that the details for the selected customer appear in the text boxes as shown in figure 20.8.

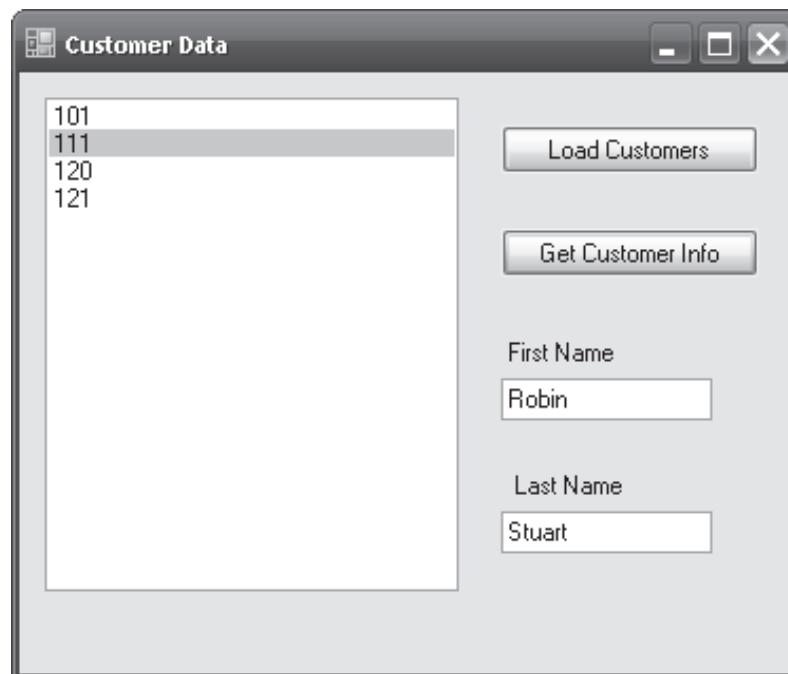


Figure 20.8: Output

### 20.4.3 LINQ to Objects

LINQ to Objects enables you to write queries over any enumerable collection, such as arrays, lists, and collections. Using LINQ to Objects, you can also query any collection that implements the interfaces, `IEnumerable`, `IQueryable`, or `IEnumerable(Of T)`.

In simplest terms, LINQ to Objects is a means of using LINQ to query perform in-memory data.

Before LINQ, when you had to work with collections of objects, you had to use for loops or foreach loops to iterate through a list or object collection to perform various tasks. With LINQ, you can make use of standard query operators which simplifies the query creation process. So if you have to query a collection and filter it based on some criteria, it becomes much easier with LINQ.

LINQ queries are better than `foreach` loops when querying in-memory collections and objects. This is because LINQ queries are more crisp and clear, especially when writing filters for multiple conditions. With just a minimum of application code, you can achieve grouping, filtering, and ordering capabilities on objects. They can also be migrated to other data sources with least modification.

The following code demonstrates how to use a basic LINQ to Objects query.

#### Code Snippet:

```
int[] votes = new int[] {100, 400, 250, 600, 350, 80, 35, 100};

int totalVotes = votes.Sum();

MessageBox.Show(totalVotes.ToString(), "Votes", MessageBoxButtons.OK,
    MessageBoxIcon.Information);
```

Here, the object is an array of `int` values. The standard query operator, `Sum`, is used here on the object to obtain the total number of votes.

Output: 1915

The following example also demonstrates a LINQ to Objects query.

#### Example:

```
public class Employee
{
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Location { get; set; }
    public int Salary { get; set; }
}

public partial class frmEmployee : Form
```

```
{  
    public frmEmployee()  
    {  
        InitializeComponent();  
    }  
  
    private void frmEmployee_Load(object sender, EventArgs e)  
    {  
        ArrayList arrList = new ArrayList();  
        arrList.Add(  
            new Employee  
            {  
                FirstName = "Lara",  
                LastName = "Baker",  
                Salary = 2000,  
                Location = "New Jersey"  
            } );  
        arrList.Add(  
            new Employee  
            {  
                FirstName = "Mark",  
                LastName = "Dell",  
                Salary = 3900,  
                Location = "New Jersey"  
            } );  
        arrList.Add(  
            new Employee  
            {  
                FirstName = "Steven",  
                LastName = "James",  
            } );  
    }  
}
```

```
Salary=2500,  
    Location="California"  
});  
  
var query = from Employee emp in arrList  
    where emp.Location == "New Jersey"  
    select emp;  
  
foreach (var employee in query)  
{  
    lstEmp.Items.Add(employee.FirstName + " " + employee.LastName);  
}  
}  
}
```

Here, an `ArrayList` collection is created based on a class `Employee`. Three objects of this class are added to the `ArrayList` collection. Then, using LINQ to Objects, a query is constructed to retrieve the data of each employee whose Location is New Jersey.

#### 20.4.4 LINQ to DataSet

LINQ to DataSet enables you to use the benefits of LINQ for ADO.NET DataSets without having to rewrite the whole data access layer. Earlier, in Windows Forms applications, if you wanted to achieve complex queries on a DataSet, you had to write custom queries.

LINQ to DataSet offers several advantages such as:

- It is easier and faster to perform queries on `DataSets`. `LINQ to DataSet` enables you to write queries from within C# applications, instead of using a separate query language. This makes it easier to query. The features such as short concise syntax, compile-time syntax checking, IntelliSense support, and so forth that also boost make it faster to write queries using `LINQ to DataSet`
  - `LINQ` capabilities such as standard query operators and others can be used with `DataSet` to construct complex queries

The LINQ to DataSet feature can be used with both untyped as well as typed DataSets, though it is more commonly used with typed DataSets.

LINQ support for `DataSet` is implemented entirely in `System.Data.Extensions.dll`.

In Visual Studio 2008, the new C# compiler supports LINQ, therefore, in such project types, Visual Studio makes use of an enhanced typed-DataSet builder that generates LINQ-compatible DataSets.

A DataSet must be populated with data before LINQ queries can be performed on it. When using LINQ to DataSet, the technique in which data is populated into the DataSet is not important. Hence, you can use any approach to load the data.

Consider the MovieDB scenario once again. Code Snippet demonstrates a commonly used approach, using the `Fill()` method, to populate the data into the DataSet.

**Code Snippet:**

```
MovieDataSet dataSet = new MovieDataSet();
MovieDataSetTableAdapters.MoviesTableAdapter mta = new
    MovieDataSetTableAdapters.MoviesTableAdapter();
mta.Fill(dataSet.Movies);
```

Once data has been loaded into the DataSet, you can start querying it. Creating and running queries over DataSet with LINQ is just like using LINQ against any other data source. When using LINQ queries over DataSet, you will be working with an enumeration of DataRow objects. This means that you can invoke the members of DataRow in query expressions.

Code Snippet demonstrates the use of LINQ to DataSet feature. Here, properties such as `MovieID`, `MovieTitle`, and `Category` are manually constructed for the new row that is built using `select new`.

**Code Snippet:**

```
var movie = from Movie in dataSet.Movies
    select new
    {
        MovieId = Movie.MovieID,
        MovieTitle = Movie.MovieName,
        Category = Movie.Category
    };
foreach (var row in movie)
{
    MessageBox.Show(row.MovieId.ToString(), "", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
    MessageBox.Show(row.MovieTitle.ToString(), "", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
```

```
MessageBox.Show(row.Category.ToString(), (), "", MessageBoxButtons.OK,
    MessageBoxIcon.Information);
}
```

Alternatively, you can use the code in Code Snippet 15 to achieve the same output. Here, you do not construct a new row using `select new` but instead retrieve the whole collection of rows into the variable `movie`. Then, you iterate through the rows using a `foreach` loop. Here, properties are not manually constructed as they were done in Code Snippet 14.

#### **Code Snippet 15:**

```
var movie = from Movie in dataSet.Movies
    select Movie;
foreach (var row in movie)
{
    MessageBox.Show(row[0].ToString(), (), "", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
    MessageBox.Show(row[1].ToString(), "", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
    MessageBox.Show(row[2].ToString(), "", MessageBoxButtons.OK,
        MessageBoxIcon.Information);
}
```

Code Snippet demonstrates the use of `where` standard query operator to retrieve all the movie names with category, Drama.

#### **Code Snippet:**

```
var names = from Movie in dataSet.Movies
    where Movie.Category.Trim().Equals("Drama")
    select Movie.MovieName;

foreach (string name in names)
{
    MessageBox.Show(name);
}
```

## 20.5 ADO.NET Entity Framework

The Microsoft ADO.NET Entity Framework was introduced with Visual Studio 2008 and ADO.NET 3.5.

ADO.NET Entity Framework is an Object-Relational Mapping (ORM) framework consisting of a data model and a set of design-time and run-time services. These allow developers to describe the application data and interact with it at a conceptual level of abstraction that is appropriate for business applications. An Object-Relational Mapping is a programming approach to convert data between relational databases and object-oriented programming languages.

ADO.NET Entity Framework enables .NET developers to run queries using LINQ and retrieve and manipulate data in the form of strongly typed objects. ADO.NET Entity Framework and its tools are part of Visual Studio 2008. They enable to create entity models from databases using wizards and drag-and-drop operations. ADO.NET Entity Framework can be used with various .NET application types including ASP.NET, WPF, and Windows Communication Foundation (WCF).

ADO.NET Entity Framework enables developers to write .NET applications against classes generated from the conceptual layer. The Entity Framework then takes care of all the behind-the-scenes tasks when data is extracted from the database and updated data is sent back.

In the ADO.NET Entity Framework, an entity is a concept in the domain of an application from which a data type is defined. For example, a Customer type often defines details such as customer code, contact name, company name, and address. Entity types are the .NET Framework classes representing entities. Entity types can have scalar, complex, and navigation properties. A Customer entity can be defined with properties such as CustomerCode, Company, Address, and Phone. Each of these properties will have a data type. Objects are instances of entity types.

The relationships between entities are represented using associations. An association is the relationship between entities, such as the relationship between a Customer and an Order. Figure 20.9 shows an overview of ADO.NET Entity Framework.

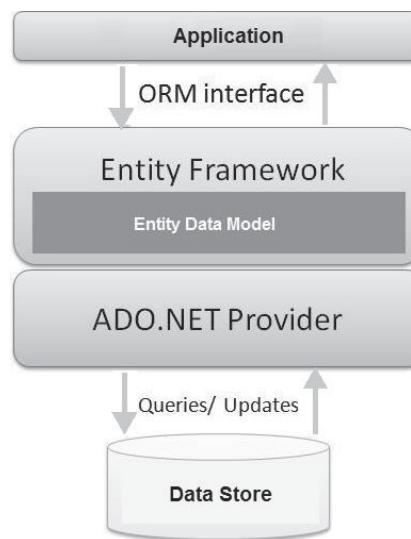


Figure 20.9: ADO.NET Entity Framework

### 20.5.1 Advantages of the ADO.NET Entity Framework

There are many advantages offered by the ADO.NET Entity Framework:

- Developers can now concentrate on application logic because the framework takes care of many data handling tasks, thus, resulting in lesser development time
- Capabilities and benefits of LINQ can be leveraged in the ADO.NET Entity Framework through LINQ to Entities
- LINQ support in the ADO.NET Entity Framework enables you to use IntelliSense and helps to validate query syntax based on the conceptual model at compile-time
- ADO.NET Entity Framework exposes data as strongly typed objects, therefore, developers can take advantage of features such as inheritance and reflection
- Developers can work with applications that support a conceptual model independent of the physical/storage model
- There is no need to frequently change the application code whenever any mapping between the object model and the storage-specific schema changes

Developers write queries using either LINQ or Entity SQL. Entity SQL is a storage-independent language similar to SQL that is designed to query and manipulate objects based on a conceptual model. The Entity Framework converts the LINQ expressions or Entity SQL queries into database queries based on the mapping information supplied.

The Entity Data Model (EDM) is an Entity-Relationship data model used by the ADO.NET Entity Framework. It describes the application-specific object or conceptual model of the data. The EDM builds on the Entity Relationship model and comprises entities and relationships.

### 20.6 Creating a Windows Forms Data Source using EDM

As mentioned earlier, ADO.NET Entity Framework is integrated into the Visual Studio 2008 SP1 IDE. Hence, you can create a Windows Forms application that uses the ADO.NET Entity Framework.

The step by step procedure to use the ADO.NET Entity Framework in a Windows Forms application is described as follows:

1. Launch Visual Studio 2008.
2. Create a new application or open an existing one.
3. Select **Project→Add New Item**. This displays the Add New Item dialog box.
4. Select **ADO.NET Entity Data Model**. Rename the Model1.edmx to MovieModel.edmx as shown in

figure 20.10. An edmx file is an XML file containing the conceptual model, the storage model, and the mappings between them.

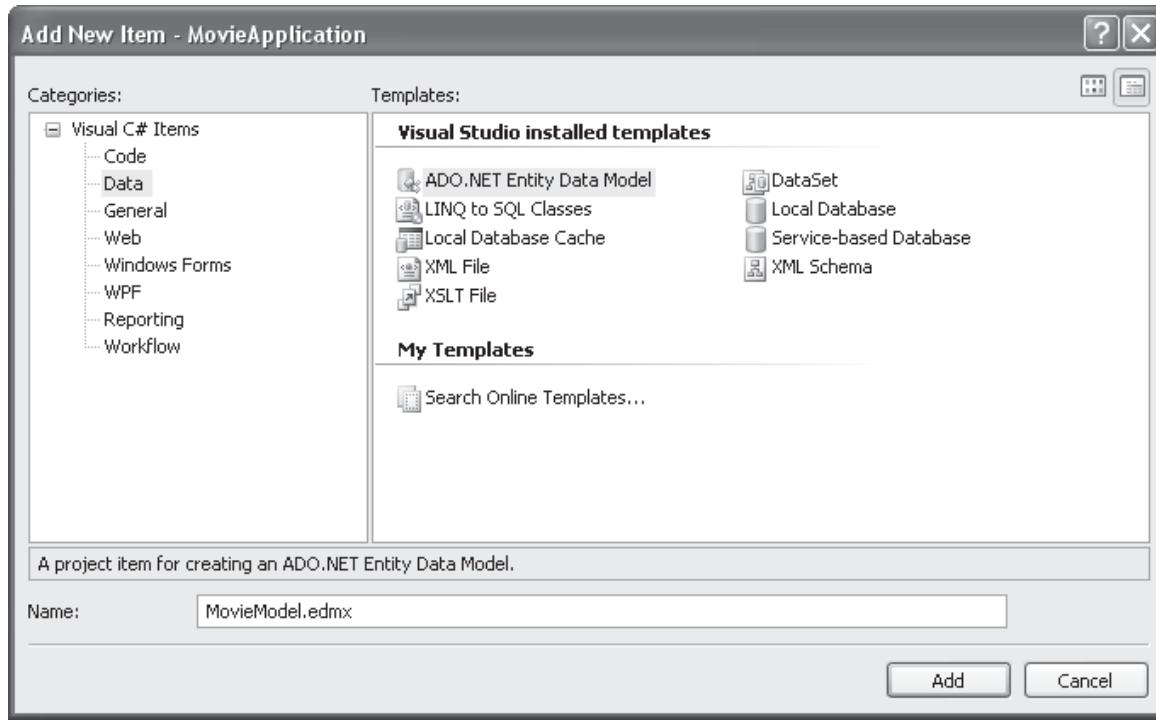


Figure 20.10: Renaming the EDM

5. Click **Add**. This displays the Entity Data Model Wizard as shown in figure 20.11.

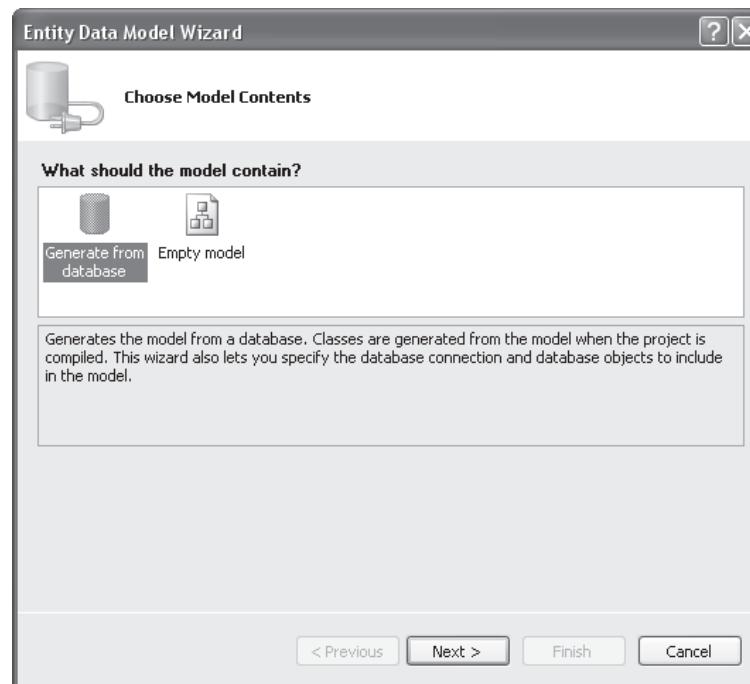


Figure 20.11: Entity Data Model Wizard

6. Select **Generate from Database**. This launches the Choose Your Data Connection page as shown in figure 20.12.

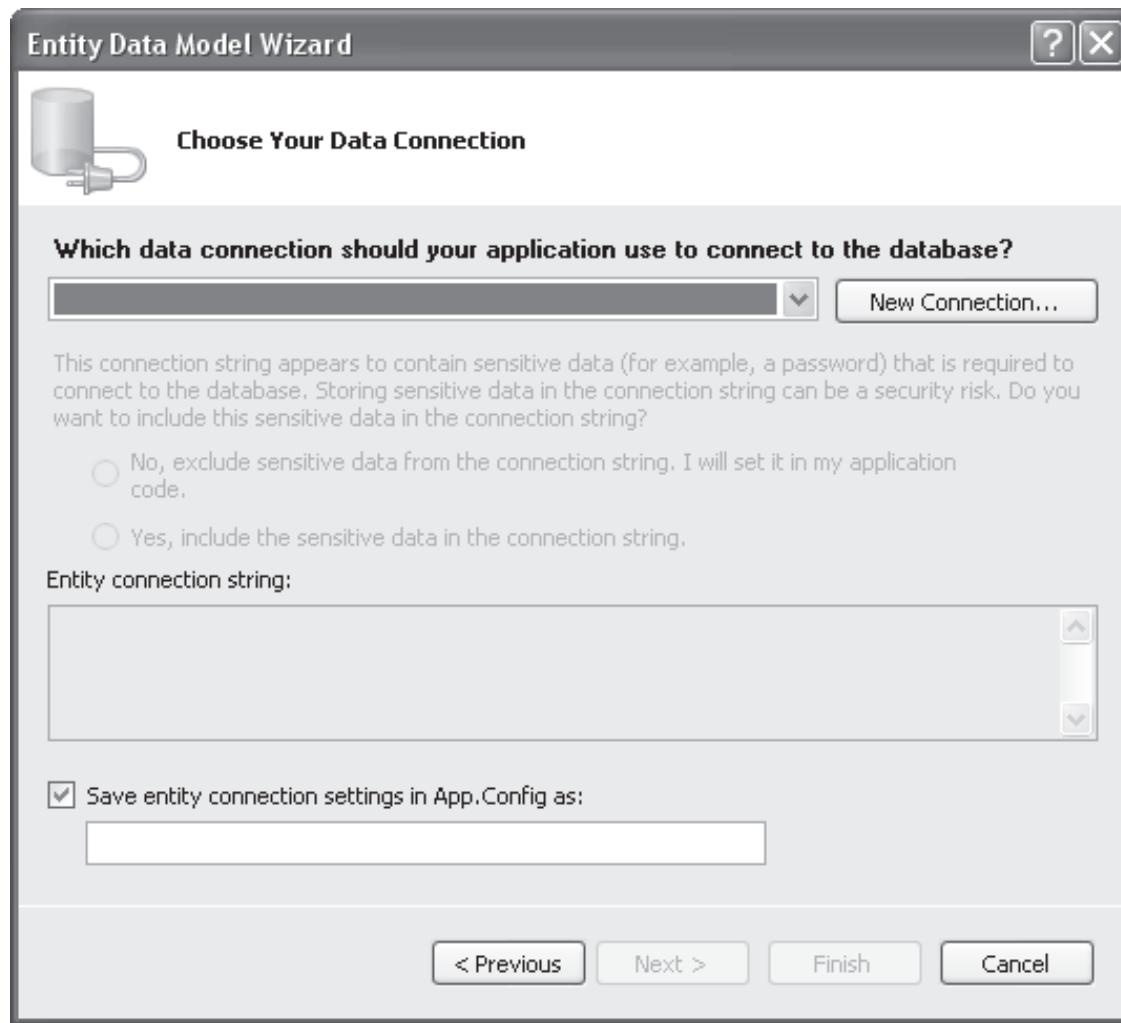


Figure 20.12: Choose Your Data Connection

7. Click **New Connection** and enter the SQL Server details as shown in figure 20.13.

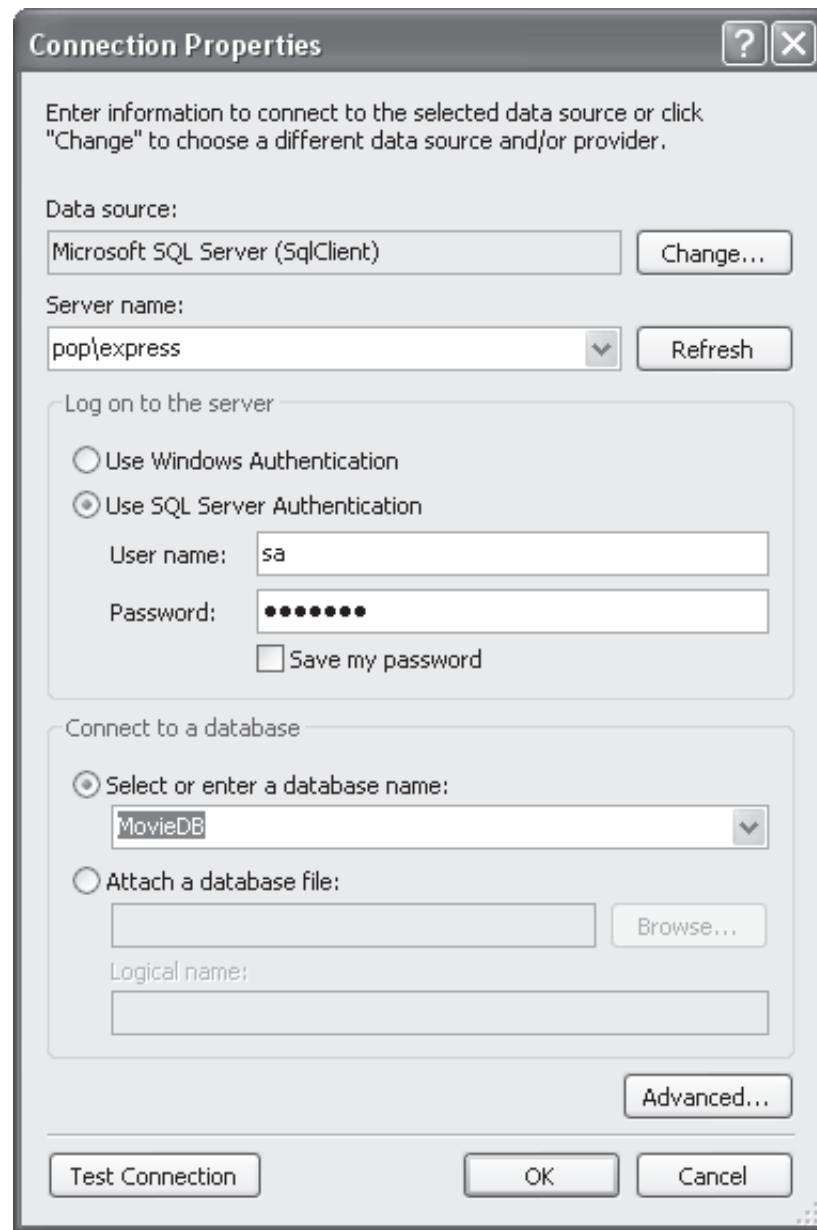


Figure 20.13: Connection Properties

8. Click **OK**. This will populate the connection details as shown in figure 20.14.

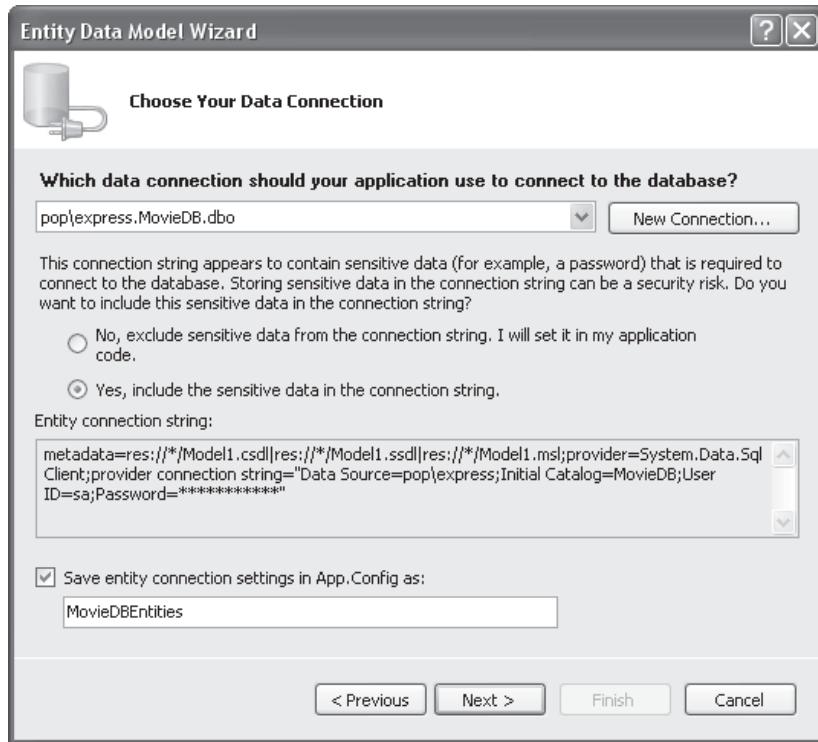


Figure 20.14: Connection Details Populated

9. Click the option button **Yes, include the sensitive data in the connection string**. It is recommended that you always click the **No, exclude...**option button but in this case, you click **Yes,..**

10. Click **Next**. This will display the Choose Your Database Objects page as shown in figure 20.15.

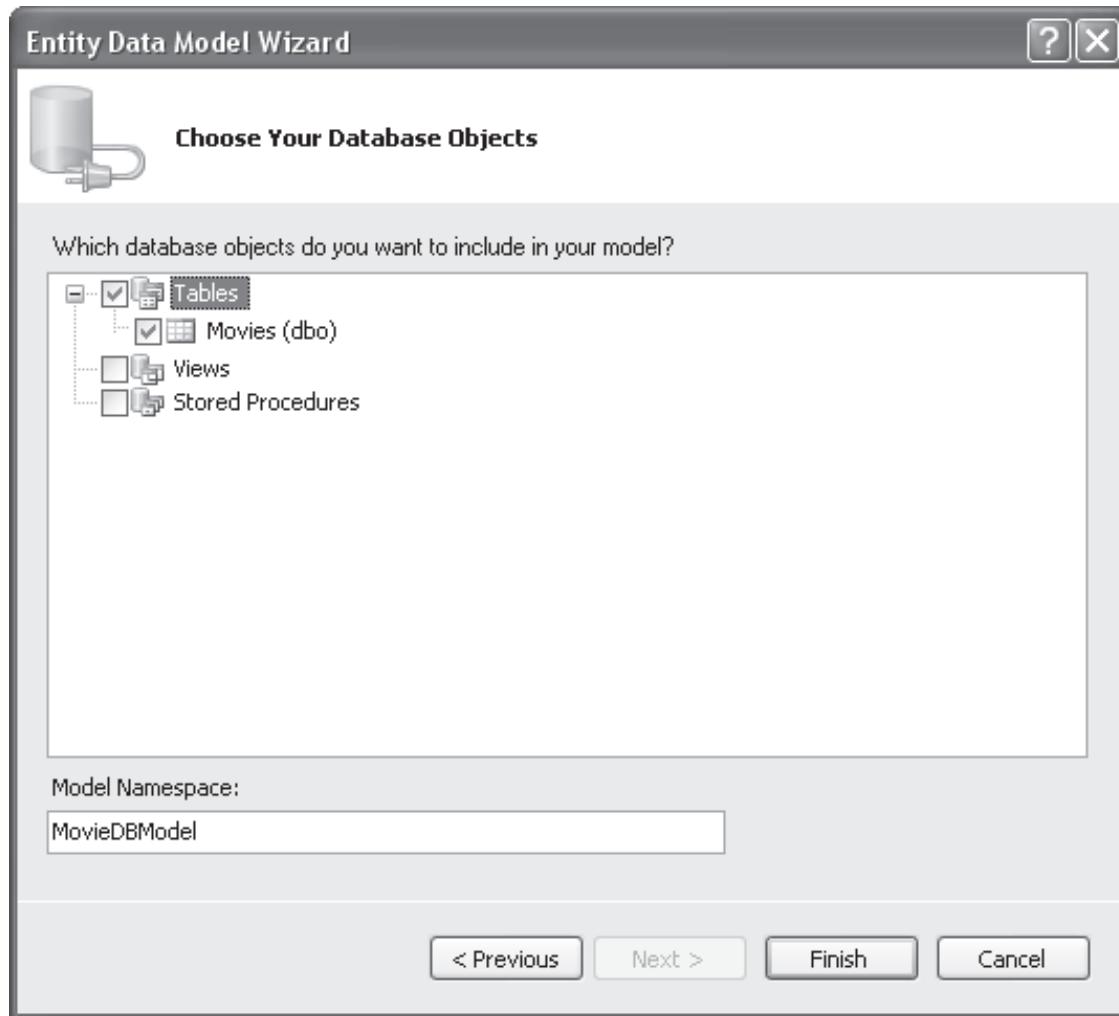


Figure 20.15: Choose Your Database Objects Dialog Box

11. In the **Choose Your Database Objects** dialog box, specify the table name to include in the data model. Select the Movies table and click **Finish**. This results in the EDM as shown in figure 20.16.

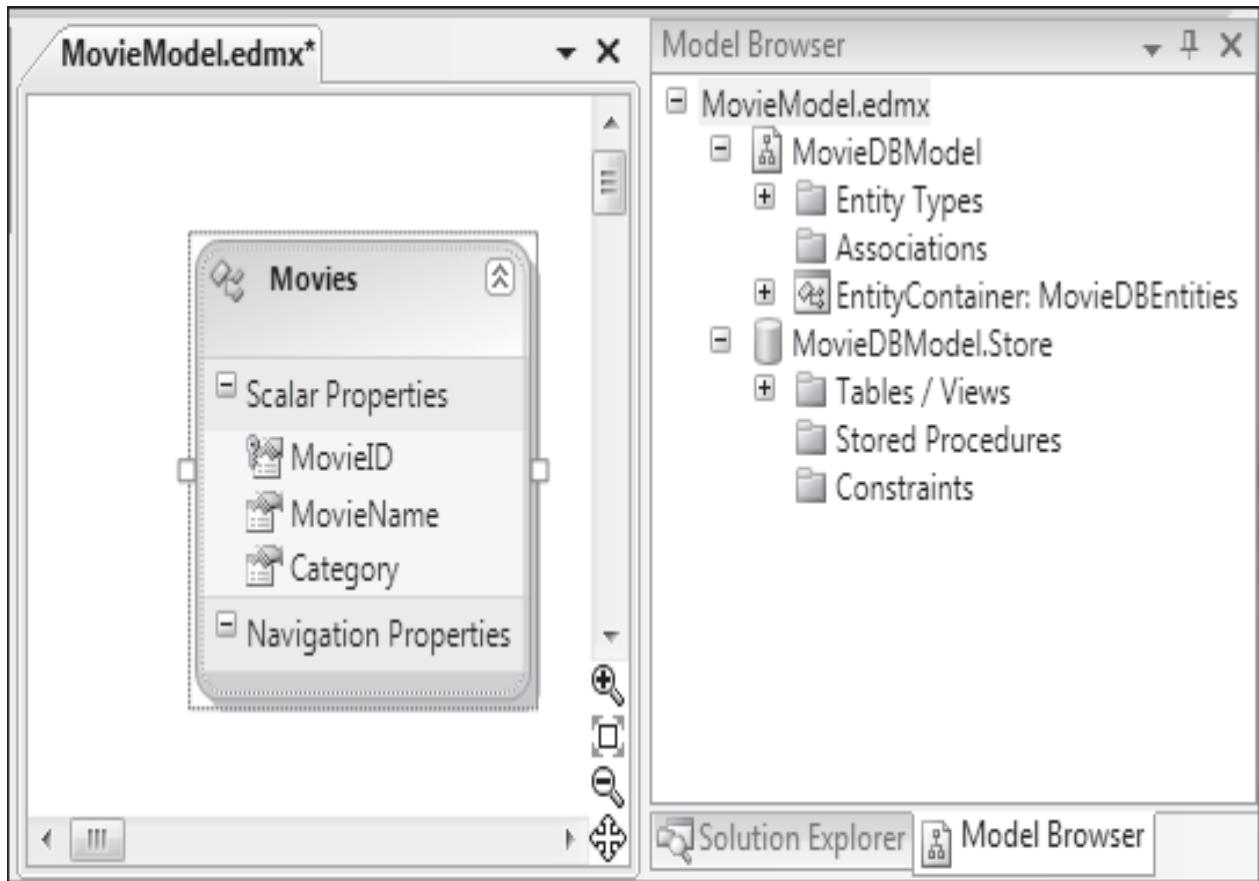


Figure 20.16: MovieModel.edmx

If you right-click the Movies entity and select **Properties**, you will see the properties as shown in figure 20.17.

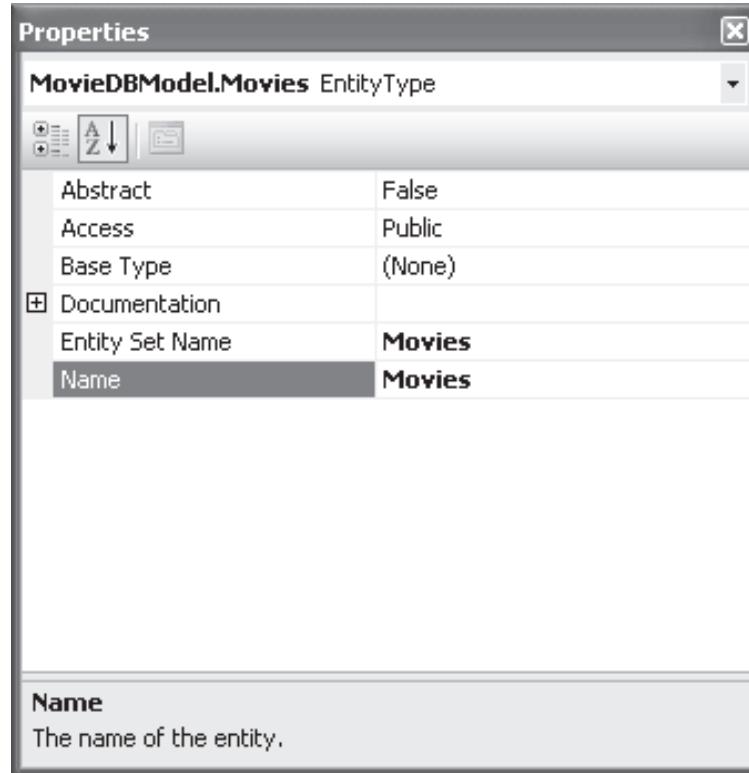


Figure 20.17: Properties of the Entity Movies

If you right-click anywhere in the designer area and select **Properties**, you will see the properties of the conceptual entity model as shown in figure 20.18.

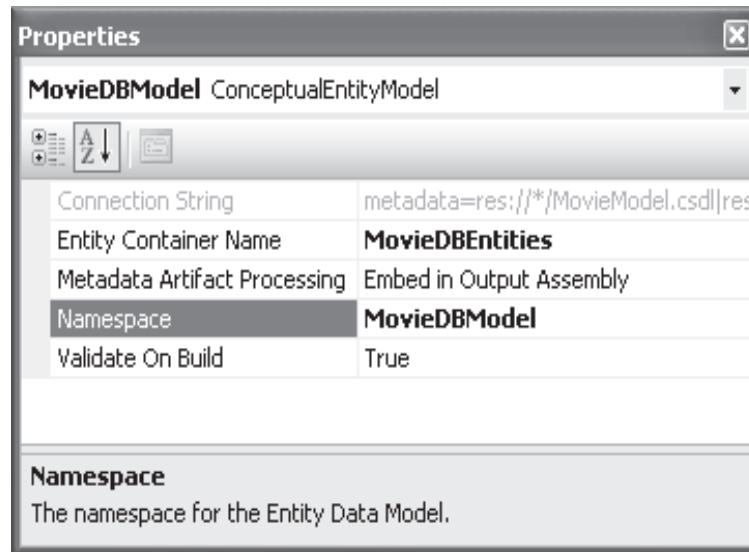


Figure 20.18: MovieDBModel Conceptual Entity Model

Now, you need to make use of this EDM in your Windows Forms application. The steps for this are described as follows:

1. Add a new data source using **Add Data Source** option from the **Data** menu. This displays the Data Source Configuration Wizard.
2. Select **Object** as the Data Source type.
3. Click **Next**. The next page displays the namespaces and classes that are available in the solution as a tree-like structure. These are the namespaces and classes that the wizard has created based on the database tables. Here, in this example, you have only one namespace.
4. Expand the tree node on the namespace to see the various classes. Figure 20.19 shows the outcome.

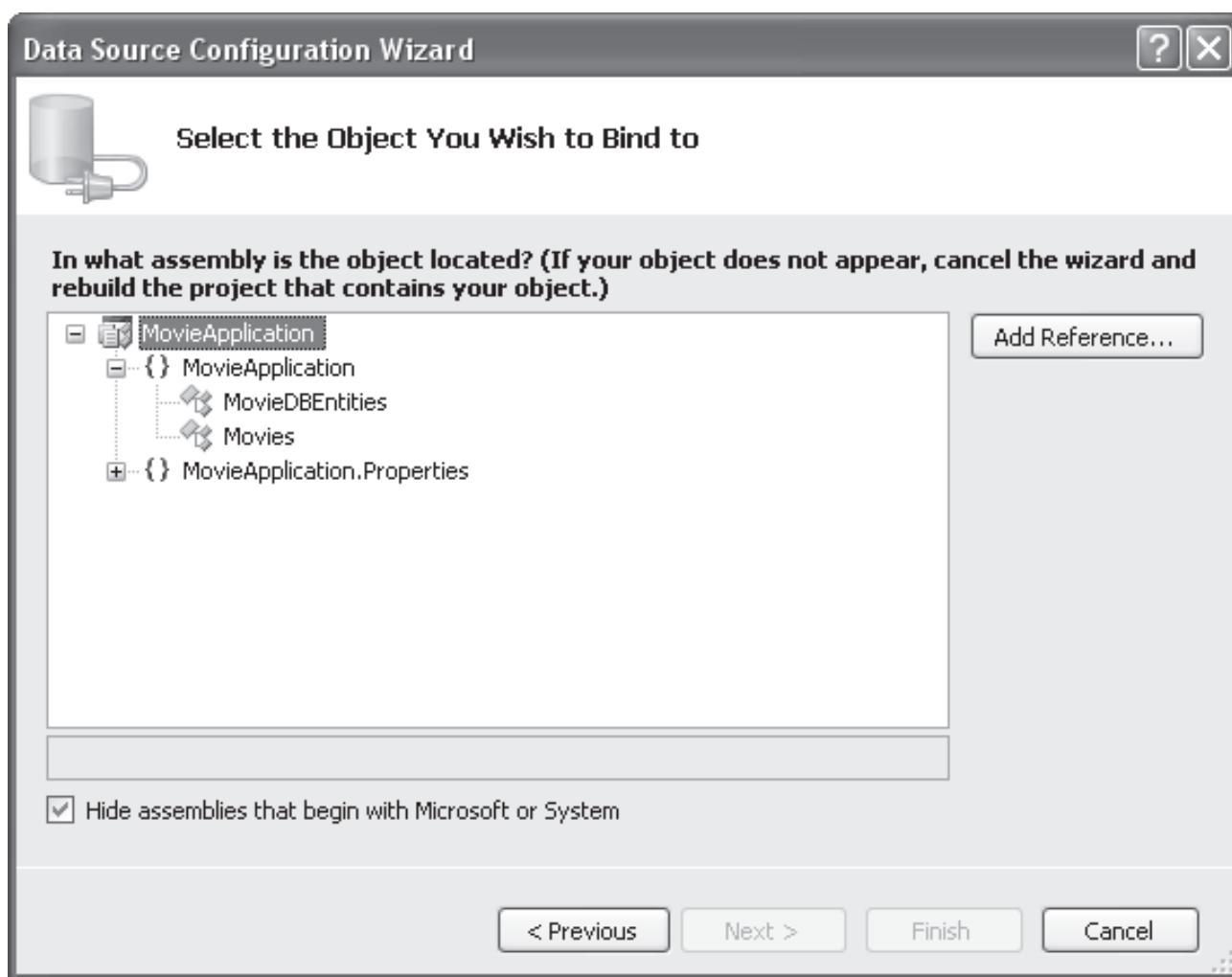


Figure 20.19: Selecting Object to Bind

5. Select the assembly based on which you want to create the Object Data Source. In this case, select Movies. The object, Movies, will be added as shown in figure 20.20.

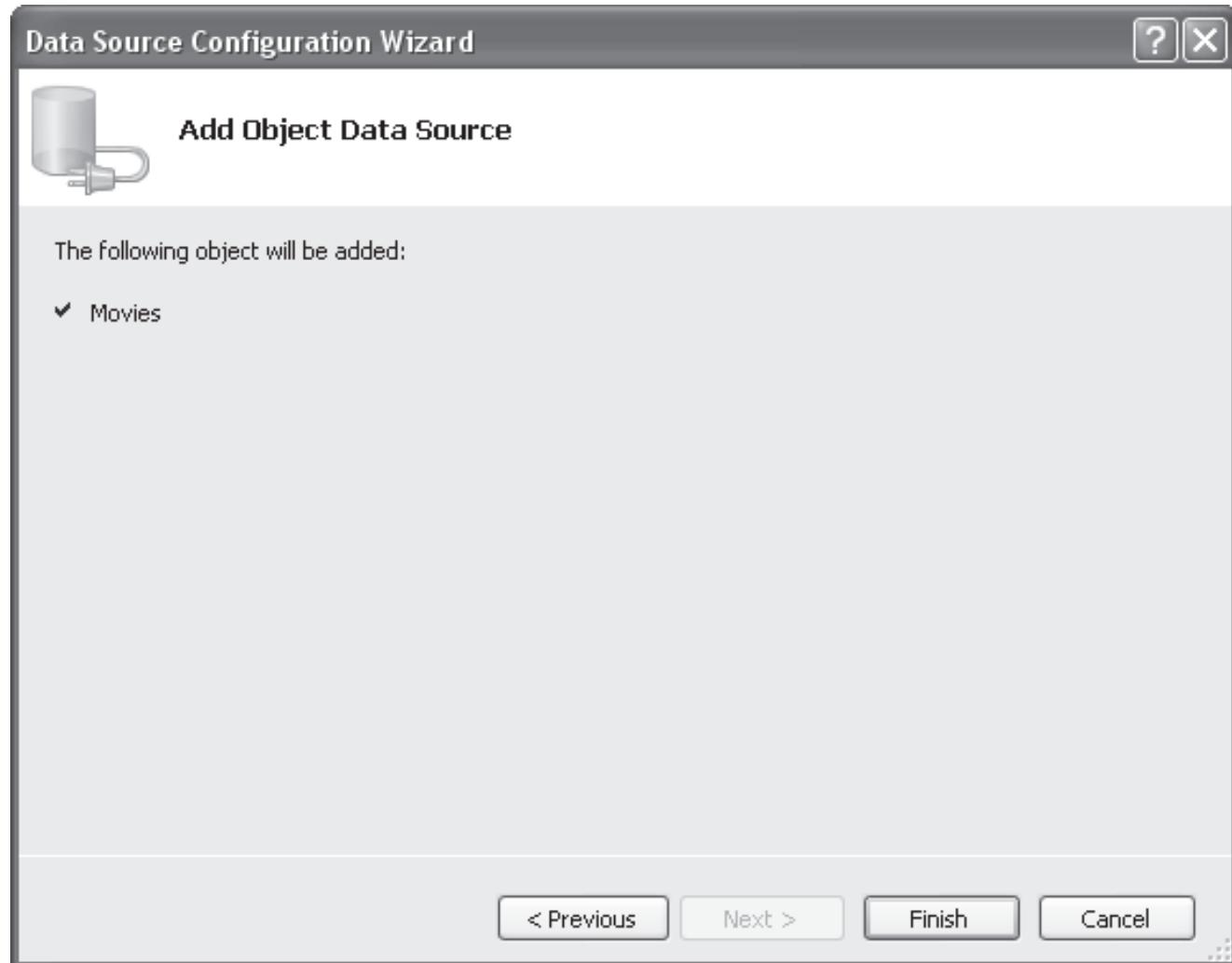


Figure 20.20: Adding the object data source

Once the data source is added, use the **Data→Show Data Sources** option to view it. Drag and drop the Movies data source from the **Data Sources** window to the form. This will auto-generate a **DataGridView** with a **BindingSource** and a **BindingNavigator** and bind them.

Assume that a form, `frmDataTest`, is created. Generate the `Load` event for the Form. Copy the code in Code Snippet into the form's `Load` event handler to access the `Movies` entity and bind to the database.

#### Code Snippet :

```
MovieDBEntities mve;  
  
private void frmDataTest_Load(object sender, EventArgs e)  
{
```

```
mve = new MovieDBEntities();
var movies = mve.Movies;
moviesBindingSource.DataSource = movies;
}
```

As you can see, no data access code is written here. The process is simple. You created a reference to **MovieDBEntities**, the Entities wrapper class. Then, you retrieved the movies into an implicitly typed variable named **movies**, and finally, you bind the data in movies to the **BindingSource**. The **BindingSource** has already been bound to the grid when you dragged the data source to the form.

This completes the process of creating and binding a data source in Windows Forms using ADO.NET Entity Framework.

## 20.7 Querying the EDM

You can access data from the EDM using LINQ to Entities. LINQ to Entities is a mechanism using LINQ to work with data in entities generated by the EDM.

Consider that you have an additional form named **frmFilterData** in your application. This form contains a **ListBox** into which you want to load all the categories of movies from the Movies database. Assume **MovieDBEntities** is already created. You need to create code as shown in Code Snippet.

### Code Snippet:

```
MovieDBEntities mve;
private void frmFilterData_Load(object sender, EventArgs e)
{
    mve = new MovieDBEntities();
    var categories = from movie in mve.Movies
                    select movie.Category;
    lstCategory.Items.AddRange(categories.ToArray());
}
```

The **AddRange()** method of **ListBox** adds a set of values into the **Items** collection. The implicitly typed variable **categories** is converted into a string array using **ToArray()**.

However, this query returns many duplicate values in the list box. To retrieve only the unique values, you will need to use the **Distinct** standard LINQ operator.

Therefore, the code is now modified as shown in Code Snippet.

**Code Snippet:**

```
MovieDBEntities mve;  
  
private void frmFilterData_Load(object sender, EventArgs e)  
{  
    mve = new MovieDBEntities();  
  
    var categories = from movie in mve.Movies  
                      select movie.Category;  
  
    var distinctCategories = categories.Distinct();  
  
    lstCategory.Items.AddRange(distinctCategories.ToArray());  
}
```

After the form is loaded, the list box will be populated with data retrieved from the object data source created using ADO.NET Entity Framework and retrieved using LINQ to Entities.

The output is as shown in figure 20.21.



Figure 20.21: Object DataSource Output

ADO.NET Entity Framework and LINQ to Entities also enable you to construct complex queries with several entities and relationships between related tables.

## Knowledge Check 1

1. Which of the following statements about standard query operators are true?

(A)	Can be called directly as methods
(B)	Provide querying capability over the collection on which they operate
(C)	Are static methods of System.Linq.Enumerable and System.Linq.Queryable
(D)	Can be used as only one standard query operator in a statement at a time

2. Which of the following options shows the output generated in the **ListBox** by the following code?

```
string[] names = {"John", "Mark", "Peter", "John", "Kim", "John"};
var result = from name in names
             orderby name
             select name;
foreach (string i in result.Distinct())
{
    lstBooks.Items.Add(i);
}
```

(A)	John Kim Mark Peter
(B)	John John John Kim Mark Peter
(C)	John Mark Peter Kim
(D)	Kim Mark Peter

3. Which of the following is not a flavor of LINQ?

(A)	LINQ to SQL
(B)	LINQ to XML
(C)	LINQ to Classes
(D)	LINQ to DataSets

4. Which of the following are important classes of LINQ to XML?

(A)	XDocument
(B)	XElement
(C)	XNode
(D)	XAttribute

5. LINQ support for **DataSet** is implemented entirely in \_\_\_\_\_.

(A)	System.Data.Extensions.dll
(B)	System.Data.dll
(C)	System.Extensions.dll
(D)	System.Data.Linq.dll

6. Which of the following statements about ADO.NET Entity Framework are true?

1. ADO.NET Entity Framework and its tools enable you to create entity model databases using wizards.
2. Capabilities and benefits of LINQ can be leveraged in the ADO.NET Entity Framework through LINQ to Entities.
3. ADO.NET Entity Framework is designed to work only with Windows Forms.
4. ADO.NET Entity Framework exposes data as strongly typed objects, therefore, developers can take advantage of features such as inheritance and reflection.

(A)	1,2,4
(B)	1,2,3
(C)	1,4
(D)	1,4,3

7. Which of the following options correctly identify the outcome of the following code?

```
MovieDBEntities mve;  
  
private void frmDataTest_Load(object sender, EventArgs e)  
{  
  
    mve = new MovieDBEntities();  
  
    var movies = mve.Movies;  
  
    moviesBindingSource.DataSource = mve;  
  
}
```

(A)	Binds the <b>BindingSource</b> , <b>moviesBindingSource</b> , to the movies table
(B)	Binds the <b>moviesBindingSource</b> to the movies class
(C)	Binds the <b>moviesBindingSource</b> to the movies entity
(D)	Causes a compiler error because a <b>BindingSource</b> cannot be directly bound to the entities class



## Module Summary

- LINQ is a set of technologies introduced in Visual Studio 2008 that simplifies data access in various formats from different data sources.
- A query expression is a query that is written in query syntax and is used to query and transform data from a LINQ-compatible data source.
- Standard query operators represent the operations available in LINQ and are used for tasks such as filtering, counting, aggregating, sorting, and so forth.
- LINQ can be categorized into following flavors: LINQ to SQL, LINQ to XML, LINQ to Objects, and LINQ to DataSets.
- ADO.NET Entity Framework is an ORM framework integrated into Visual Studio 2008.
- The EDM is an Entity-Relationship data model used by the ADO.NET Entity Framework that describes the application-specific objects.

# Module - 21

## LINQ and ADO.NET Entity Framework (Lab)

In this Module, you will learn about:

- Create an application using LINQ and ADO.NET Entity Framework
- Create an application to construct an XML tree using LINQ
- Develop an application using LINQ to SQL

## Part I – 60 Minutes

### Exercise

Michigan Holidays is a company in USA that offers exciting trips to various tourist destinations. It was founded in 2001 to provide holidays for all class of citizens.

Michigan Holidays started with a single resort in Washington. Presently, the company has 51 resorts in USA and abroad, and has a customer base of over 100,000 members.

At present, the company has a manual system to handle all kind of bookings. The management has decided to automate the entire tasks. The first step is that they want their customers to view the details of the resorts.

The company has asked SoftData Technologies to create an application that gives the details of the resort.

Consider that you are one of the programmers in the IT team and are assigned the following tasks:

- Create a database that stores the details of the resort
- Design an interface for the application using Windows Forms
- Generate an Entity Data Model
- Query Entities and perform data binding
- Retrieve and display the records

#### Solution:

##### Design the MichiganHolidays database and Resorts table

To create a database named, MichiganHolidays, perform the following steps:

1. Open Microsoft SQL Server Management Studio. Enter the appropriate login details.
2. Create a new database named MichiganHolidays.

**3. Design a table named Resorts with the structure specified in figure 21.1.**

Column Name	Data Type	Allow Nulls
ResortID	int	<input type="checkbox"/>
ResortName	nchar(20)	<input type="checkbox"/>
Country	nchar(15)	<input type="checkbox"/>
Summary	nchar(500)	<input type="checkbox"/>
Rate	int	<input type="checkbox"/>
Photo	varbinary(MAX)	<input checked="" type="checkbox"/>

Figure 21.1: Resorts Table Structure

**4. Insert a record in the table using INSERT command.**

```

INSERT INTO Resorts
    (ResortID,
     ResortName,
     Country,
     Summary,
     Rate,
     Photo)

    SELECT
        10
        , 'Michigan Rain Forest'
        , 'USA'
        , 'Michigan Rain Forest is a wonderful resort on Lake Quinault at the
          southern entrance of Olympic National Park on Washington State's Olympic
          Peninsula.'
        , $500
        , * FROM OPENROWSET (BULK N'c:\MichiganRF.jpg', SINGLE_BLOB) as Photo
  
```

A record is added to the Resorts table.

**5. Insert more records and display the records.**

```

USE MichiganHolidays
GO

SELECT * FROM RESORTS
  
```

Figure 21.2 displays the records of Resorts table.

	ResortID	ResortName	Country	Summary	Rate	Photo
1	10	Michigan Rain Forest	USA	Michigan Rain Forest is a wonderful resor...	500	0xFFD8FFE000104A46494
2	20	Michigan Saga Resort	USA	Michigan Saga Resort is situated on Bolg...	850	0xFFD8FFE000104A46494
3	30	Michigan Dreams	Malaysia	The Royal Malaya Dreams, the "grand p...	675	0xFFD8FFE000104A46494
4	40	Michigan Luxury Ness	England	Experience the luxury of a Luxury Ness C...	675	0xFFD8FFE000104A46494

Figure 21.2: Resorts Table

Three more records are added to the table with all the details. Here, the image is stored as binary data.

### Create a Windows Forms Application

You need to create a Windows Forms application for displaying the resort details. To create the application using Visual Studio 2008, perform the following:

1. Open Microsoft Visual Studio 2008 IDE. On the File menu, click New→Project. The New Project dialog box is displayed.
2. Choose Visual C# in the Project types pane.
3. Select Windows Forms Application in the Templates pane.
4. Name the project as MichiganHolidays and click OK.

Figure 21.3 shows the New Project dialog box.

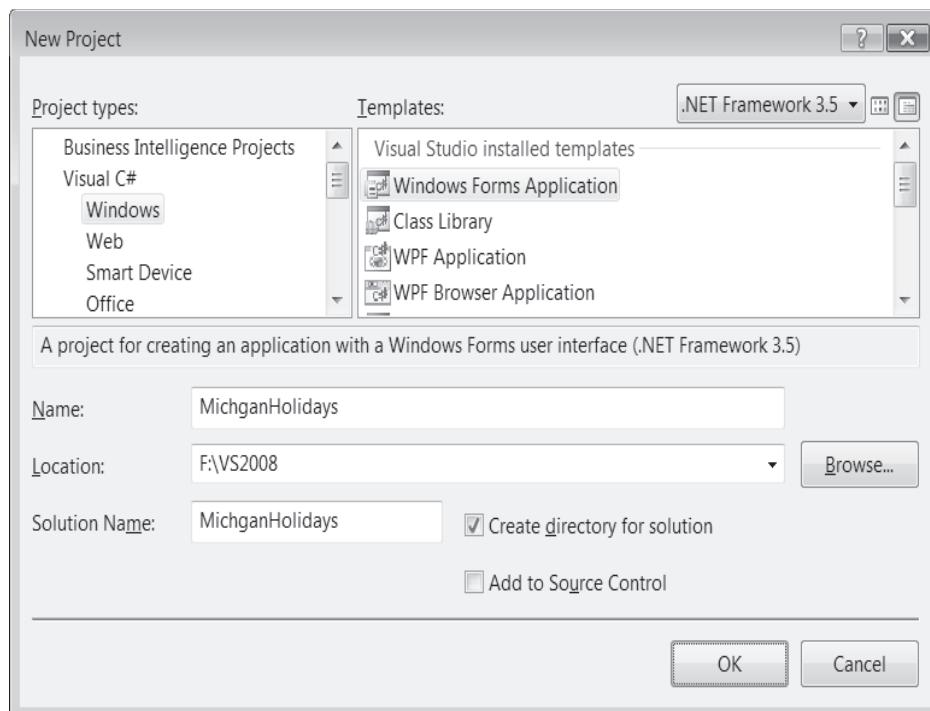


Figure 21.3: New Project Dialog Box

5. Rename Form1.cs to MichiganHolidays.cs.
6. Design the user interface as shown in figure 21.4.



Figure 21.4: Form Design

The details of the controls are listed in table 21.1.

Control	Property	Value
Form	Name	frmMichiganHolidays
	Text	Michigan Holidays
Label	Name	lblHeading
	Text	Resorts
Label	Name	lblCountry
	Text	Country:
Label	Name	lblDetails
	Text	Details:
ComboBox	Name	cboCountry
DataGridView	Name	dgvDetails
Button	Name	btnUpdate
	Text	Update
Button	Name	btnExit
	Text	Exit

Table 21.1: Controls

### Generate an ADO.NET Entity Data Model

In this entity data model, the form uses an SQL Server 2008 database named MichiganHolidays. To add the ADO.NET Entity Data Model item template, perform the following steps:

1. Select the project in Solution Explorer. In the Project menu, select Add New Item. This displays the Add New Item dialog box.
2. Select ADO.NET Entity Data Model from the Templates pane.
3. Type ResortsDataModel.edmx in the Name box as shown in figure 21.5 and click Add.

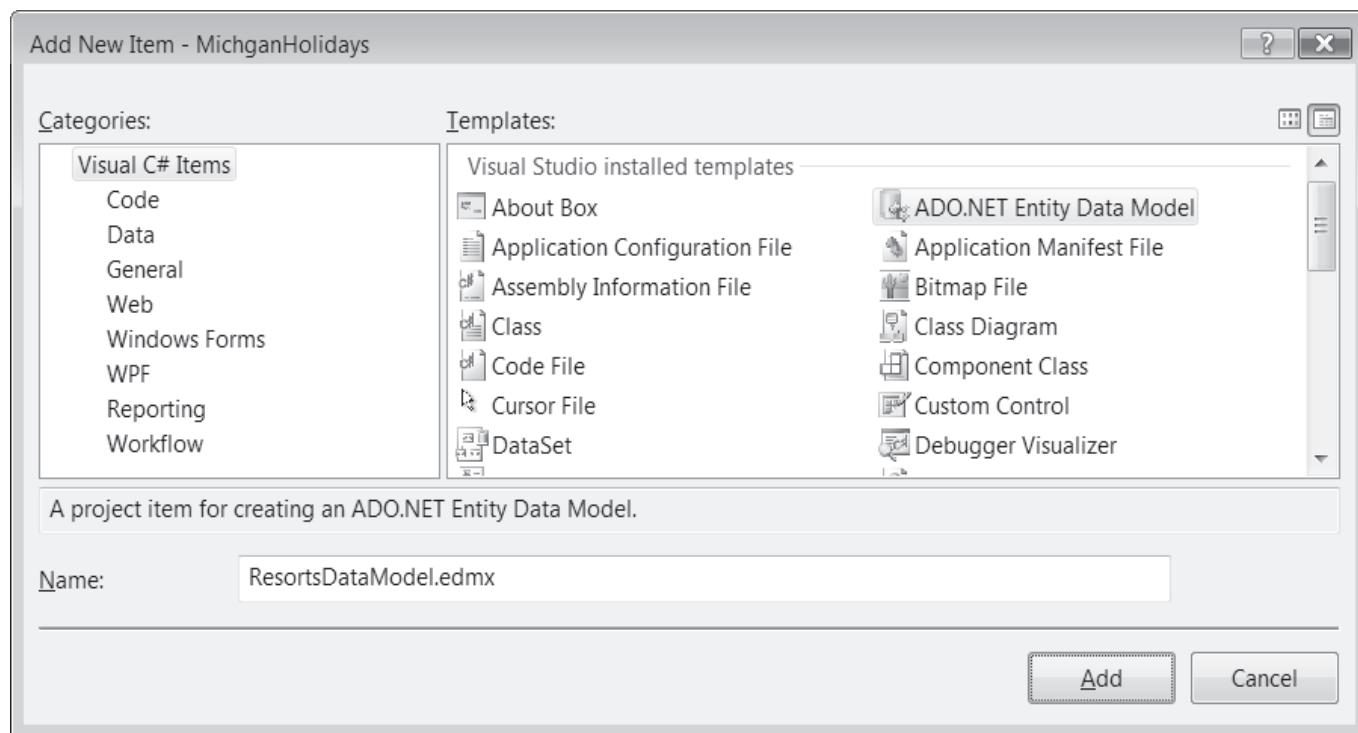


Figure 21.5: Add New Item Dialog Box

This displays the opening page of the Entity Data Model Wizard.

4. Select Generate from database in the Choose Model Contents dialog box as shown in figure 21.6.

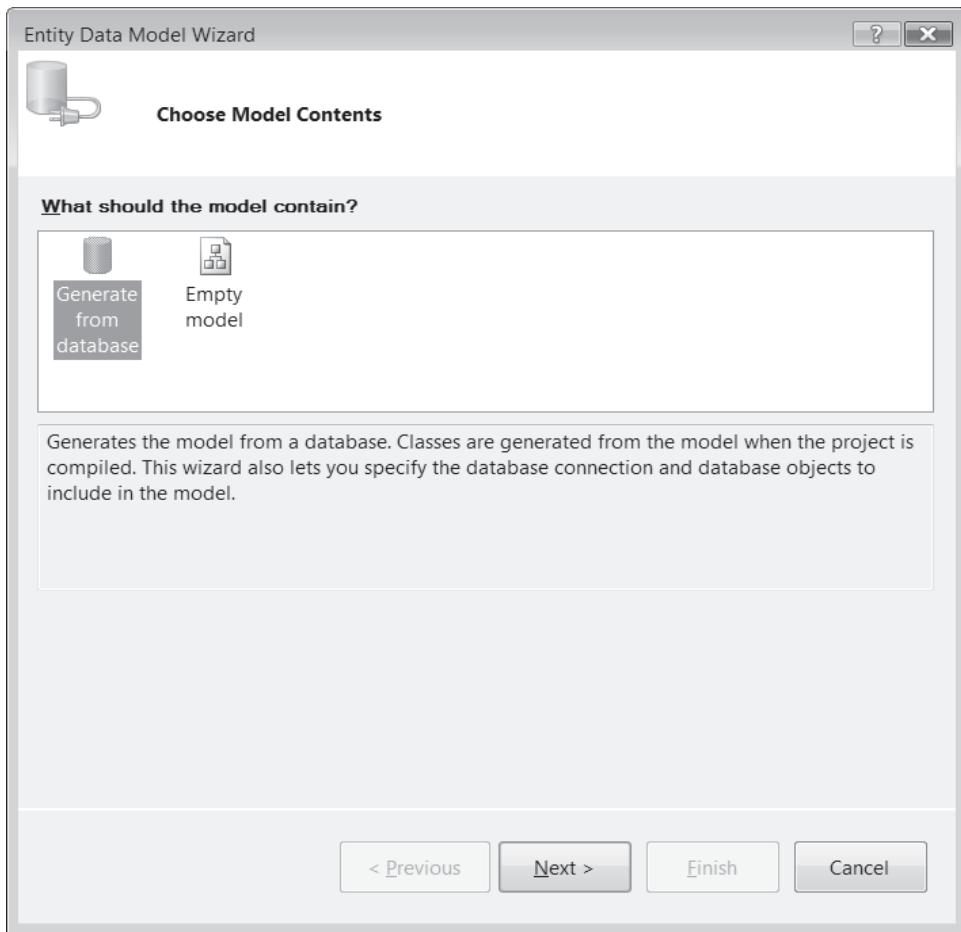


Figure 21.6: Choose Model Contents Dialog Box

5. Click Next.

The Choose Your Data Connection dialog box is displayed as shown in figure 21.7.

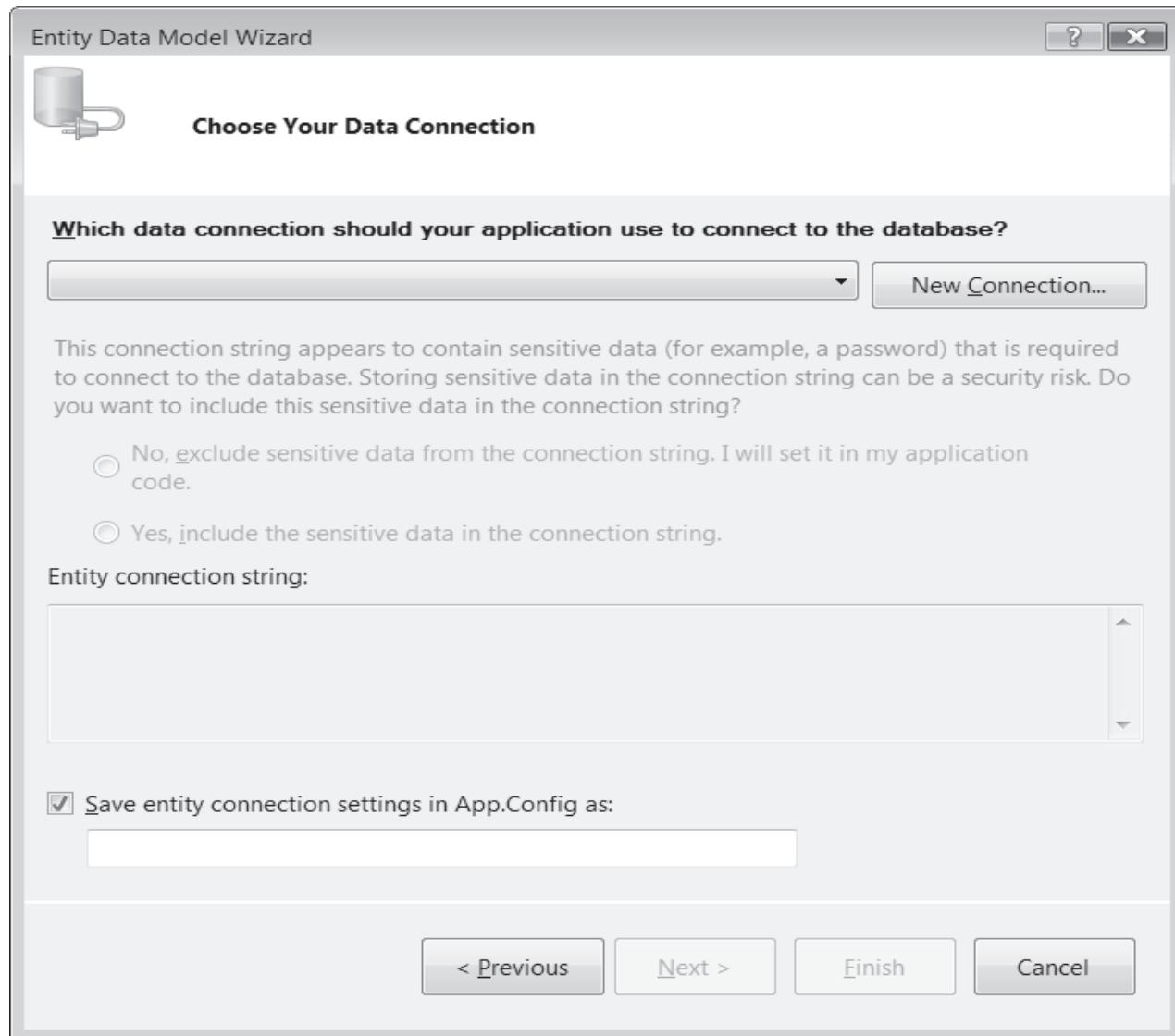


Figure 21.7: Choose Your Data Connection Dialog Box

6. Click New Connection. The Connection Properties dialog box is displayed.
7. Enter the connection details as shown in figure 21.8 and Click OK.

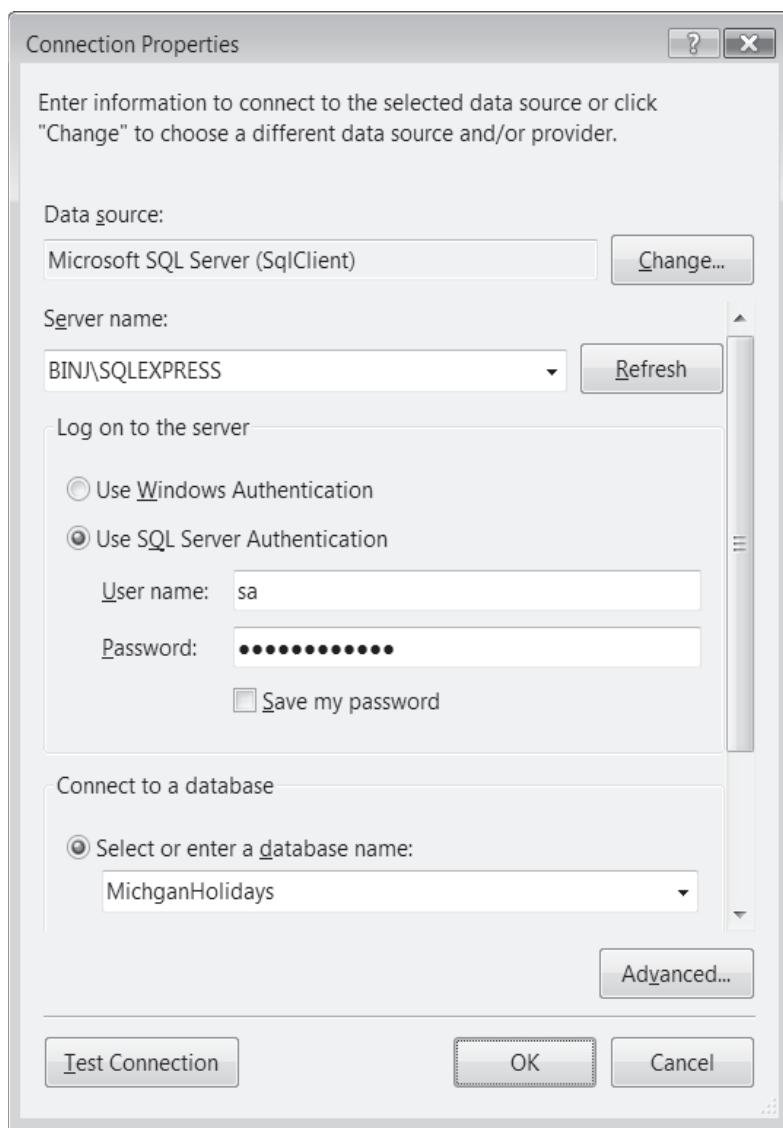


Figure 21.8: Connection Properties Dialog Box

The Choose Your Data Connection dialog box is updated with the database connection properties.

8. Select Yes, Include the sensitive data in the connection string. option button.
9. Type the name as MichiganResortsEntities in Change the name of Save entity connection settings in App.Config as box as shown in figure 21.9.

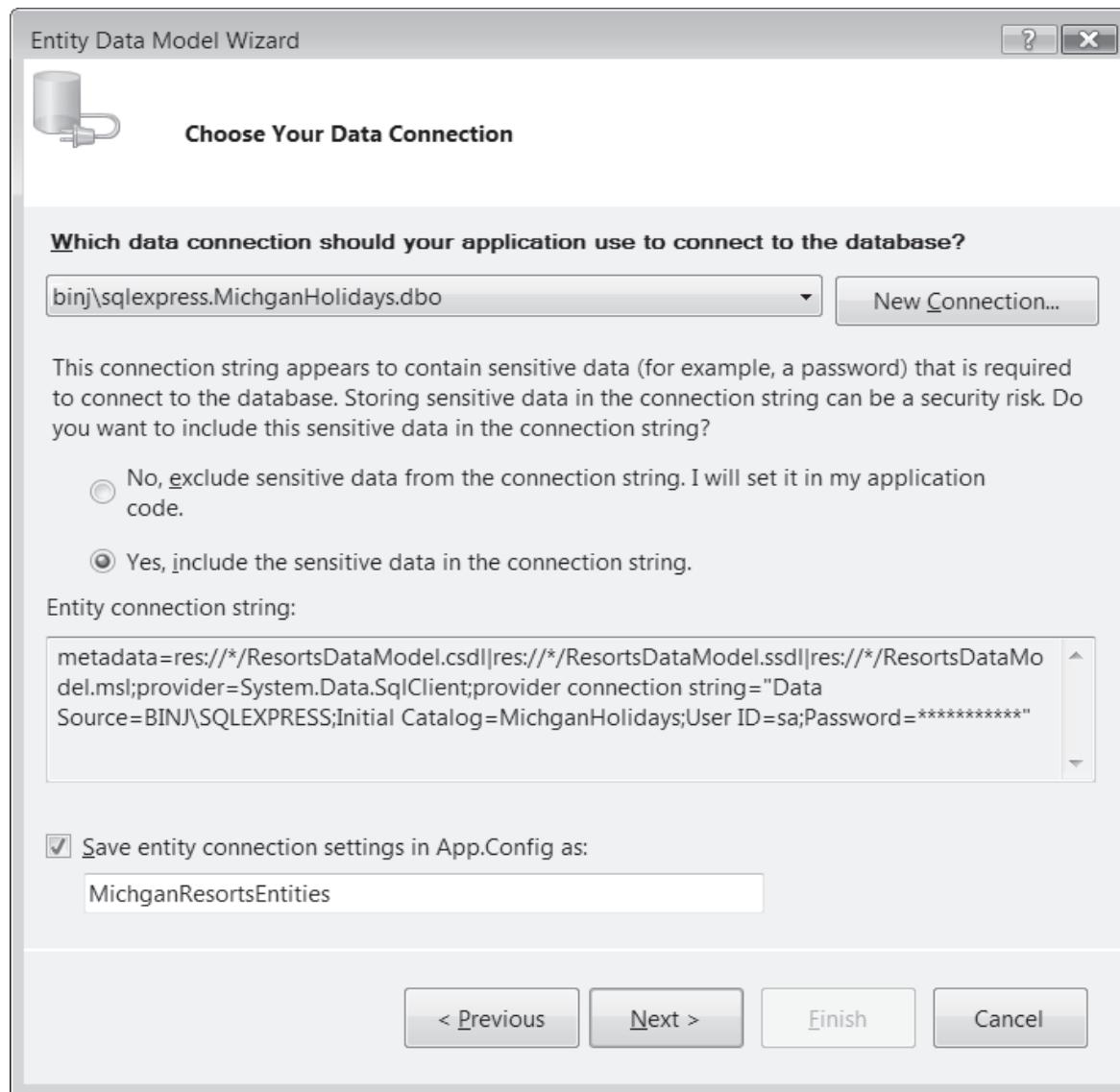
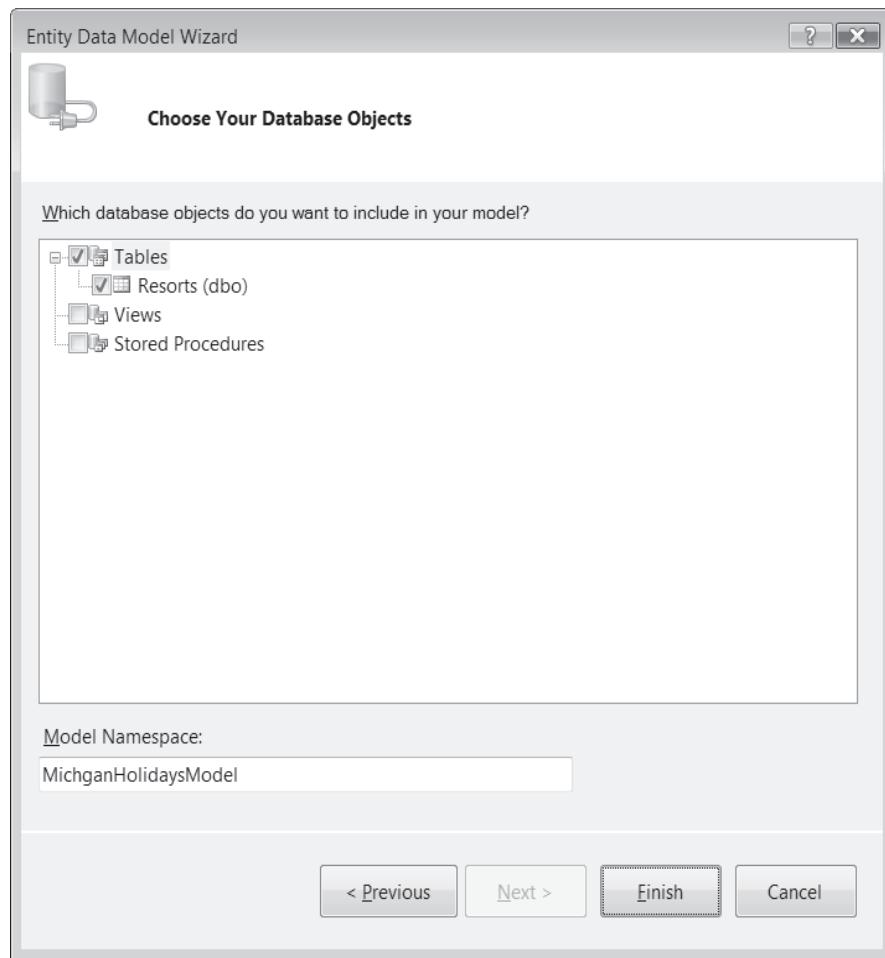


Figure 21.9: Database Connection Details

10. Click Next.

The Choose Your Database Objects dialog box is displayed.

**11. Select the Resorts table as shown in figure 21.10.****Figure 21.10: Choose Your Database Objects Dialog Box****12. Click Finish to complete the wizard.**

After the wizard has finished execution, the following tasks are performed:

- The wizard adds the following assemblies as references to the application: `System.Data.Entity`, `System.Runtime.Serialization`, and `System.Security`
- Generates the `ResortsDataModel.edmx` file that defines the entity data model
- Creates a source code file that contains the classes that were generated based on the entity data model. You can view the contents of this file by expanding the `ResortsDataModel.edmx` file
- Creates the `App.Config` file that contains the database connection properties

The contents of the App.Config file is shown figure 21.11.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
    <connectionStrings>
        <add name="MichiganResortsEntities"
            connectionString="metadata=res://*/ResortsDataModel.csdl|res://*/
ResortsDataModel.ssdl|res://*/ResortsDataModel.msl;provider=System.Data.
SqlClient;provider connection string="Data Source=BINJ\SQLEXPRESS;
Initial Catalog=MichiganHolidays;User ID=sa;Password=michganholidays_123;
MultipleActiveResultSets=True"" providerName="System.Data.EntityClient
" />
    </connectionStrings>
</configuration>
```

Figure 21.11: App.Config File

To display the ADO.NET Entity Data Model Designer and view the entity data model, perform the following steps:

- 13. Double-click ResortsDataModel.edmx file from Solution Explorer. Figure 21.12 displays the model in the ADO.NET Entity Data Model Designer Window.**

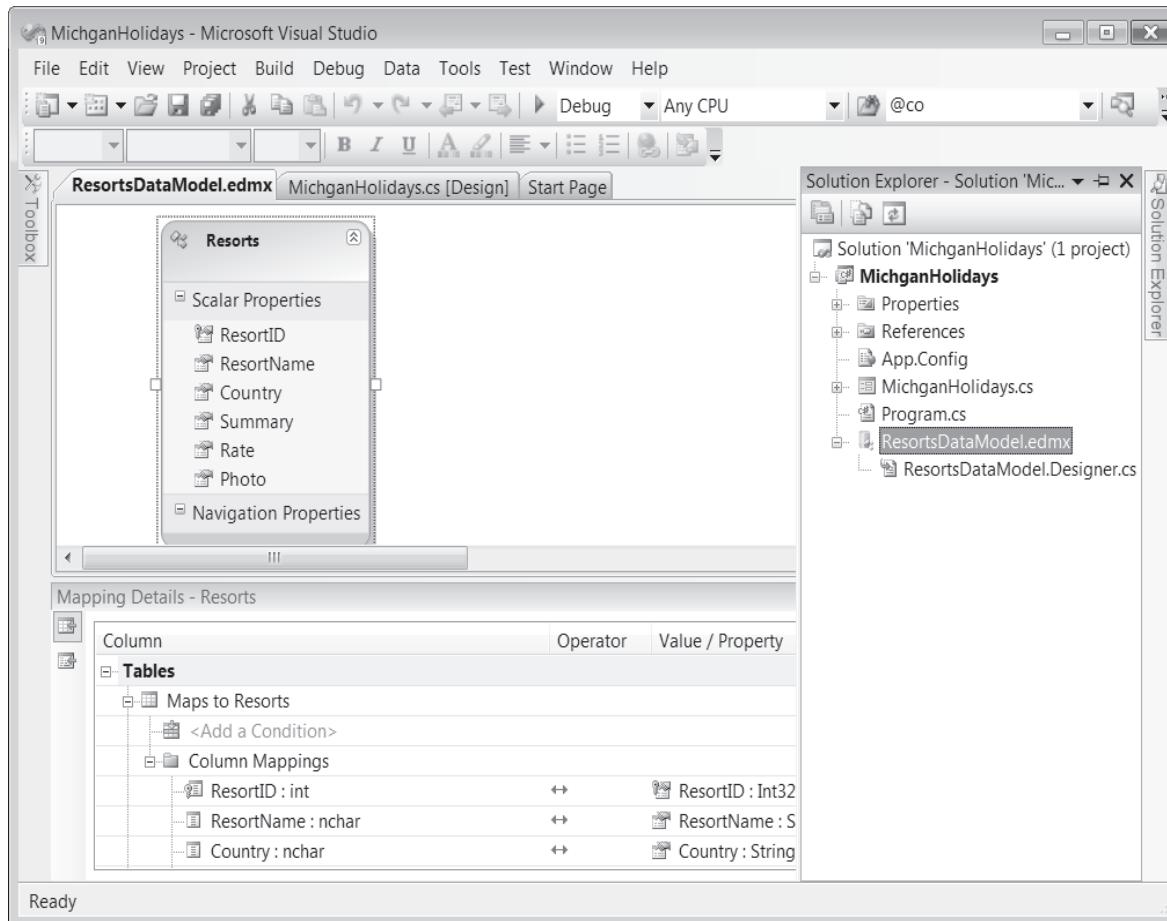


Figure 21.12: Entity Data Model

## Query Entities and Data Binding

Next step is to query the database. To do this, perform the following steps:

- Add the following namespaces in the beginning of MichiganHolidays.cs code-behind file:**

```
using System.Data.Objects;
using System.Data.Objects.DataClasses;
```

These namespaces enable you to refer to the model created from the database. You can query, insert, update, and delete data by using the classes in these namespaces.

- Create variable named objResorts in the class definition of the form.**

```
/// <summary>
/// Class that inserts and updates data
/// to the database using entity data model.
/// </summary>
public partial class frmMichiganHolidays : Form
{
    MichiganResortsEntities objResorts;
```

A variable named objResorts of type MichiganResortsEntities is created.

- Double-click the form. The Load event handler method is generated.**

- Add code in Load event handler method of frmMichiganHolidays.**

```
/// <summary>
/// Populates the Country in the Combo Box
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void frmMichiganHolidays_Load(object sender, EventArgs e)
{
    objResorts = new MichiganResortsEntities();

    var query = (from c in objResorts.Resorts
```

```

        select c.Country).Distinct();

try
{
    this.cboCountry.DataSource = query;
}
catch (Exception ex)
{
    MessageBox.Show(ex.Message);
}
}

```

The code creates an instance of `MichiganResortsEntities` class. The LINQ query extracts the country values from the Resorts table and placed it in the variable, `query`. Note that the `Distinct()` method in LINQ extracts only unique values.

The code in the `try` block populates the Country Box control with the values from `query` variable. The statement to populate the combo box is placed within a `try` block to handle any runtime exceptions that may occur. Note that although the variable `query` is defined in the previous line, the LINQ query is executed only when it is used in the assignment statement in the `try` block. This is called deferred execution.

**5. Build and execute the application. The output is displayed as shown in figure 21.13.**

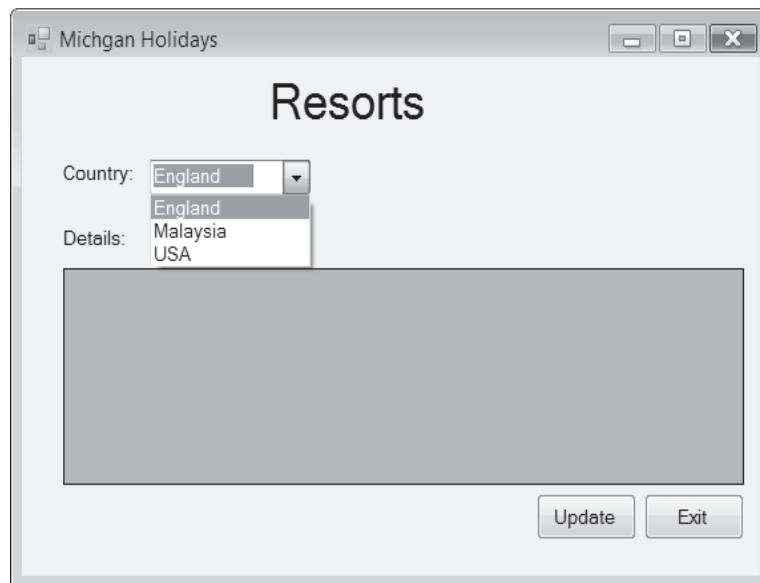


Figure 21.13: Populated Values in Combo Box

Here you can see the values from the country column are displayed in the combo box.

## Displaying data

To display the details of the resort in the data grid view, perform the following steps:

1. Double-click the Country Box control. This generates the `SelectedIndexChanged` event handler for the Country Box control, `cboCountry`.
2. Add code in `cboCountry _ SelectedIndexChanged` event handler method.

```

/// <summary>
/// Displays the resort details in Data Grid View
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void cboCountry_SelectedIndexChanged(object sender, EventArgs e)
{
    try
    {
        string countryValue = this.cboCountry.SelectedItem.ToString();
        var query = from c in objResorts.Resorts
                    where c.Country == countryValue
                    select c;
        dgvDetails.DataSource = query;
        dgvDetails.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCells);
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

```

The value selected from the combo box control is stored in the string variable, `countryValue`. Then, a LINQ query is given to extract those records from the table whose country matches with the variable, `countryValue`. The data source property of the data grid view control assigned with the variable, `query`.

3. Build and execute the application. The output is displayed as shown in figure 21.14.



Figure 21.14: DataSource Output

The output shows the details of the resort with **ResortID**, 40, from the country, England.

4. In the Country box control, select the value USA. This displays all the records related to the country, USA.
5. Adjust each row of the data grid view control to display the image completely. The output is displayed as shown in figure 21.15.



Figure 21.15: DataGridView Output

You can add and modify data in the table through the data grid view. To save changes in the database, perform the following steps:

**1. Add code in the Click event handler of Update command button.**

```
/// <summary>
/// Save object changes to the database
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btnUpdate_Click(object sender, EventArgs e)
{
    try
    {
        objResorts.SaveChanges();

        MessageBox.Show("Changes saved to the database", "Save");
        this.Refresh();
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}
```

The code saves changes to the database and then, refreshes the form.

2. Build and execute the application.
3. In the Country box control, select the value USA. You can see from figure 21.15 that, for the ResortID 10, the rate specified is 500.
4. Select the Rate cell for ResortID, 10. Change the value to 575 and click Update. The message will be displayed as shown in figure 21.16.

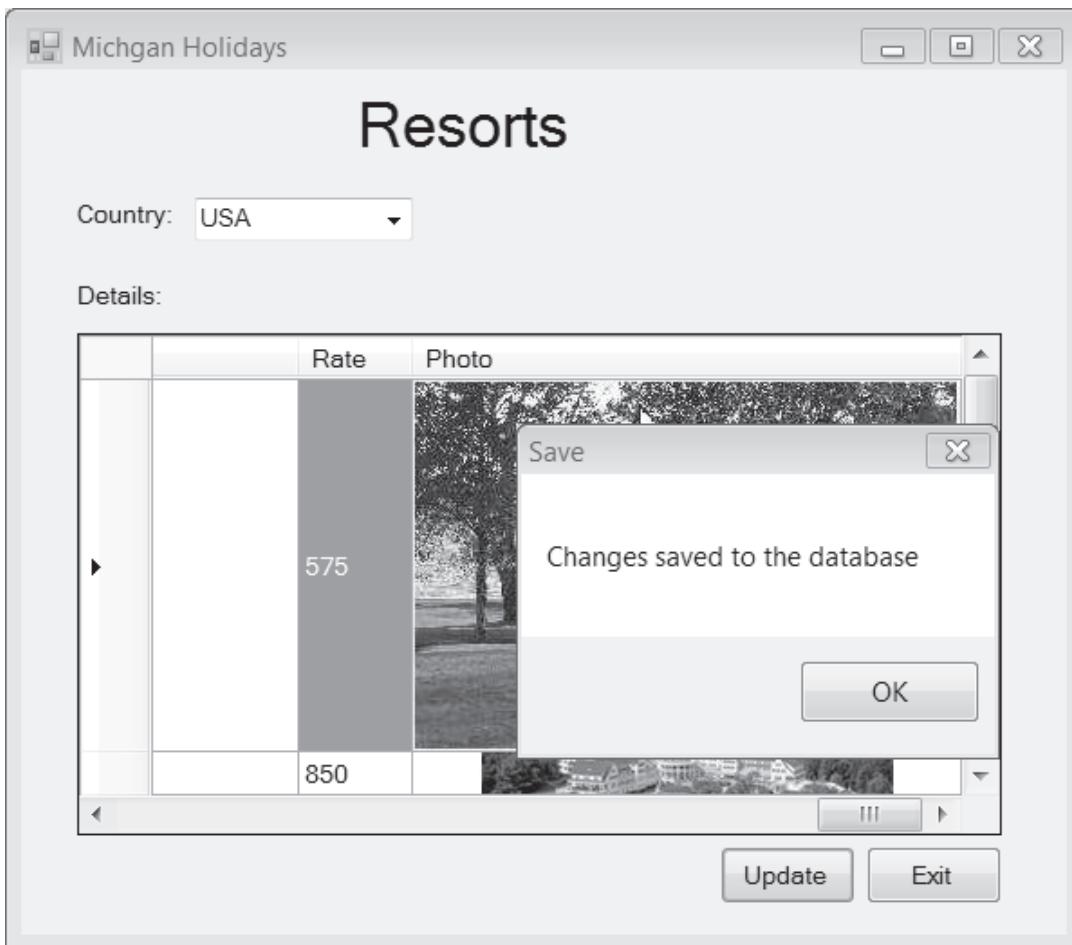


Figure 21.16: Updated Records Output

5. Add code in the Click event handler of Exit command button.

```
/// <summary>
/// Closes the form
/// </summary>
/// <param name="sender"></param>
```

```

/// <param name="e"></param>
private void btnExit_Click(object sender, EventArgs e)
{
    objResorts.Dispose();
    this.Close();
}

```

The code first disposes the object context and then, the application is exited.

### Exercise

Michigan Hotels is a worldwide chain of hotels owned by Michigan group, one of USA's largest business conglomerates. They own and operate 51 hotels and 9 resorts and spas. The hotels spanned in 31 destinations in 9 countries and employ over 10000 people.

Besides USA, Michigan Hotels are located in the United Kingdom, Africa, Maldives, Mauritius, China, India, Germany, and China.

Recently the management has decided to upgrade their hotel booking application to use the new features of .NET framework and Visual Studio 2008.

The development team in Michigan wants to demonstrate a part of the modified application. Consider that you are one of the programmers in the team and are assigned the following tasks:

- Create a class that stores all the booking details
- Create and construct XML from the class

### Solution:

To implement the solution, you will use LINQ to SQL.

#### Create the Customer class and define the properties

1. Create new Windows Forms application named **MicghanHotels**.
2. Rename **Form1.cs** to **frmBookingDetails.cs** and change the **Text** property of the form to **Booking Details**.
3. Add a new class named **Customer.cs**.
4. Define the properties in **Customer class**.

```

/// <summary>
/// Class that defines the booking details.
/// </summary>

```

```
class Customer
{
    /// <summary>
    /// Property to store the customer name.
    /// </summary>
    public string CustomerName { get; set; }

    /// <summary>
    /// Property to store the booking date.
    /// </summary>
    public string BookingDate { get; set; }

    /// <summary>
    /// Property to store the room details.
    /// </summary>
    public string Room { get; set; }

    /// <summary>
    /// Property to store the room rate.
    /// </summary>
    public int Rate { get; set; }

    /// <summary>
    /// Property to store the discount details.
    /// </summary>
    public int Discount { get; set; }
}
```

The properties are defined to store the booking details of each customer.

5. Design the user interface as shown in figure 21.17.

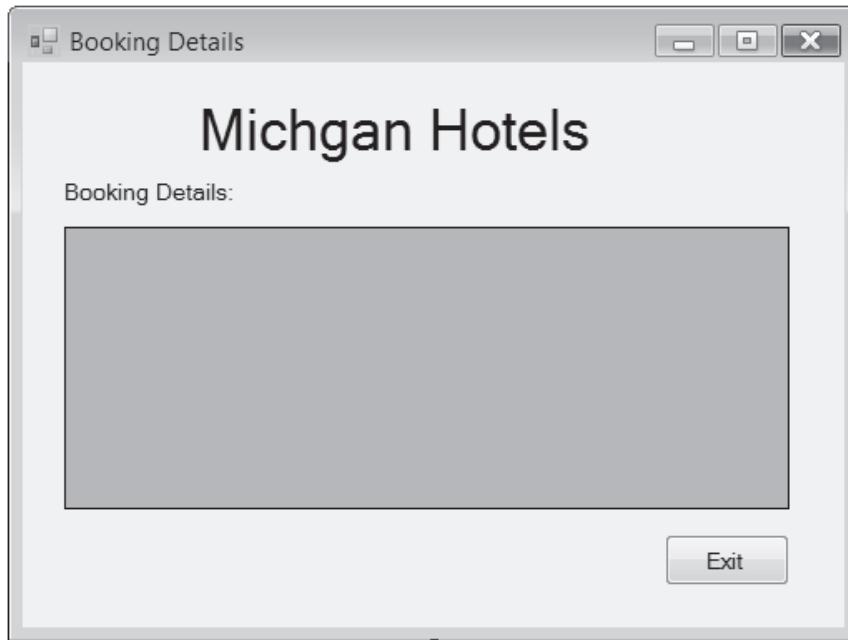


Figure 21.17: Form Design

The details of the controls are listed in table 21.2.

Control	Property	Value
Label	Name	lblHeading
	Text	Michigan Hotels
Label	Name	lblBookingDetails
	Text	Booking Details:
DataGridView	Name	dgvBookings
Button	Name	btnExit
	Text	Exit

Table 21.2: Controls

6. Add the following namespace in the `frmBookingDetails` class.

```
using System.Xml.Linq;
```

This namespace contains classes that can be used for LINQ to XML.

7. Create a method named `BookingDetails()` that returns a generic `List` collection.

```
/// <summary>
/// Stores the customer details.
/// </summary>
```

```

/// </summary>
/// <returns>List<Customer></returns>
private List<Customer> BookingDetails()
{
    List<Customer> customerList = new List<Customer>()
    {
        new Customer (){ CustomerName="David Blake",
                         BookingDate="10-July-2010",
                         Room="304, Deluxe Suite",
                         Rate=1200,
                         Discount=12 },
        new Customer (){ CustomerName="Susan Jones",
                         BookingDate="11-July-2010",
                         Room="101, A/C Express Suite",
                         Rate=630,
                         Discount=6 },
        new Customer (){ CustomerName="John Cleetes",
                         BookingDate="15-August-2010",
                         Room="601, Diamond Deluxe Suite",
                         Rate=2305,
                         Discount=23 }
    };
    return customerList;
}

```

The method stores the details of three customers with their name, booking date, room details, rate, and discount in the generic list collection, `customerList`. After storing the details, the method returns the collection variable.

- 8. Create a generic list collection variable in the Load event handler method of `frmBookingDetails` form and call the method `BookingDetails()`.**

```
// <summary>
```

```

/// Constructs the XML file using LINQ.
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void frmBookingDetails_Load(object sender, EventArgs e)
{
    List<Customer> customerList = BookingDetails();
}

```

A generic list collection named `customerList` is created. The call to `BookingDetails()` method stores the details of the three customers in `customerList`.

**9. To construct and save data to an XML tree, add the following code in the Load event handler method of frmBookingDetails form.**

```

 XElement customers = new XElement("MichiganHotels",
    from c in customerList
    select
        new XElement("CustomerBookings",
            new XElement("CustomerName", c.CustomerName),
            new XElement("BookingDate", c.BookingDate),
            new XElement("Room", c.Room),
            new XElement("Rate", c.Rate),
            new XElement("Discount", c.Discount)));

```

```

customers.Save("Booking.xml");

```

Here the XML tree is constructed using the `XElement` class which is a member of `System.Xml.Linq` namespace. First, the root node, `MichiganHotels` is created, under that the node named `CustomerBookings` is created. Then, the child members named, `CustomerName`, `BookingDate`, `Room`, `Rate`, and `Discount` are added into it. The values for each child node are taken from the generic collection list variable, `customerList`.

The `Save()` method of `XElement` class saves the XML tree in `Booking.xml` file.

Figure 21.18 shows the XML tree structure based on the content of Booking.xml file.

```

<?xml version="1.0" encoding="utf-8" ?>
- <MichiganHotels>
  - <CustomerBookings>
    <CustomerName>David Blake</CustomerName>
    <BookingDate>10-July-2010</BookingDate>
    <Room>304, Deluxe Suite</Room>
    <Rate>1200</Rate>
    <Discount>12</Discount>
  </CustomerBookings>
  - <CustomerBookings>
    <CustomerName>Susan Jones</CustomerName>
    <BookingDate>11-July-2010</BookingDate>
    <Room>101, A/C Express Suite</Room>
    <Rate>630</Rate>
    <Discount>6</Discount>
  </CustomerBookings>
  - <CustomerBookings>
    <CustomerName>John Cleetes</CustomerName>
    <BookingDate>15-August-2010</BookingDate>
    <Room>601, Diamond Deluxe Suite</Room>
    <Rate>2305</Rate>
    <Discount>23</Discount>
  </CustomerBookings>
</MichiganHotels>

```

Figure 21.18: XML Tree

- To display the XML data in the data grid view control, add the remaining code in the `frmBookingDetails _ Load()` method.

```

DataSet ds = new DataSet();
ds.ReadXml("Booking.xml");

dgvBookings.DataSource = ds;
dgvBookings.DataMember = "CustomerBookings";
dgvBookings.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCells);

```

A `DataSet` named `ds` is created. The `ReadXml()` method of the `DataSet` reads the data from the data from the XML file.

Then, the data source of the data grid view control is set to the `DataSet` and the data member is set as `CustomerBookings` node in the XML file.

11. Build and execute the application. The output of the application is displayed in figure 21.19.

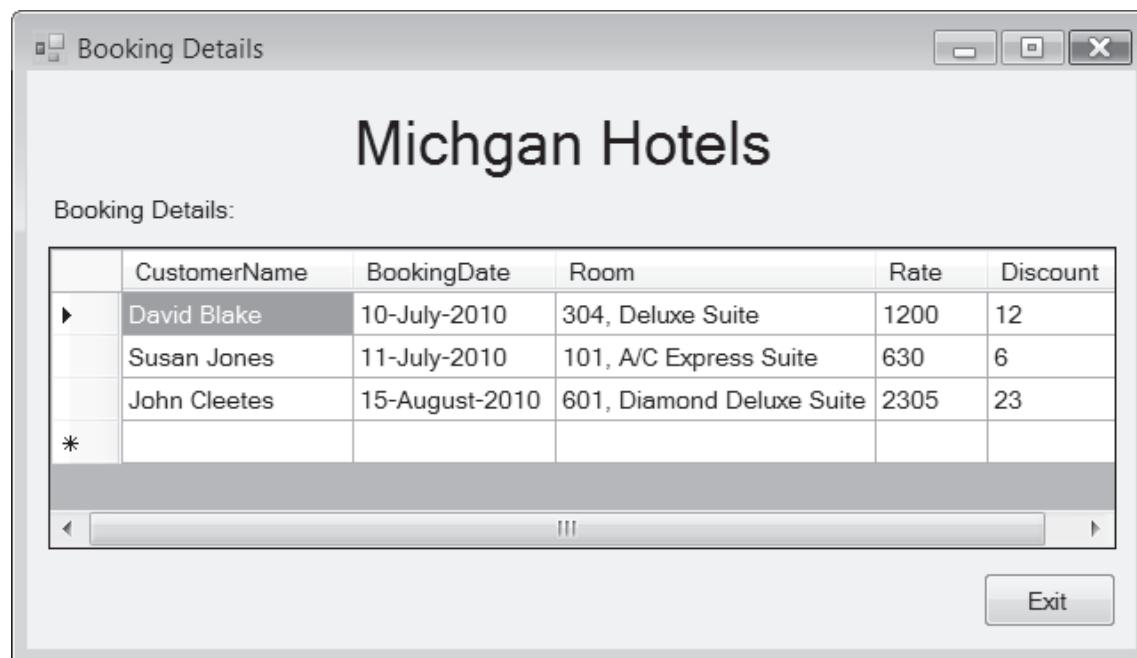


Figure 21.19: Booking Details Output

As displayed in the output, all three booking details from the XML file are populated in the data grid view control.

12. To close the form, add code in the Click event handler of Exit command button.

```

/// <summary>
/// Closes the form
/// </summary>
/// <param name="sender"></param>
/// <param name="e"></param>
private void btnExit_Click(object sender, EventArgs e)
{
    this.Close();
}

```

## Part II – 60 Minutes

1. Modify the application given in Part I Exercise to perform the following tasks:
  - Modify the database to create a new table named Country with CountryID, Country, and Location columns. Link the two tables, Resorts and Country based on CountryID column.
  - Modify the application to create a master-detail relationship. When the user selects a country, it should display all the records related to that country.

**Hints:**

- Create the new table.
  - Create an entity data model with Resorts and Country tables. Make proper associations between two tables in the ADO.NET Entity Data Model Designer Window.
  - Write appropriate LINQ queries to extract records from two tables.
2. Modify the application given in Part I Exercise to do the following:
    - Add text boxes to display the values of the current row in the data grid view control.
    - Add buttons for Add, Modify, and Delete values in XML tree. After modifying the XML tree, display the current values in the data grid view control.

**Hints:**

Use the `Add()`, `Remove()`, and `ReplaceNodes()` methods of `XElement` to add, remove, and modify values in the XML tree.

## *Try it Yourself*

1. Create an application for a Security Service company located in London, United Kingdom. The application should fulfill the following requirements:
    - A database to store the details of all security guards in the company
    - A Windows Forms application with two projects, one for data layer and the other for displaying the details
    - The data layer project should use LINQ to SQL Classes to connect to the database
    - The second project contains two forms. The first form is the user interface for adding, modifying, and deleting records in the database. Using the second form, a user can search for security guards based on the city or region. The result will be displayed in a data grid view control

Figure 21.20 shows the output of a search criteria based on city.

Search

# Micghan Security Services

City	<input type="text" value="London"/>				
Region	<input type="text"/>				
<input type="button" value="Search"/>					
	GuardID	FirstName	LastName	Address	City
▶	DBAL	David	Blake	304, Londo...	London
	JCOL	John	Cleetus	120 Hanove...	London
*					

**Figure 21.20:** Search Output

## Try it Yourself

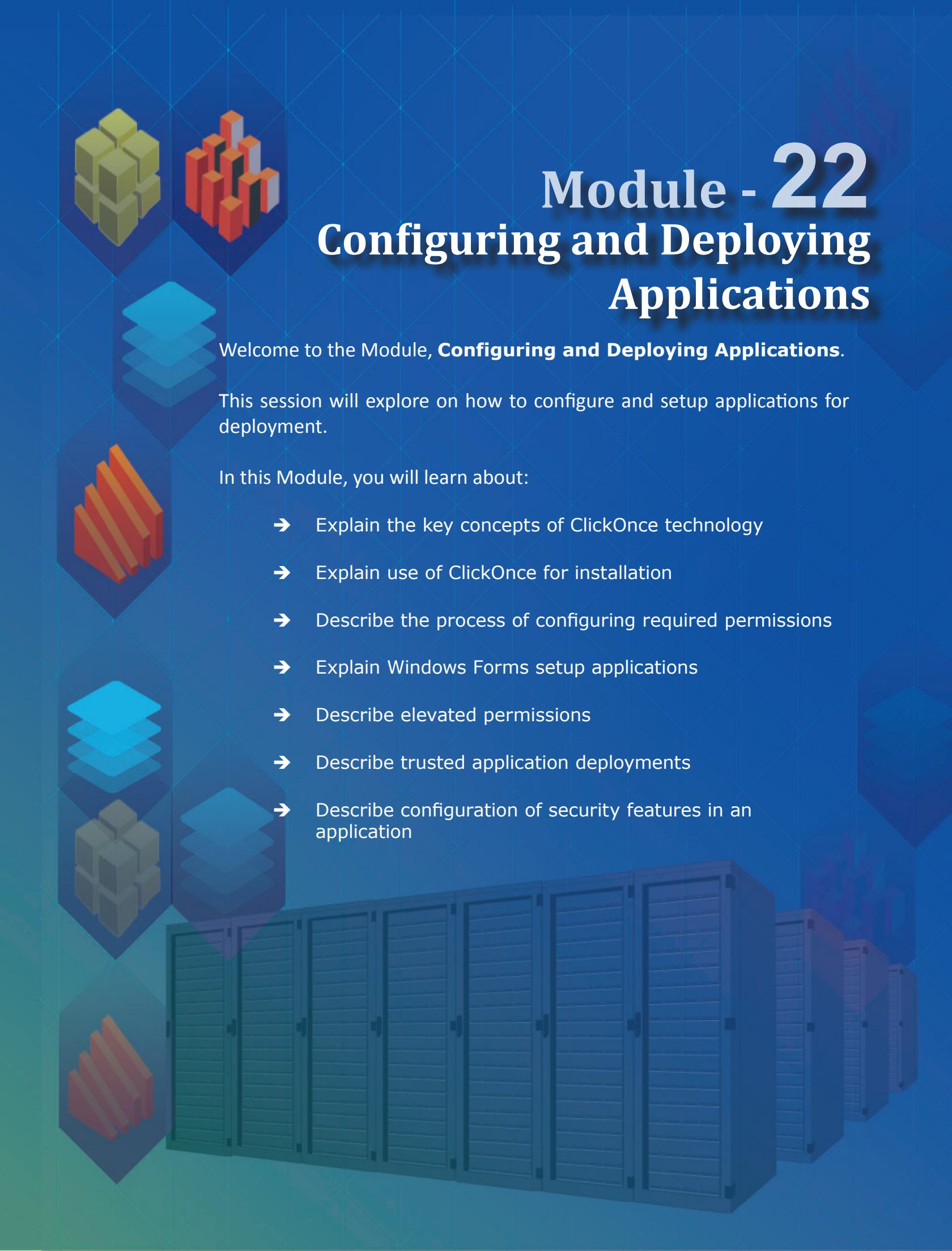
Figure 21.21 shows the form to add, modify, and delete records from the database.

The screenshot shows a Windows application window titled "Guard". The main title bar says "Micghan Security Services". Inside the window, there are several text input fields and one multi-line text area. The fields are labeled as follows: "Guard ID:" with value "RLTE", "First Name:" with value "Romario", "Last Name:" with value "Leonardo", "Address:" with value "123, Swing Street, Highway Road, Next to Blue Moon Hotel", "City:" with value "Manchester", "Region:" with value "ARMY", "Postal Code:" with value "011-89765", and "Phone:" with value "10-78756698". At the bottom of the window are three buttons: "Save", "Delete", and "Exit".

Figure 21.21: Add/Modify/Delete Records



Visit the  
**Frequently Asked Questions**  
section @



# Module - 22

## Configuring and Deploying Applications

Welcome to the Module, **Configuring and Deploying Applications**.

This session will explore on how to configure and setup applications for deployment.

In this Module, you will learn about:

- Explain the key concepts of ClickOnce technology
- Explain use of ClickOnce for installation
- Describe the process of configuring required permissions
- Explain Windows Forms setup applications
- Describe elevated permissions
- Describe trusted application deployments
- Describe configuration of security features in an application

## 22.1 Introduction

Application design and development require a lot of planning and effort. However, a developer's work does not end as soon as the application is developed, because the next stage which is, configuring and deploying, also requires good amount of planning and effort. Microsoft facilitated deployment of Windows-based applications through a new technology called ClickOnce that was introduced in Visual Studio 2005.

In Visual Studio 2008, both ClickOnce and Windows Installer deployment have been enhanced. This session explores the ClickOnce technology, its enhancements, setup applications, and configuring applications for deployment.

## 22.2 Introduction to ClickOnce Deployment

For years, Windows-based applications were not as easy to deploy as Web based applications, where you just had to click a link to begin deployment. A technology was needed to enable easier deployment and overcome issues faced with conventional deployment models. Some of the issues were: difficulty in performing updates to deployed applications, need for administrator permissions to install applications, and so on. As a solution for all this, a deployment technology was introduced by Microsoft called as ClickOnce.

ClickOnce is a deployment technology that facilitates creation of self-updating Windows-based applications which can be installed and executed with least user interaction.

ClickOnce can be used to publish Windows Forms, Windows Presentation Foundation (WPF), Console-based applications, and so forth. Any Windows Forms (.exe file), Windows Presentation Foundation (.xbap file), console application (.exe file), or Office (.dll library) solution that is published using ClickOnce technology is termed as a ClickOnce application.

The key features of ClickOnce applications are as follows:

- Each ClickOnce application is self-contained and isolated from the other. This means that one ClickOnce application will not interfere with or break another application.
- ClickOnce applications are self-updating. This means that they check for latest versions as soon as they are available and perform the updates automatically.
- ClickOnce-deployed applications do not require administrator privileges for installation.

### 22.3 ClickOnce Deployment

Figure 22.1 shows an overview of the ClickOnce deployment process.

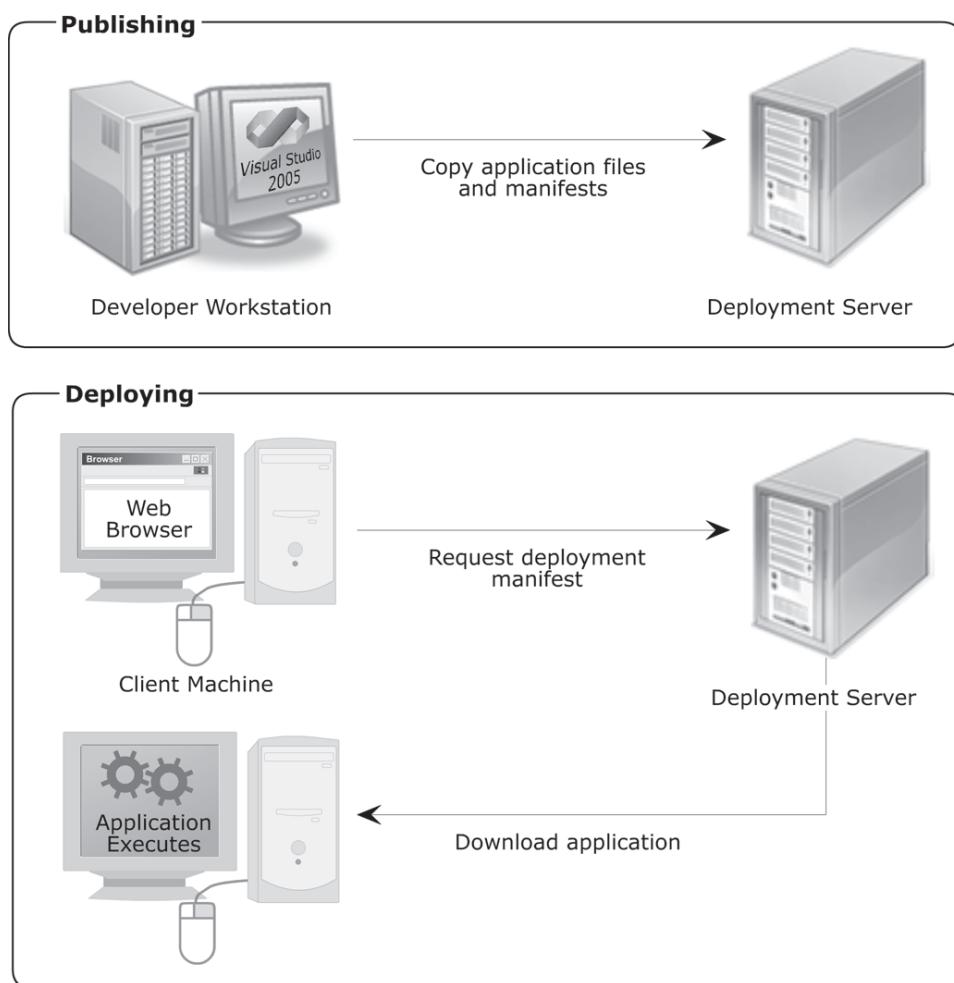


Figure 22.1: ClickOnce Process

When an application is built and successfully tested, it can be published to the deployment server. This is done by copying all the application files into a file server or a folder on the Web server in case of Web deployment. The application and deployment manifest files which contain deployment information for the ClickOnce runtime are also copied.

After the files are published to the deployment server, you provide a link to the deployment manifest to the user as a hyperlink or shortcut. On clicking that link, the .NET runtime downloads and executes the application on the user's desktop, thus, implementing a ClickOnce deployment.

The following sections describe various types of installations that can be done using ClickOnce.

### 22.3.1 Installing Windows Forms applications with ClickOnce

ClickOnce applications published on a Web server can be installed on a user's machine irrespective of the machine being online or offline. A ClickOnce application is published using the **Publish** wizard of Microsoft Visual Studio 2008. The wizard allows you to set the required publishing properties of the application.

Consider the `TabControl` example created in an earlier session. The steps to publish this as a ClickOnce application to a Web server are listed as follows:

1. Launch the Publish Wizard in Visual Studio 2008 IDE by selecting **Project→Properties, Publish tab** and then, Publish Wizard button as shown in figure 22.2.

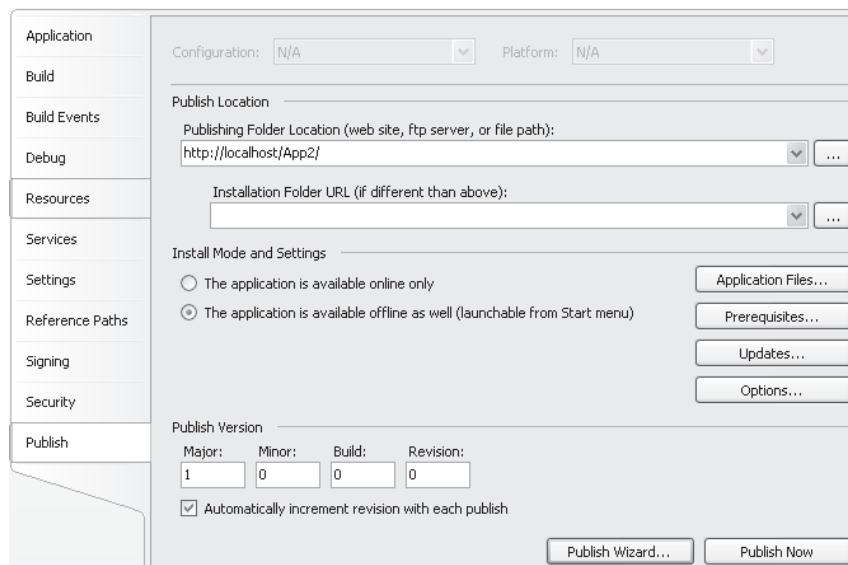


Figure 22.2: Publish tab in the Project Properties window

This displays the page as shown in figure 22.3.

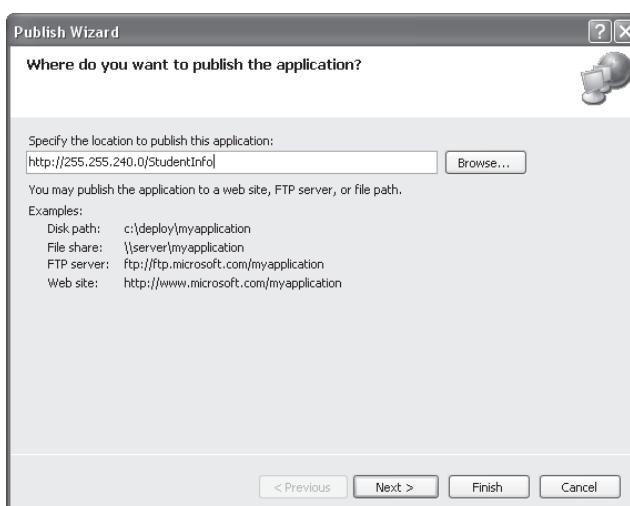


Figure 22.3: Step 1 of Publish Wizard

2. In the **Where do you want to publish the application?** page shown in figure 22.3, enter a valid Universal Resource Locator (URL) for the location.
3. Click **Next**.

This displays the next page of the Publish Wizard as shown in figure 22.4. This page changes its look depending on what you enter in the location on figure 22.3. Assuming that you enter a Web site URL, the steps to proceed further are described.

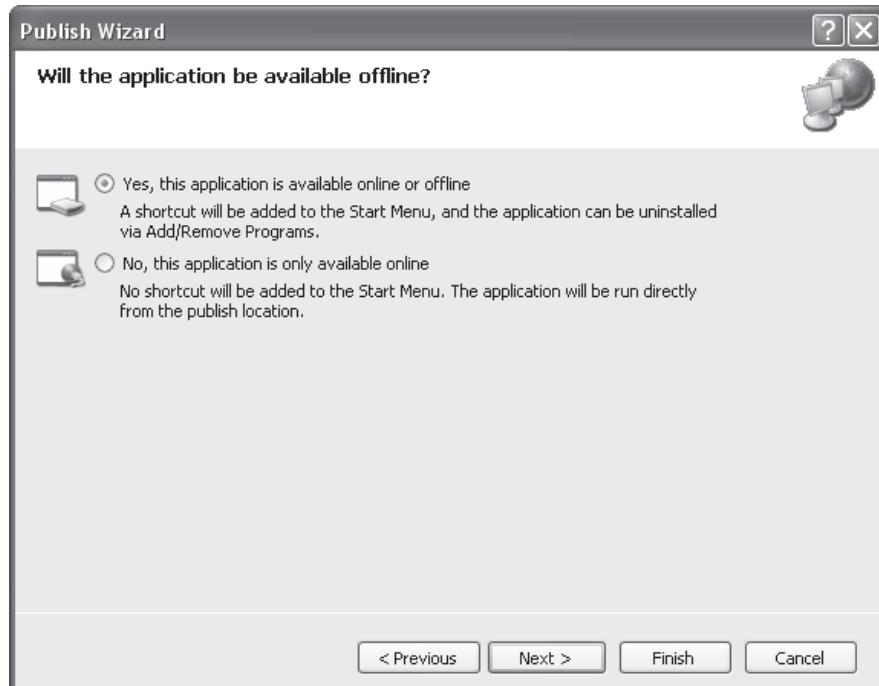


Figure 22.4: Step 2 of Publish Wizard

4. In the **Will the application be available offline?** page, select the suitable option among the following:
  - Click the **Yes, this application will be available online or offline** option if you want to run the application even when the user is offline. A shortcut will be created on the Start menu to install the application.
  - Click the **No, this application is only available online** option if you want to run the application only if the user is connected to the network. There will not be any shortcut on the Start menu.

5. Click **Next**. The final page, **Ready to Publish**, is displayed as shown in figure 22.5.

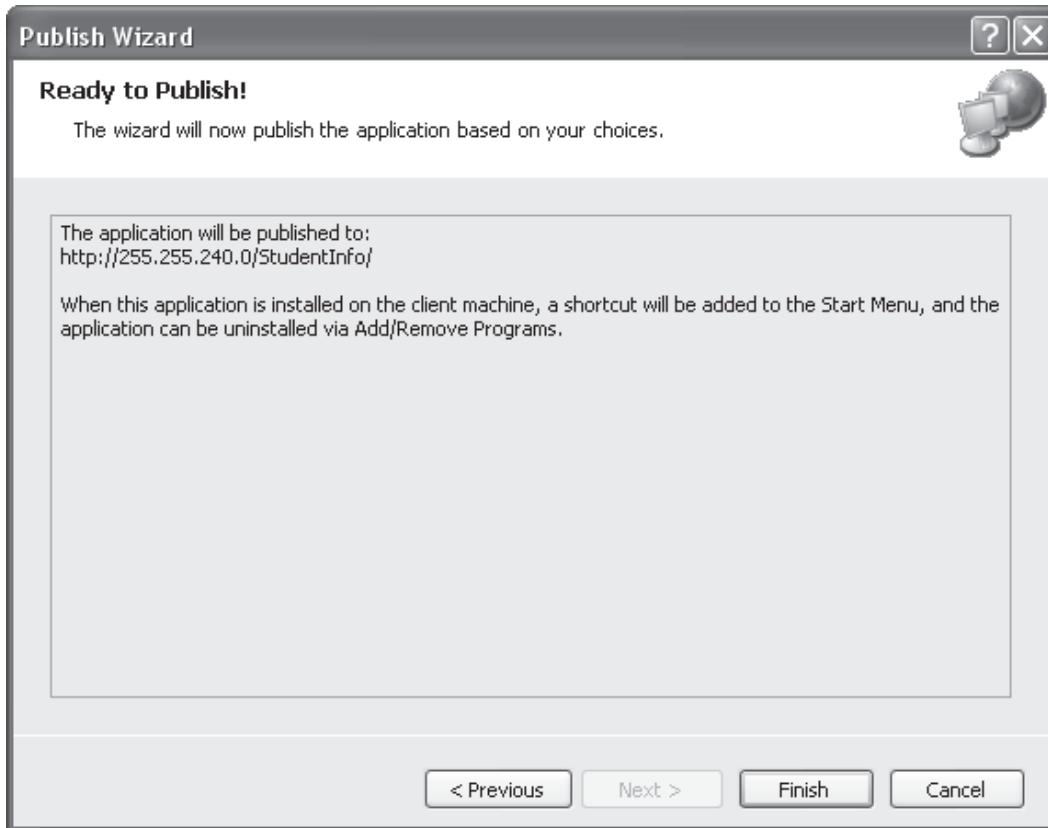


Figure 22.5: Step 3 of Publish Wizard

6. Click **Finish** to publish the application.

Similarly, ClickOnce applications can be published on a file share, FTP path, and so on. The pages shown in each stage of the Publish Wizard will change depending on where you decide to publish the application.

One of the new enhancements in Visual Studio 2008 is that the **Publish Options** dialog box has been modified. This dialog box is accessed by clicking **Options** button on the **Project Properties** page (shown earlier in figure 22.2).

The information supplied through this dialog box is now characterized into four separate sections that can be accessed through a navigation pane.

Figure 22.6 shows the enhanced **Publish Options** dialog box.

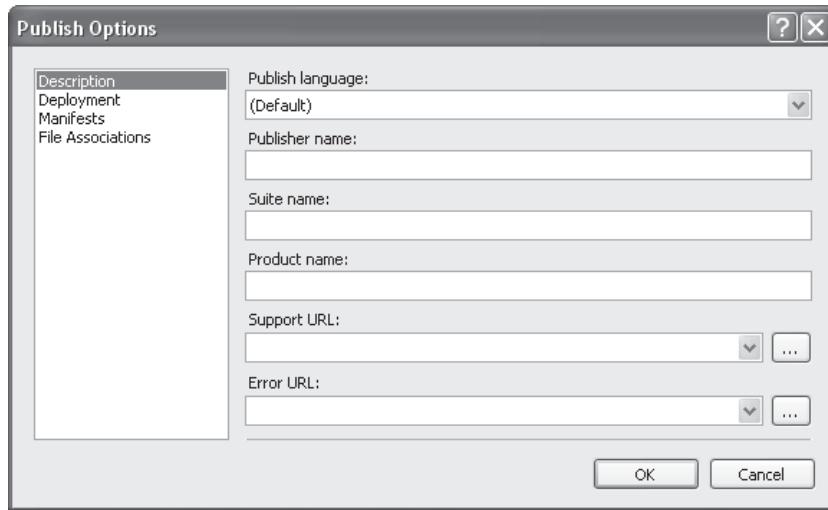


Figure 22.6: Publish Options

### *22.3.2 Installing WPF browser applications by using ClickOnce*

One of the enhancements made to ClickOnce in Visual Studio 2008 is support for deploying WPF browser applications.

WPF browser applications are online-only applications. They are not permanently installed on the machine but they are cached to enable frequent executions to be faster.

The procedure to install WPF browser applications by using ClickOnce is similar to that described for Windows Forms earlier except that the **Will the application be available offline?** page is not seen here. This is because such applications are not available offline and are published in online-only mode.

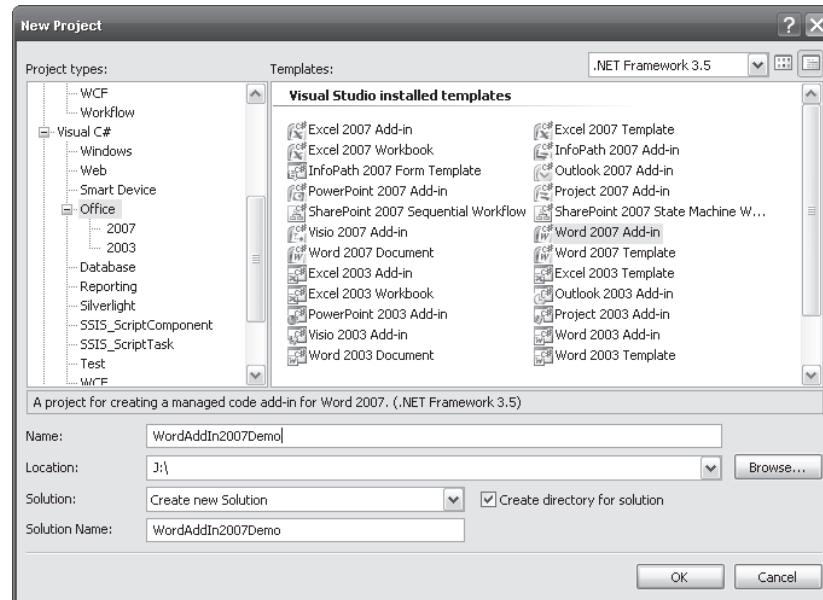
After you provide the publishing location of your application in Step 1 of the **Publish Wizard**, you will see the page similar to figure 22.5 shown earlier.

### *22.3.3 Installing a VSTO application using ClickOnce*

Visual Studio Tools for Office (VSTO) is a Microsoft technology that enables you to extend Microsoft Office applications using .NET languages such as C#. It comprises development tools and a runtime. The development tools include Visual Studio add-ins (project templates) that enable VSTO applications and Office add-ins to be developed using the Visual Studio IDE.

In Visual Studio 2008, the Visual Studio Tools for Office are integrated into the IDE as project templates for Add-Ins.

You can create a new VSTO project by using any of the templates shown in figure 22.7.

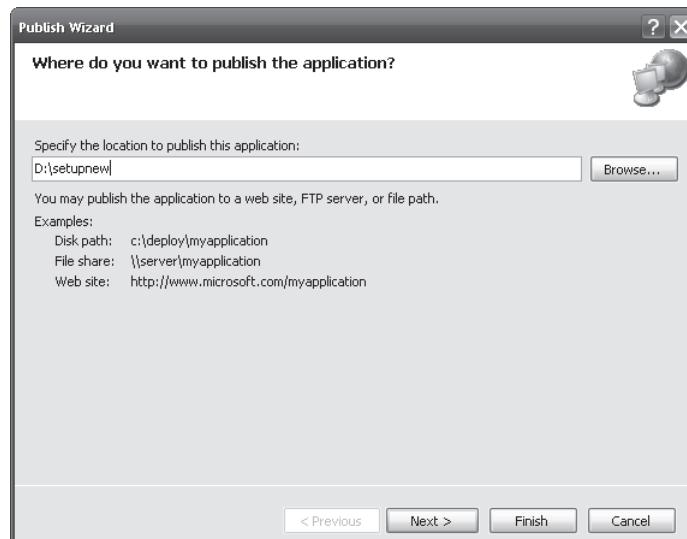


**Figure 22.7: Creating a VSTO project**

Installing such a project using the ClickOnce technology is almost similar to the installations described earlier.

Note that you require the appropriate version of VSTO runtime on your machine and primary interop assemblies for the relevant Microsoft Office version. For example, in the current scenario, you need VSTO runtime for Visual Studio 2008 and Microsoft Office 2003 primary interop assemblies. Both are available for free download from the Microsoft Web site.

When you right-click the Project name in **Solution Explorer** and click **Publish**, the **Publish Wizard** appears as usual. The publish location is specified as shown in figure 22.8.



**Figure 22.8: Specifying Publish Location**

After the publish location has been specified, no other details have to be entered. As seen in figure 22.9, the **Publish Wizard** indicates that the project is ready to publish.

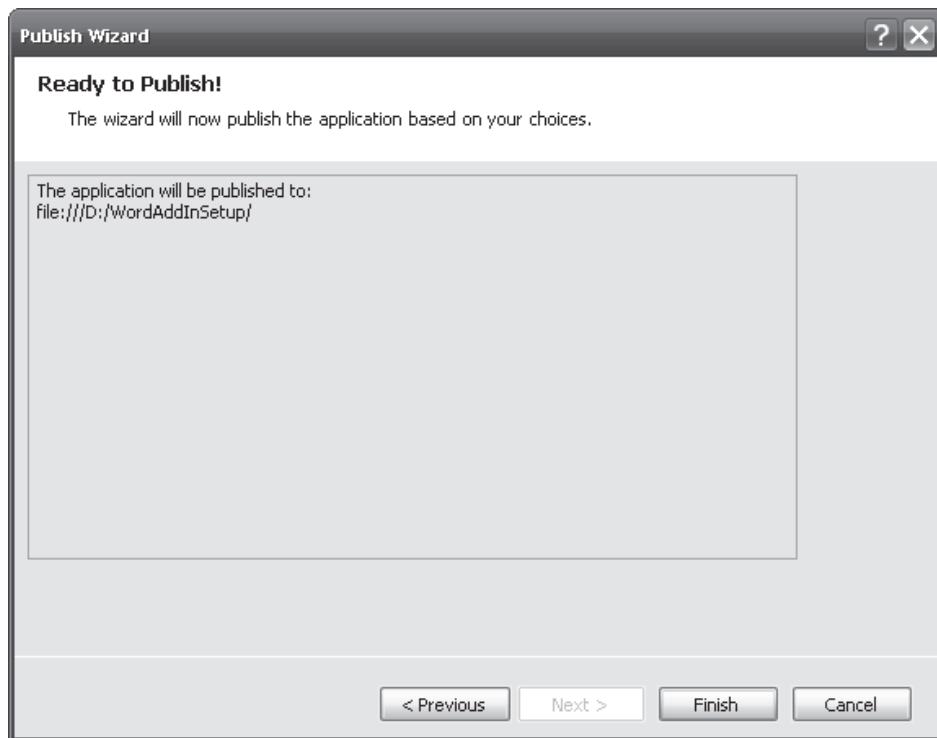


Figure 22.9: Publish Wizard showing success

The outcome of publishing will comprise manifest files, a Setup file, and optionally a folder containing redistributable files that can install primary interop assemblies required for Word 2007. These are shown in figure 22.10.

Name	Type	Date Modified
Application Files	File Folder	6/17/2010 12:48 PM
Office2007PIARedist	File Folder	6/17/2010 12:48 PM
setup	Application	6/17/2010 12:47 PM
WordAddIn2007Demo	VSTO Deployment Manifest	6/17/2010 12:47 PM

Figure 22.10: Deployment files

Using these files, the application can be executed on the client computer to install the application. Copy the files to the client machine. The client or user need to double-click the setup file to begin the installation.

The process begins with installing Office customization information as shown in figure 22.11.

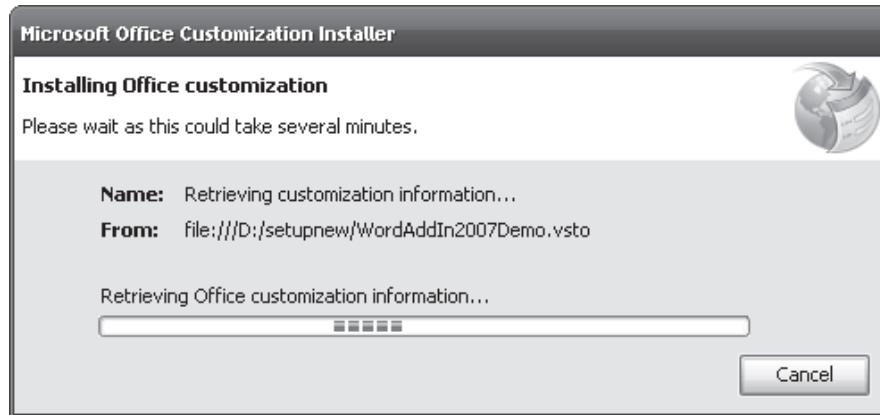


Figure 22.11: Installing Office customization

When the Office customization information is ready, the user will be prompted about publisher verification as shown in figure 22.12.

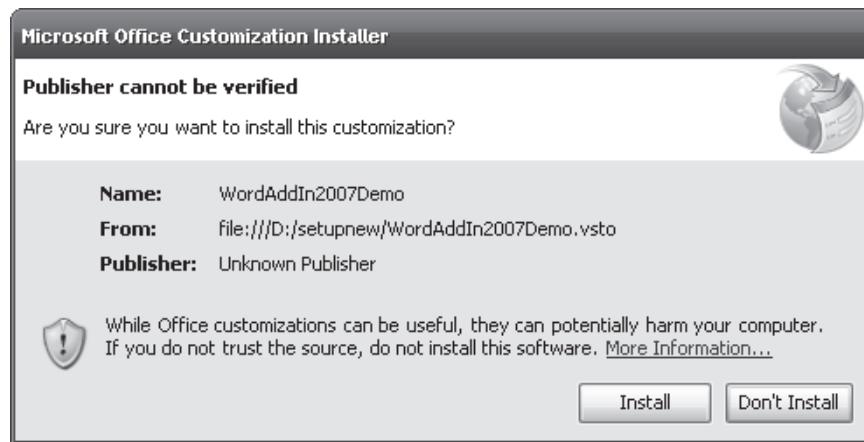


Figure 22.12: Prompt for User Permission

If the user clicks Install, the installation of the VSTO application begins. In this manner, a VSTO application can be deployed and installed using ClickOnce.

### 22.3.4 Configuring for Automatic Updates

The **Updates** button on the Publish properties page enables you to configure ClickOnce applications to automatically check for updates.

Figure 22.13 shows this button highlighted. Assume that an application, App2, is being configured.

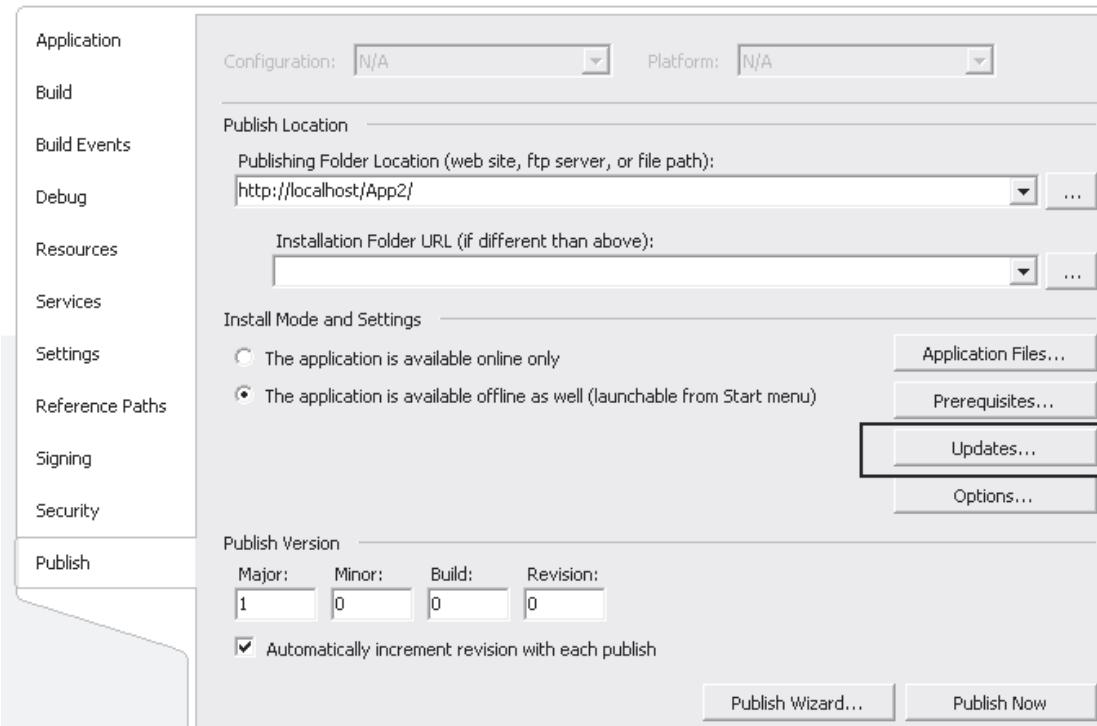


Figure 22.13: Publish tab in Project Properties

Clicking the **Updates** button displays the **Application Updates** dialog box.

You need to select the check box, **The application should check for updates** so that the application can check for updates. This will enable the other options in the dialog box as seen in figure 22.14.

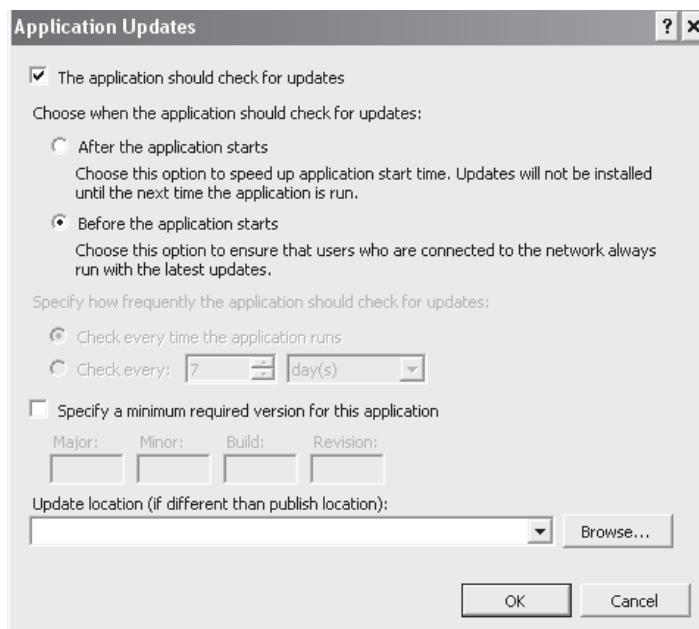


Figure 22.14: Application Updates dialog box

You can choose when the application should check for updates. You can select either **After the application starts** or **Before the application starts**.

Selecting **Before the application starts** will cause the application will check for new updates whenever the application is started. While this will keep the user up to date with the most recent version of the application, it also slows down performance by taking more time at startup.

Selecting **After the application starts** will cause the application to check for updates whenever it is run, or at a designated time interval. This interval is specified under the heading **Specify how frequently the application should check for updates**.

If necessary, you may also select the check box, **Specify a minimum required version for this application** and provide the version information.

To specify a different location for updates so that the updates will be hosted in a location other than the install location, you can provide the location in the **Update location** text box.

### 22.3.5 Configuring Windows Vista User Account Control (UAC)

Windows Vista supports a new feature and security infrastructure called UAC. UAC treats the users belonging to the local administrator groups as if they had no administrative privileges. Only when an administrator grants or elevates permissions, software applications will be allowed to run. This ensures that only trusted applications are executed, thus, keeping out the malwares and malicious programs. While developing applications for Windows Vista, you must take care to see that they are UAC-aware.

You can make your application UAC-aware for Windows Vista by embedding a manifest file into the executable or code library with the `requestedExecutionLevel` element.

An example manifest file is shown here:

**Example:**

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<assembly xmlns="urn:schemas-microsoft-com:asm.v1" manifestVersion="1.0">
  <trustInfo xmlns="urn:schemas-microsoft-com:asm.v3">
    <security>
      <requestedPrivileges>
        <requestedExecutionLevel level="asInvoker" uiAccess="false"/>
      </requestedPrivileges>
    </security>
  </trustInfo>
</assembly>
```

The techniques for adding the manifest differ between technologies and languages. For your Windows Forms applications, you can use the Microsoft **MT.exe** tool. This tool generates signed files and catalogs.

In the example shown here, a manifest file saved as TestApp.exe.manifest is added to an application named TestApp.exe.

```
mt.exe -manifest "TestApp.exe.manifest" -outputresource:"TestApp.exe"
```

The last parameter given to this tool changes depending on whether the application is a library (dll) or an executable (exe).

Hence, whenever you build UAC-aware applications for Windows Vista, an embedded manifest is generated. ClickOnce deployment architecture relies on two manifest files that are XML-based: an application manifest and a deployment manifest. The files contain information about the ClickOnce applications such as the install source of ClickOnce applications, how and when updates will be performed.

ClickOnce projects need external deployment manifests. The embedded manifest that is created for UAC-aware applications will not be useful for a ClickOnce project.

Hence, for ClickOnce projects that are UAC-aware, Visual Studio will generate an external manifest file that will contain the UAC data. Information from a file called app.manifest will be used to generate external UAC manifest information.

### 22.3.6 Configuring Required Permissions for a ClickOnce Application

ClickOnce applications executed from the Internet, by default, run in the Internet security zone whereas ClickOnce applications executed from a file share run in the intranet security zone.

For applications that require additional permissions to run, you can use the **Security** tab of the **Project Properties** window to configure the same. This is shown in figure 22.15.

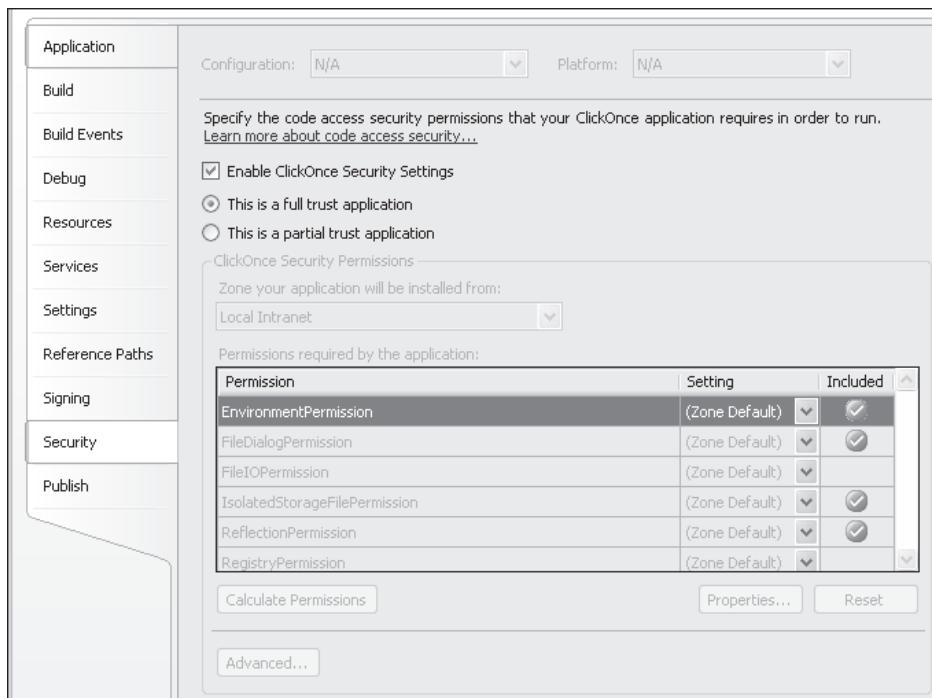


Figure 22.15: Security tab

There are two approaches to determine the permissions through the **Security** tab of the **Project Properties** window. They are as follows:

- ➔ Manually select permissions to be granted to the application
  1. Select the check box, **Enable ClickOnce Security Settings**
  2. Choose the option button, **This is a partial trust application**
  3. In the **Permissions required by the application** table, find the permission of interest
  4. In the **Setting** column, choose **Include** to include the permission as required or choose **Exclude** to not require the permission
- ➔ Calculate the permissions needed by the application and then, configure the application to request those permissions
  1. Select the check box, **Enable ClickOnce Security Settings**
  2. Choose the option button, This is a partial trust application
  3. Click the button, **Calculate Permissions**

The permissions required by an application are added to the permissions that will be requested by the application upon installation.

### 22.3.7 Installing a ClickOnce Application on a Client Computer

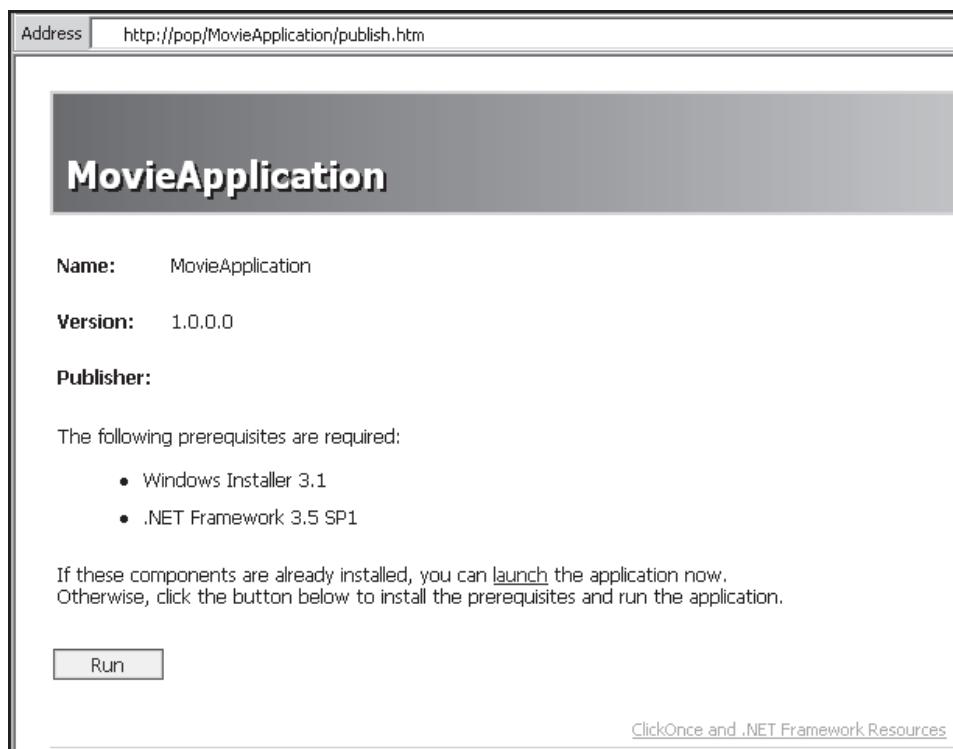
You can install a ClickOnce application on a client computer in two ways.

- ➔ Installing a ClickOnce application from a Web site:

1. Launch the Publish.htm Web page from the installation URL.

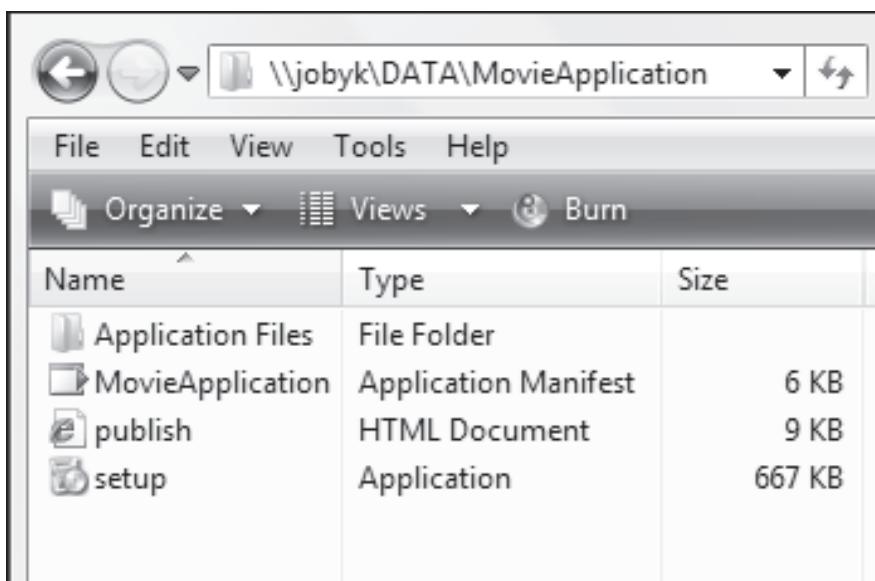
For example, assume that your application named MovieApplication was published to <http://localhost/MovieApplication>, and your machine name was named **pop**, the Publish.htm file would be located at <http://pop/MovieApplication/publish.htm>.

Figure 22.16 shows an example.



**Figure 22.16: Installing from a Web site**

2. Click **Run** and follow the steps (if any) in the Install wizard.
- To install a ClickOnce application from a file share as follow:
1. Open the file share. An example of this is shown in figure 22.17.



**Figure 22.17: Installing from a file share**

2. Double-click **Setup** and follow the steps (if any) in the Install wizard.

## 22.4 Windows Forms Setup applications

ClickOnce facilitates easy deployment of applications. However, there may be scenarios that require lot of configuration changes to be made during deployment. This can be achieved through setup projects.

Setup projects enable you to create Windows Installer files for your solutions so that the solutions can be distributed for installation on another machine or Web server. Setup applications or projects are configurable to a great extent.

You can add a setup project to your solution as follows:

1. Open your solution in Visual Studio 2008.
2. Choose **File→Add→ New Project** to open the **Add New Project** dialog box.
3. Expand Other Project Types in the **Project Types** pane and click **Setup And Deployment** as shown in figure 22.18.

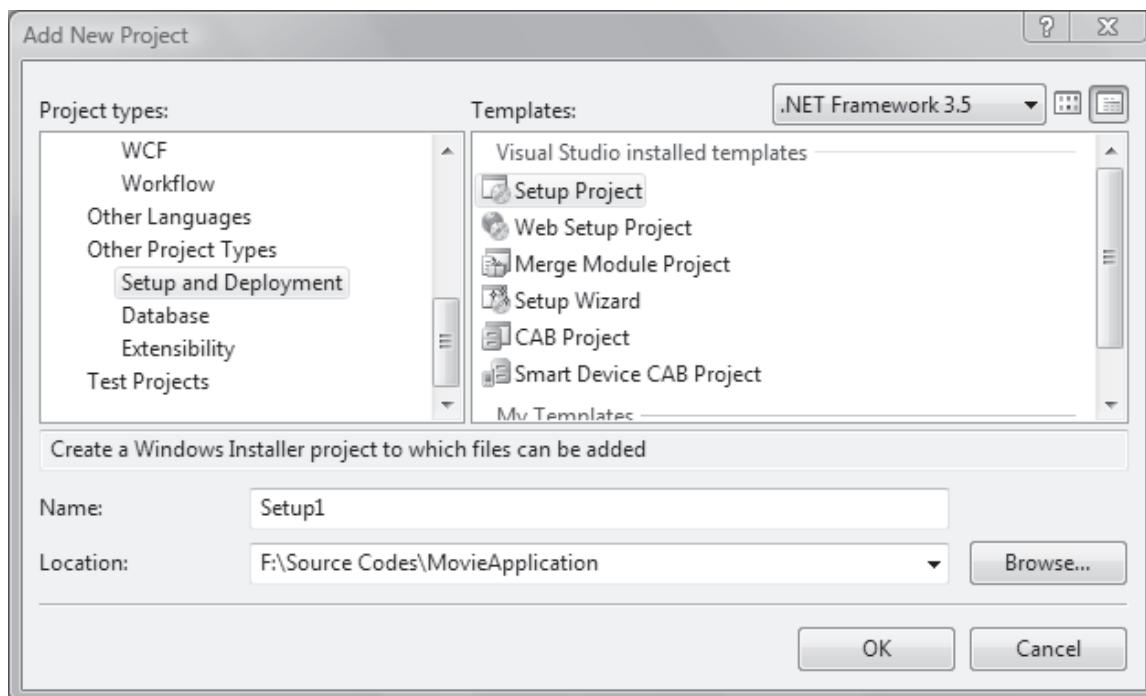


Figure 22.18: Setup and Deployment Project types

4. Click **Setup Project** in the **Templates** pane, and then, click **OK**.

After compilation, a setup project generates an .msi file. This file includes a setup wizard for the application. On double-clicking the .msi file, installation begins. Each setup project includes six editors that allow you to configure the contents and the behavior of the setup project. These six editors are as follows:

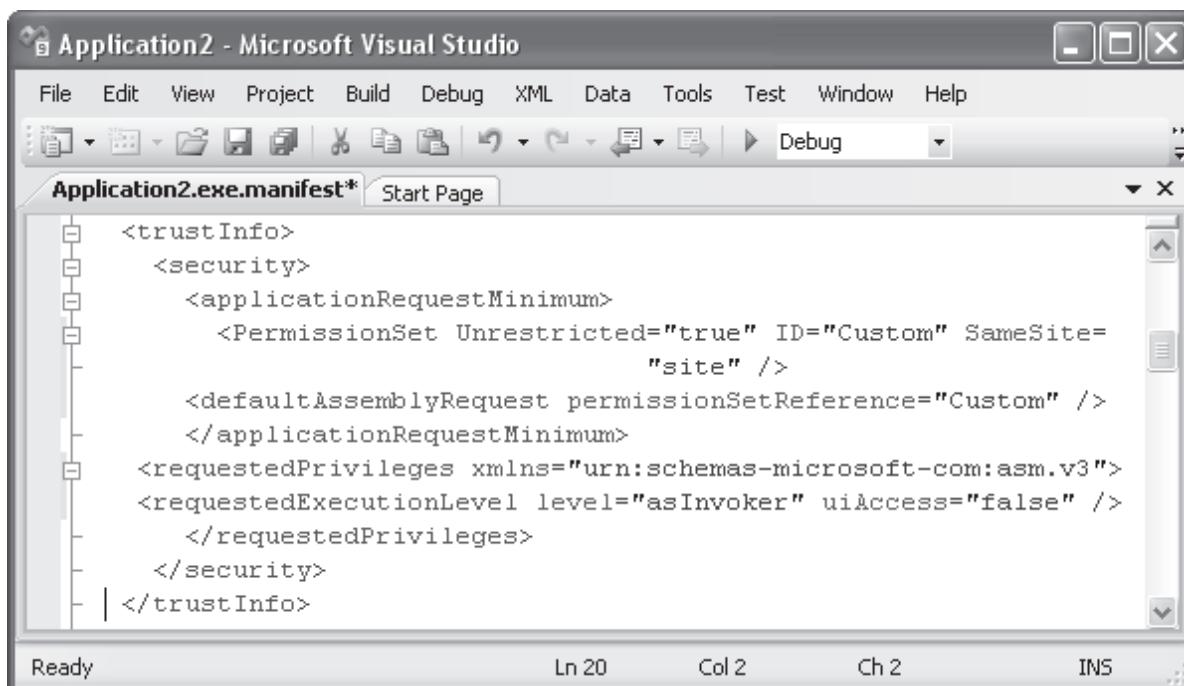
- ➔ File System Editor

- Registry
- File Types
- User Interface
- Custom Actions
- Launch Conditions

These setup Project Editors are similar to those in Visual Studio 2005 with no significant change.

## 22.5 Configuring Elevated Permissions

A number of features and tasks in Windows-based applications can be completed without administrator permissions. However, occasionally, a ClickOnce application may need elevated permissions (such as that of an administrator). For example, applications interacting with a Microsoft Word document would require additional permissions. In such cases, during installation, the user will see a prompt in the form of a message box to grant permission. If the user grants the permission, the installation continues and the application will be installed. If the user chooses not to grant, the installation will be stopped. Settings given in the `<trustInfo>` element of the application manifest enable to determine permissions for an application. This information is auto-generated by Visual Studio based on the settings on the project's **Security** property page. A sample manifest file showing the `<trustInfo>` element is displayed in figure 22.19.



```

<trustInfo>
    <security>
        <applicationRequestMinimum>
            <PermissionSet Unrestricted="true" ID="Custom" SameSite=
                "site" />
            <defaultAssemblyRequest permissionSetReference="Custom" />
        </applicationRequestMinimum>
        <requestedPrivileges xmlns="urn:schemas-microsoft-com:asm.v3">
            <requestedExecutionLevel level="asInvoker" uiAccess="false" />
        </requestedPrivileges>
    </security>
</trustInfo>

```

Figure 22.19: The `trustInfo` element

ClickOnce enables developers to build applications that can execute in one of three modes: partially trusted, fully trusted, or partially trusted with elevated permissions.

ClickOnce provides two technologies to request elevated permissions on a client computer:

#### → **Permission Elevation**

This displays a security dialog box to the user when your application runs for the first time. The **Permission Elevation** dialog box notifies users about the publisher of the application. An example is shown in figure 22.20.

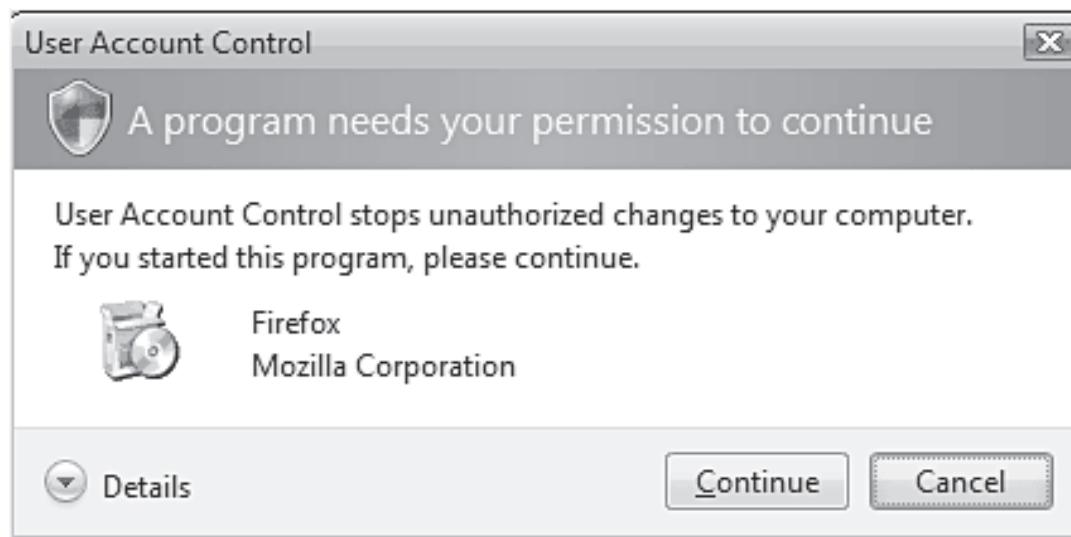


Figure 22.20: Permission Elevation

This approach enables users to decide whether or not additional trust should be granted. In case the administrator has not enabled permission elevation, the application will not get permission to execute. The application will not run and no notification will be displayed to the user.

#### → **Trusted Application Deployment**

Trusted Application Deployment enables a system administrator to install a publisher's certificate on a client computer. This installation is one-time only, which means that after it is installed the first time, system administrator will not need to install it again on the same machine.

The certificate enables to assure your users that the application source is trustworthy. After installation, any applications signed with that certificate will be considered as trusted. They can run at full trust on the local computer without being prompted for elevating permissions. Windows Vista includes a certificate generation tool.

This is shown in figure 22.21.

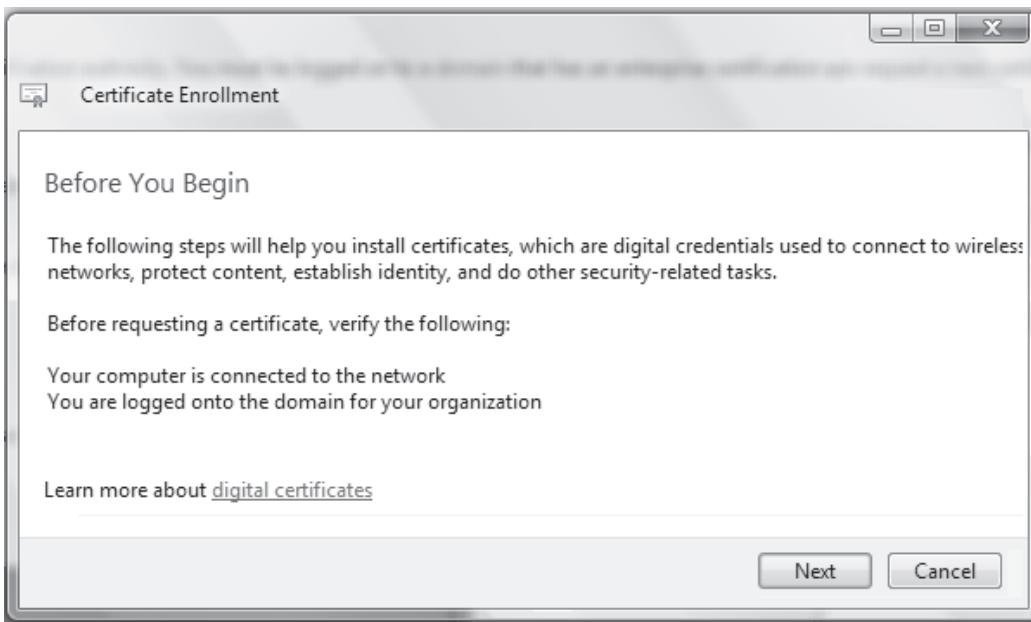


Figure 22.21: Certificate creation tool

Among these two, the appropriate technology to be used for an application will depend on the deployment environment. Trusted Application Deployment is described in more detail in the next section.

## 22.6 Configuring Trusted Application Deployments

Trusted Application Deployment enables enterprises to grant more permissions to managed applications safely and securely without interrupting the user for elevating permissions. By using Trusted Application Deployment, a client computer can be configured by the enterprise to include a list of trusted publishers identified using Authenticode certificates.

Once that is done, whenever any ClickOnce application signed by a publisher in the list is encountered, it receives a higher level of trust.

As mentioned earlier, for implementing Trusted Application Deployment, a user's computer needs to be configured only the first time. Hence, at an organizational level, this configuration can be achieved through a global policy.

The following steps are required to implement Trusted Application Deployment:

1. Acquire a certificate for the publisher
2. Include the publisher in the trusted publishers list on all clients
3. Create the ClickOnce application
4. Sign the manifest with the publisher's certificate
5. Publish the application on client computers

Trusted Application Deployment can be used to provide elevated permissions to ClickOnce applications deployed either over the Web or through a file share. However, it is not required for CD distributions of ClickOnce applications because, by default, these applications have full trust.

## 22.7 Configuring Security Features in an Application

As a developer, it is important to ensure the security of your Windows Forms applications before deploying them.

Some security considerations that can be taken care of are as follows:

### → Securing File Access in Windows Forms

When applications are written that deal with files, it is the `FileIOPermission` class that is used to control file and folder access in the .NET Framework. This class by default, does not have access to partial trust environments. These environments can include the local Intranet and Internet zones.

When working with dialog boxes, it is the `FileDialogPermission` class that specifies what type of file dialog box your application can use. Table 22.1 shows the permission value required in order to use each `FileDialog` class.

Class	Required Permissions
<code>OpenFileDialog</code>	Open
<code>SaveFileDialog</code>	Save

Table 22.1: `FileDialog` Classes

Note that permissions differ depending on whether an application is present in the Internet zone or Intranet zone. Internet zone applications are allowed access only to `OpenFileDialog`, whereas Intranet applications have unrestricted file dialog permission.

For your applications, you can configure these permissions by opening the Project properties window and selecting Security tab.

Figure 22.22 shows these permissions being configured in the Visual Studio 2008 IDE.

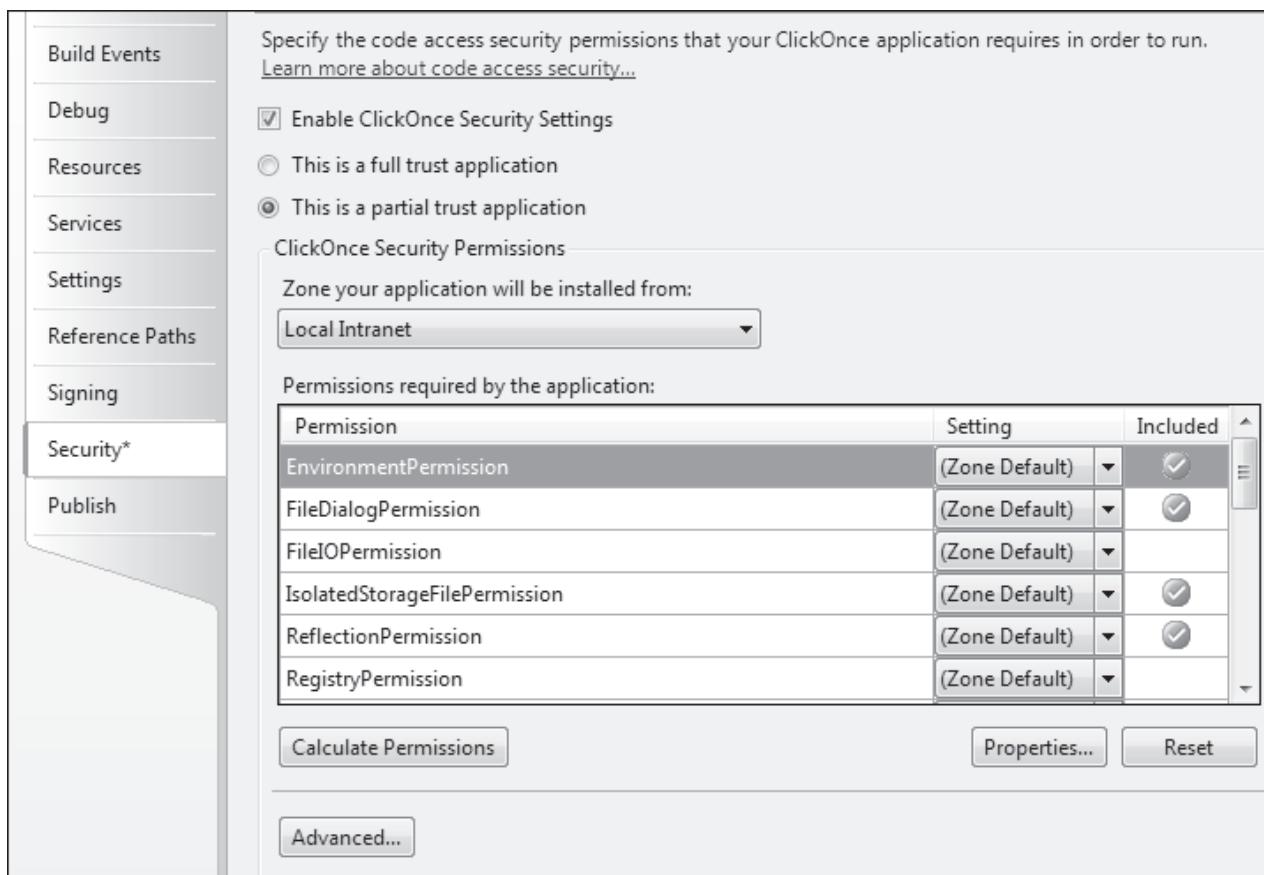


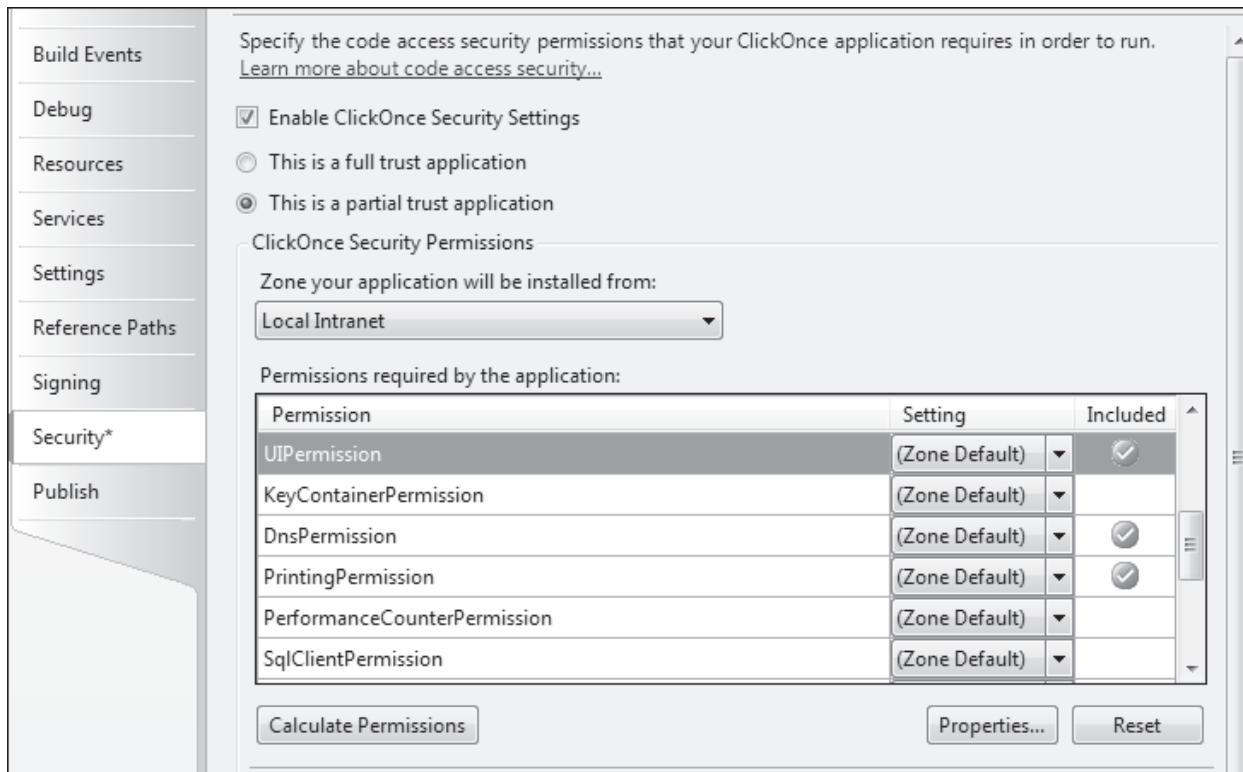
Figure 22.22: Configuring file and dialog permissions

As seen in figure 22.22, it is important to select the Enable ClickOnce Security Settings check box. Then, you can choose to give either full trust or partial trust for your application by choosing one of the two option buttons. In the ClickOnce Security Permissions section, you can choose one of the zones: Intranet, Internet, or Custom. After that, you can choose the various permissions such as FileIOPermission and FileDialogPermission.

#### → Securing Clipboard Access in Windows Forms

Access to the Clipboard is controlled by the UIPermission class.

Figure 22.23 shows this class in the Security tab.



**Figure 22.23: UIPermission class**

The associated `UIPermissionClipboard` enumeration value indicates the level of access.

Table 22.2 lists the possible permission levels.

UIPermissionClipboard Value	Description
AllClipboard	Indicates that the Clipboard can be used without restriction
OwnClipboard	Indicates that the Clipboard can be used with a few restrictions
NoClipboard	Indicates that the Clipboard cannot be used at all

**Table 22.2: Permission Levels**

By default, the Local Intranet zone has `AllClipboard` access whereas the Internet zone has `OwnClipboard` access. The outcome of this is that applications can copy data to the Clipboard, but cannot paste to or read from the Clipboard through code. To get complete Clipboard access for your application, appropriate permissions must be elevated. In this manner, you can configure Clipboard security. You can access these options by selecting `UIPermission` and then, clicking the Properties button.

Figure 22.24 shows how you can specify these options.

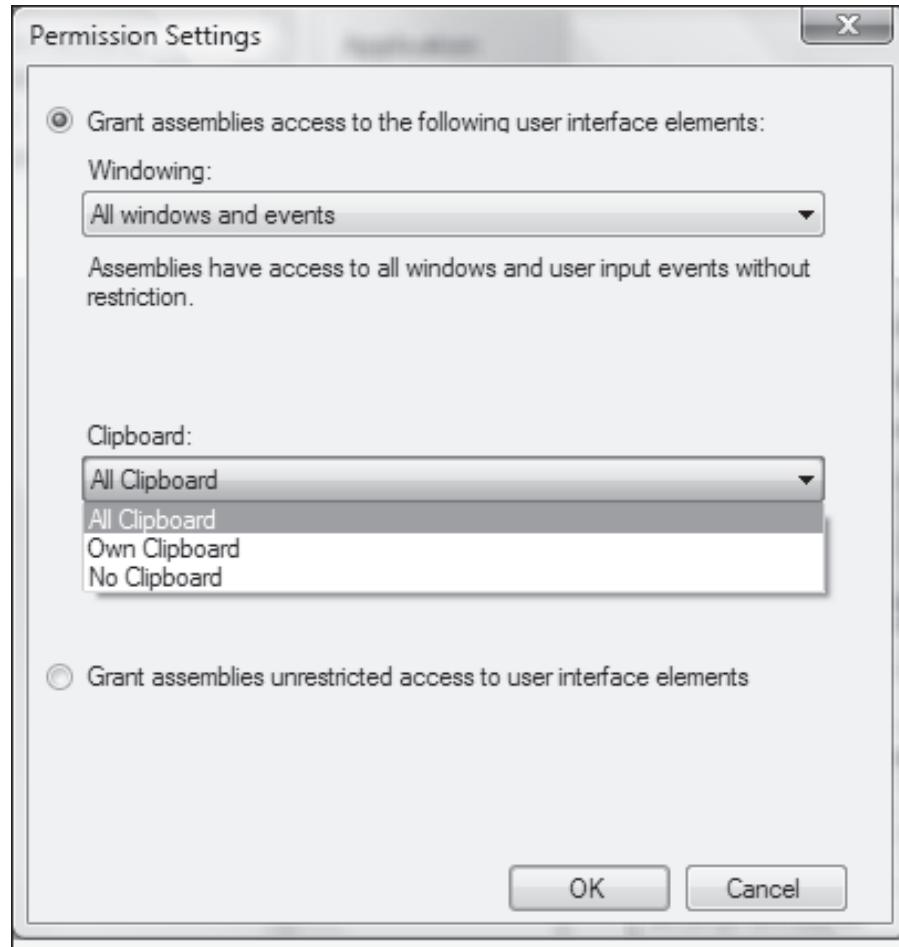


Figure 22.24: Granting Clipboard access

## Knowledge Check 1

1. Which of the following types of applications that are published using ClickOnce technology can be termed as a ClickOnce application?

(A)	Windows Forms (.exe file)
(B)	Windows Presentation Foundation (.xbap file)
(C)	Console application (.exe file)
(D)	Office documents (.doc files)

2. Which of the following statements about Trusted Application Deployment are true?

(A)	Trusted Application Deployment is necessary for CD distributions of ClickOnce applications because, by default, these applications have full trust.
(B)	By using Trusted Application Deployment, a client computer can be configured to include a list of trusted publishers identified using Authenticode certificates.
(C)	Trusted Application Deployment is a technology provided by ClickOnce to request elevated permissions on a client computer.
(D)	Using this technology, a system administrator performs frequent installations of a publisher's certificate on a client computer.

3. Which one of the following describes the correct reason why the offline publish location is not required while installing WPF browser applications using ClickOnce?

(A)	WPF browser applications are not available offline and are published in online-only mode
(B)	WPF browser applications are cached therefore, the location is not required
(C)	WPF browser applications are not supported by ClickOnce
(D)	WPF browser applications are published in a default offline location

4. Which one of the following statements is the correct definition of ClickOnce?

(A)	ClickOnce is a deployment technology facilitating creation of self-updating Windows-based applications which can be installed and executed with least user interaction
(B)	ClickOnce is a development technology that facilitates creation of secure Windows-based applications
(C)	ClickOnce is a technology that facilitates debugging Windows-based applications
(D)	ClickOnce is a technology used to elevate permissions

5. When you build UAC-aware applications in Visual Studio for Windows Vista, by default, what kind of manifest file is generated?

(A)	External
(B)	Assembly
(C)	Embedded
(D)	Resource

6. Which of the following denote valid locations where a ClickOnce application can be published?

(A)	Disk Path
(B)	File Share Path
(C)	Web site Path
(D)	FTP Path



## Module Summary

- ClickOnce is a deployment technology that facilitates creation of self-updating Windows-based applications which can be installed and executed with least user interaction.
- ClickOnce can be used to install Windows Forms applications, WPF browser applications, and VSTO applications.
- Visual Studio generates an external manifest file containing the UAC data for ClickOnce projects that are UAC-aware.
- Windows Forms setup projects enable you to create Windows Installer files so that the solutions can be distributed for installation on another machine or Web server.
- ClickOnce provides two technologies to request elevated permissions on a client computer namely, Permission Elevation and Trusted Application Deployment.
- You must configure the security features of your Windows Forms applications that are going to be deployed so that they would be secure.

# Module - 23

## Configuring and Deploying Applications (Lab)

In this Module, you will learn about:

- ➔ Configure ClickOnce security features in an application
- ➔ Use ClickOnce for publishing a Windows Forms application
- ➔ Use ClickOnce for publishing a VSTO application
- ➔ Configure required permissions in a Windows Forms application
- ➔ Create Windows Forms setup applications
- ➔ Use ClickOnce to publish and install a WPF application

## Part I – 60 Minutes

### Exercise

GoodReads is a popular book store in Surrey, UK. You have been hired by the GoodReads proprietors to develop a BookStore application. Using Visual Studio 2008, Windows Forms, and SQL Server 2008, you have successfully built this application. Now, your next task is to publish and deploy it on a file share. You have been advised to use ClickOnce for publishing the application, and also configure appropriate permissions for it. You must also ensure that updates happen regularly. The application should check for updates every three days after it starts.

### Solution:

Before publishing the BookStore application, you need to configure its security.

### Configuring Security

The steps to configure the security of the BookStore application are listed as follows:

- 1. Select Project → BookStore Properties in the Visual Studio 2008 IDE.**
- 2. Click the Security tab.**

The Security properties page will have Enable ClickOnce Security Settings selected by default. If not, then select it as shown in figure 23.1.

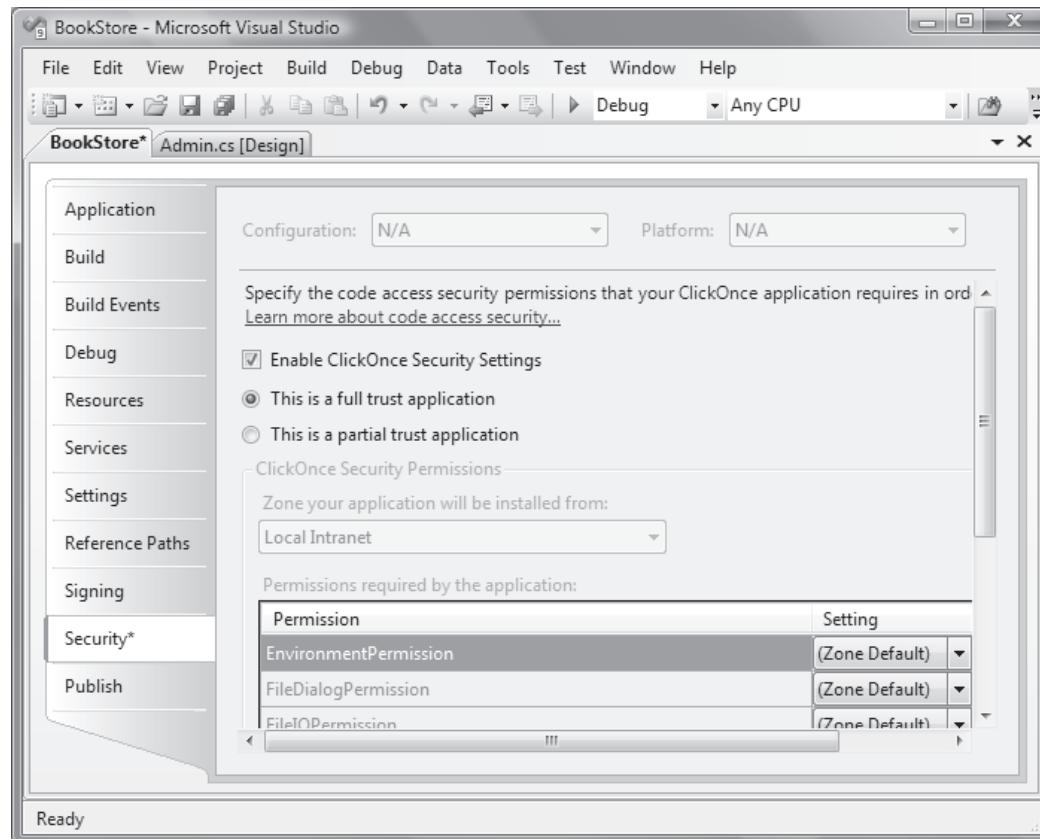


Figure 23.1: The Security Properties Page

This will ensure that ClickOnce security settings are applied to the project.

**3. Select the option button, This is a partial trust application, as shown in figure 23.2.**

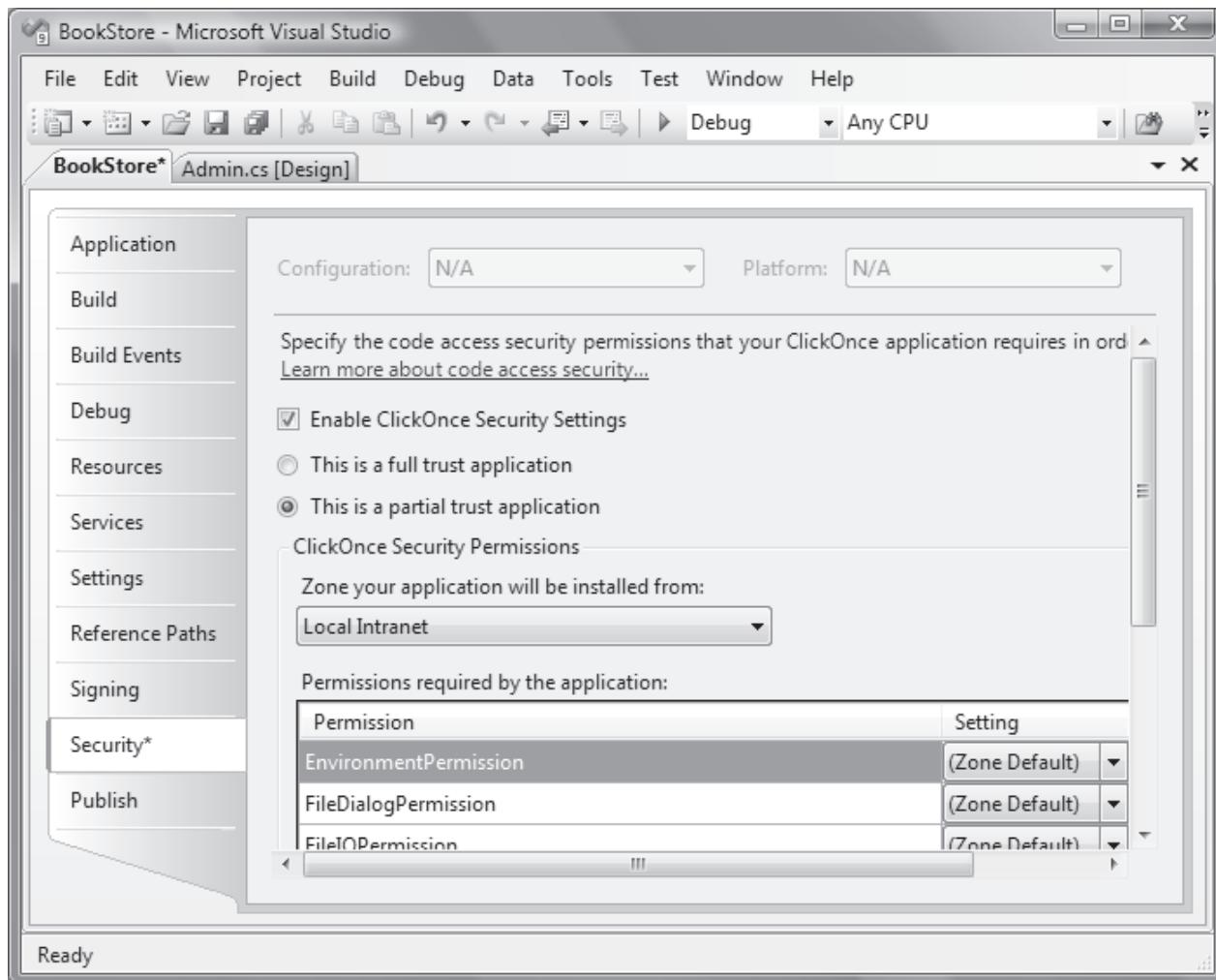


Figure 23.2: Setting Partial Trust

A full trust application will have unrestricted access to the application and system resources, whereas a partial trust application will provide restricted access.

As a developer of an application, you need to control user access to the application and system resources. The partial trust option in a ClickOnce application enables you to achieve this.

4. Select the Zone dropdown to view the available options as shown in figure 23.3.

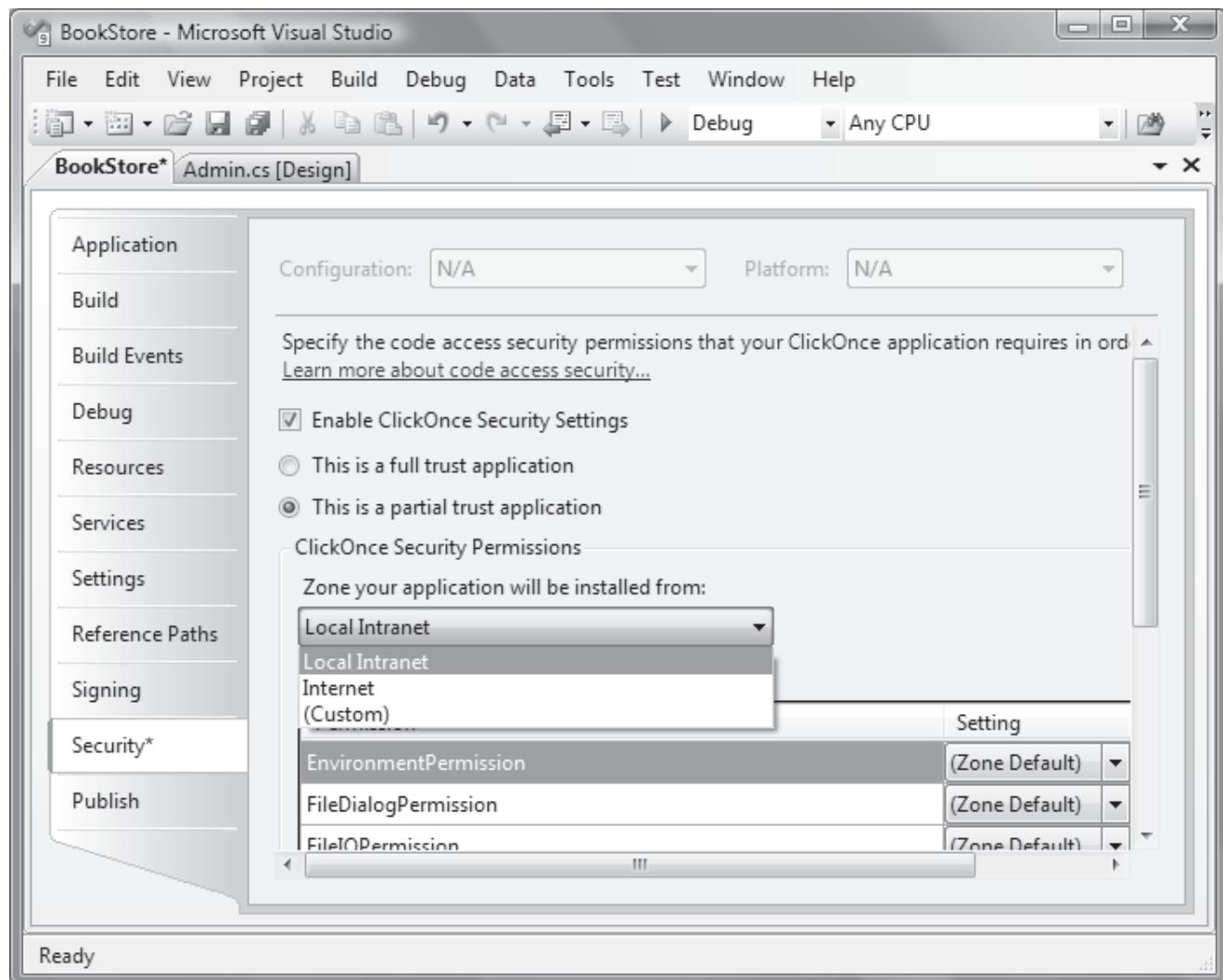


Figure 23.3: Selecting the zone

There are three zones from which your application can be installed: Local Intranet, Internet, and Custom. ClickOnce applications that are executed from the Internet, by default, run in the Internet security zone whereas ClickOnce applications executed from a file share run in the intranet security zone. You can also set the zone to custom to provide custom security settings.

5. Scroll down in the Permission list to view the available permissions as shown in figure 23.4.

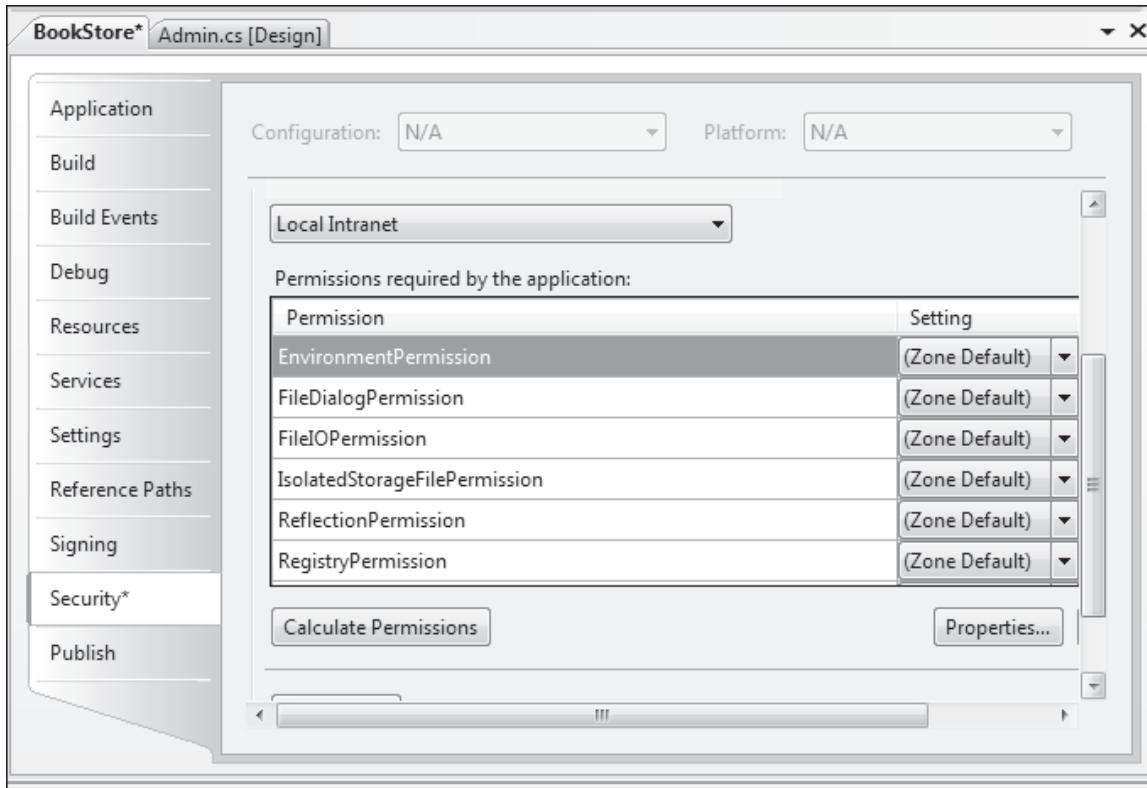


Figure 23.4: Permissions

Since the BookStore application interacts with an SQL Server database, permissions related to SQL Server need to be configured.

**6. Expand the options for SqlClientPermission as shown in figure 23.5.**

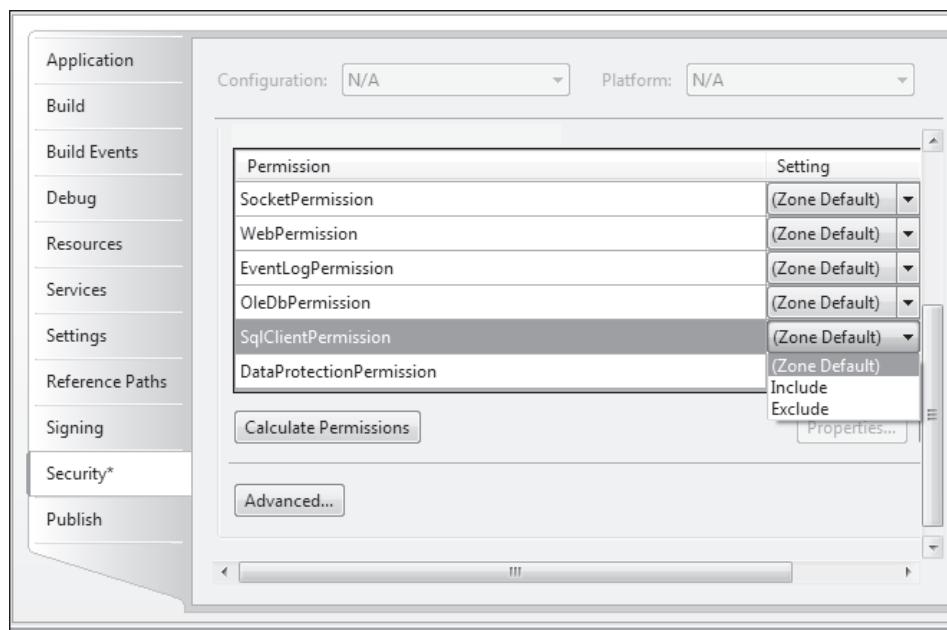


Figure 23.5: SqlClientPermission Settings

**7. Select Include in the dropdown as shown in figure 23.6.**

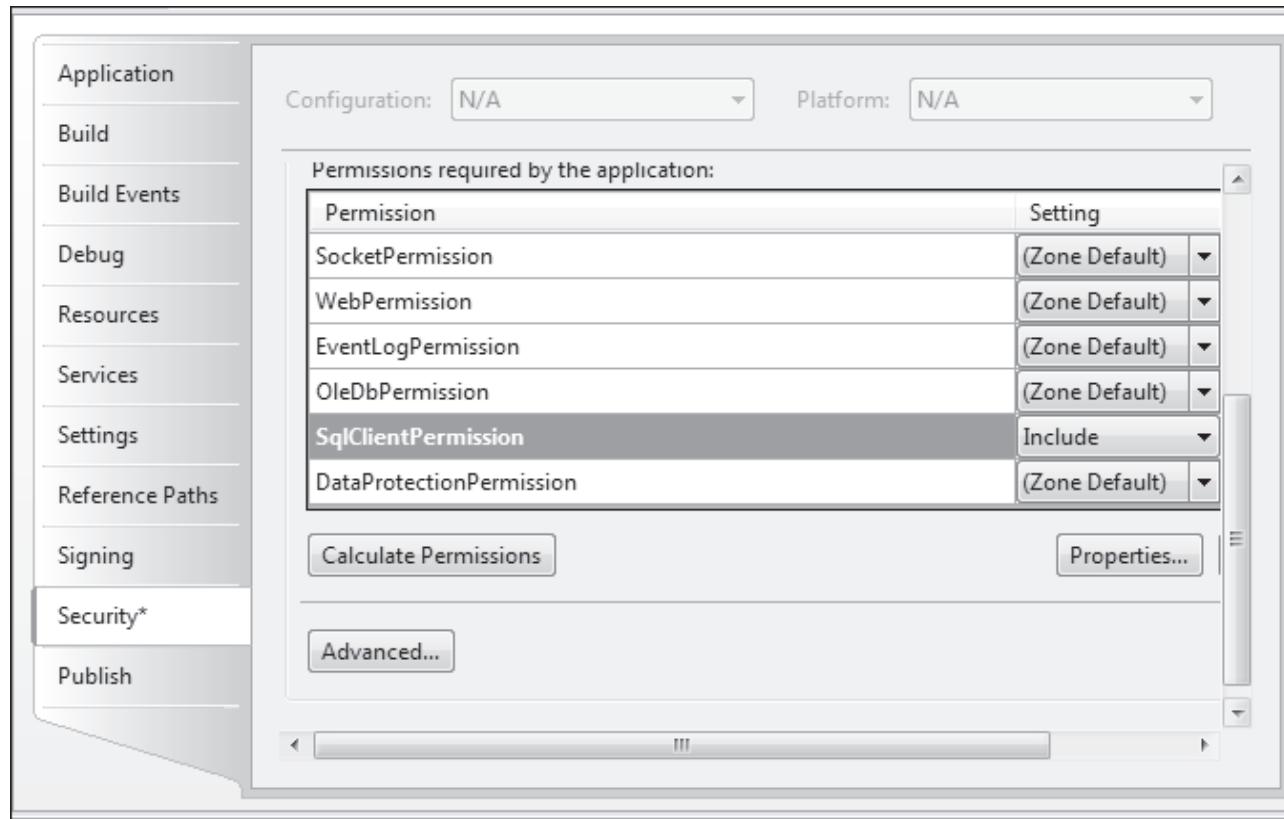


Figure 23.6: Including SqlClientPermission

Through `SqlClientPermission`, you can specify whether the user can connect to SQL Server and whether any restrictions are placed on the connection string.

**8. Click Properties and retain the default grant permissions as shown in figure 23.7.**

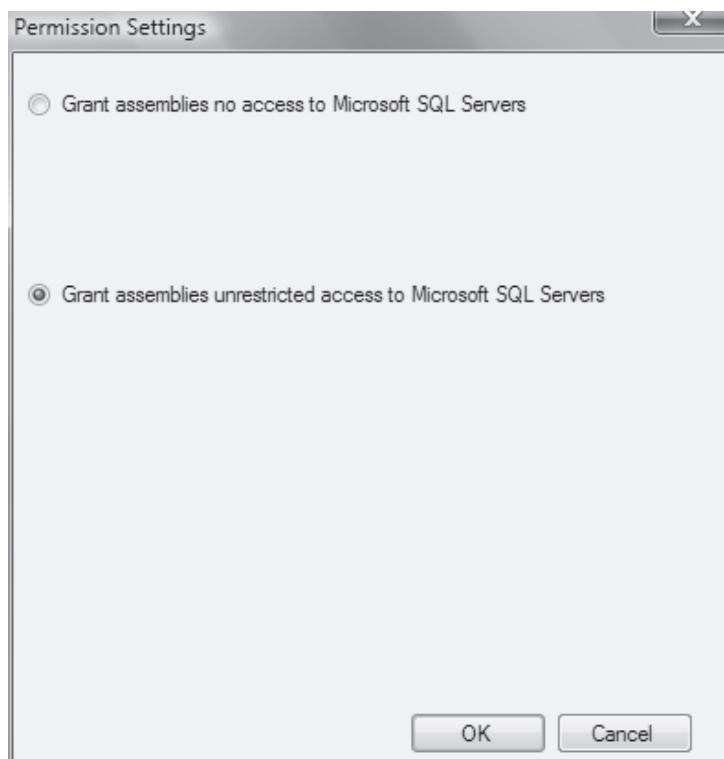


Figure 23.7: Granting assemblies SQL permissions

If you choose the first option button, no access will be given to SQL Server and when your code attempts any connection to an SQL Server, an exception would be thrown. Since the BookStore utilizes a lot of data handling operations, you need to grant unrestricted access to SQL Server.

Similarly, you can configure other categories of permissions that you require.

### Publishing using ClickOnce:

- Click the Publish tab. This displays the Publish properties page as shown in figure 23.8.

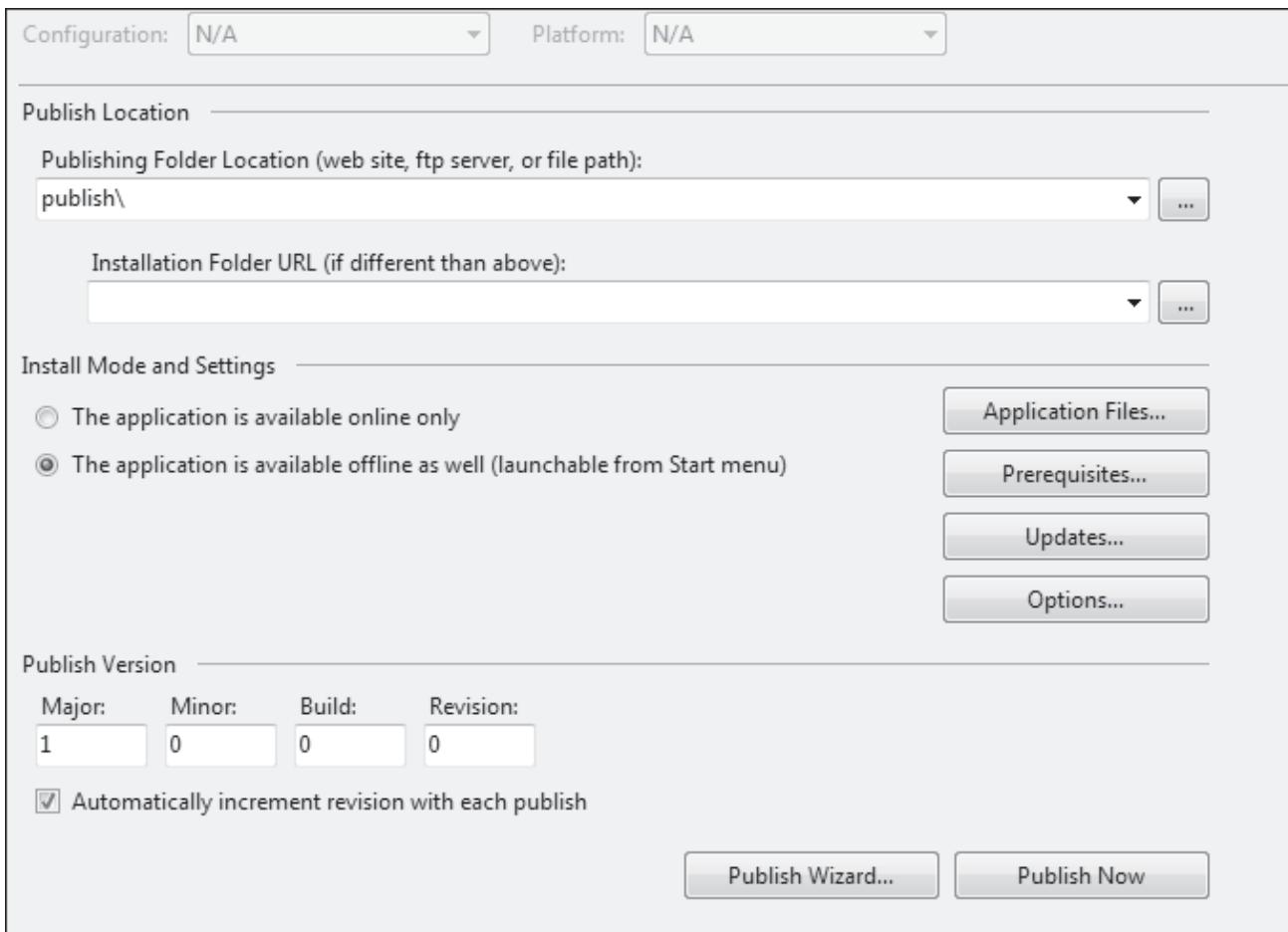
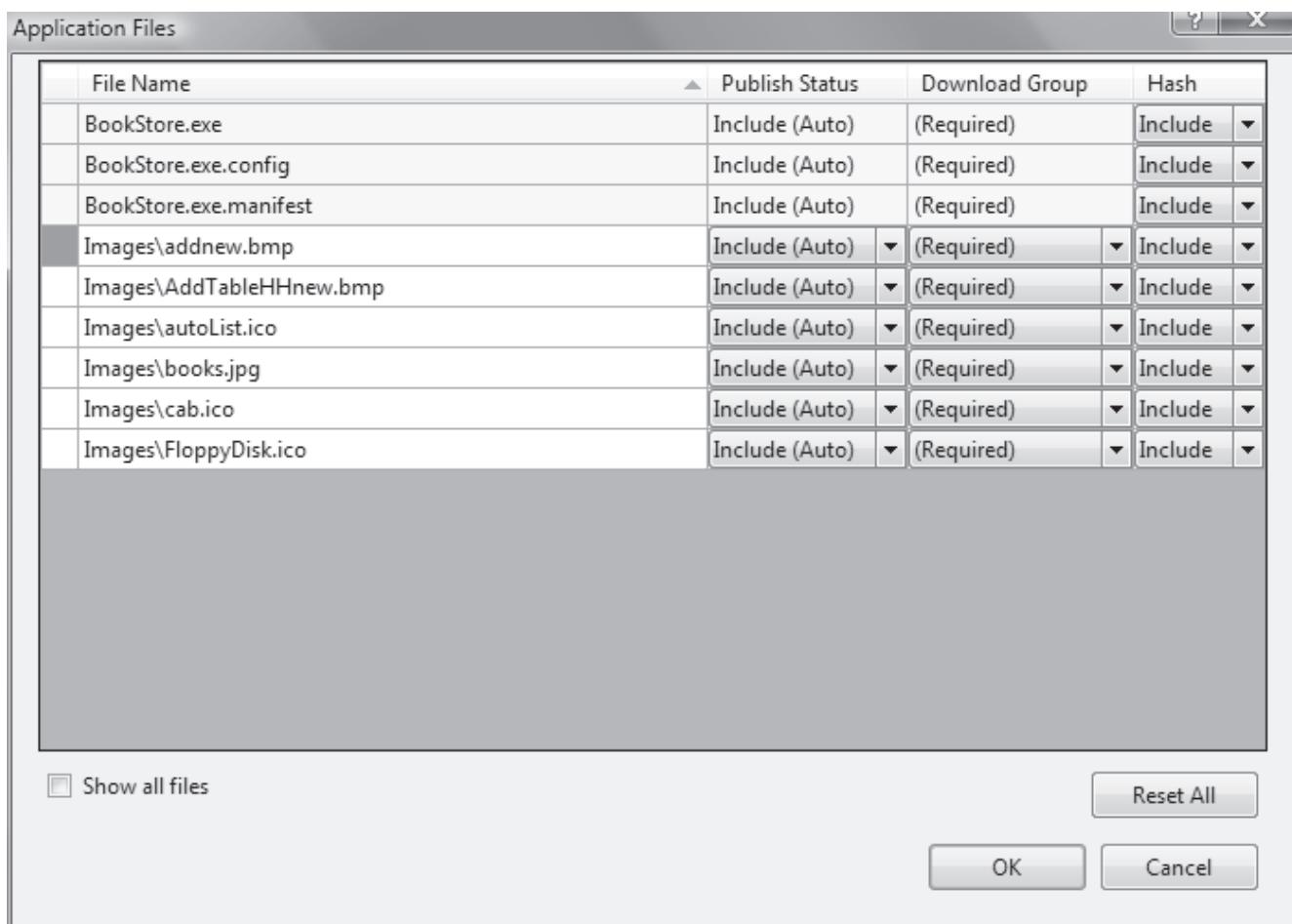


Figure 23.8: Publish Properties Page

- Click Application Files in the Install Mode and Settings section.

This launches the Application Files dialog box as shown in figure 23.9.



**Figure 23.9: Application Files dialog box**

Here, you can specify which application files are required in the final installation. If any of the files are not required, you can use Exclude option to exclude them from the published project.

- 3. Click OK.**

4. Click Prerequisites. This displays the Prerequisites dialog box as shown in figure 23.10.

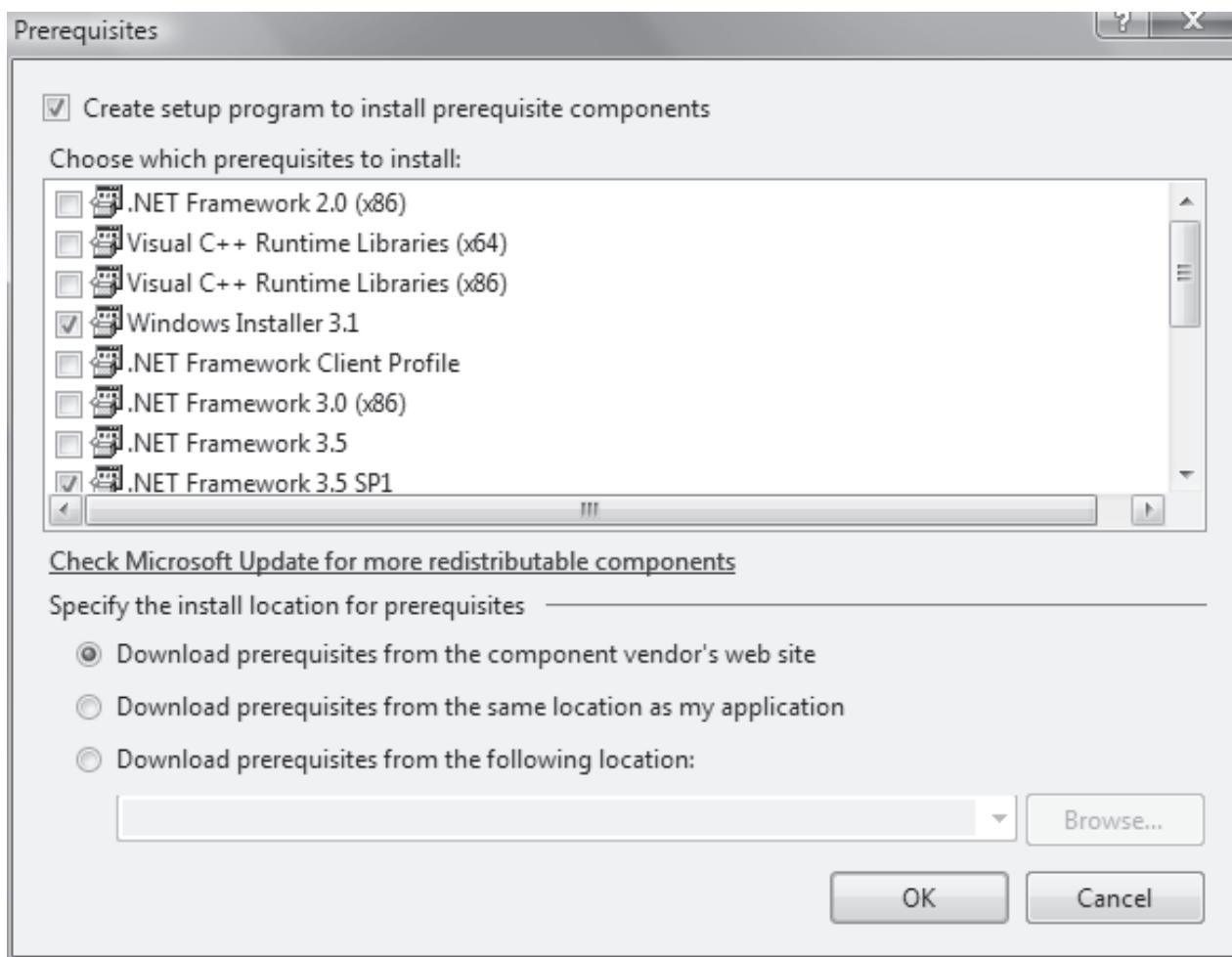


Figure 23.10: Prerequisites dialog box

Here, you can specify the prerequisites required for the application to run. For example, there are different versions of .NET Framework available. You can specify the version that should be a prerequisite for your application to run.

5. Retain the defaults for the BookStore application and click OK.  
6. Click Updates.

This launches the Application Updates dialog box as shown in figure 23.11. Here, you can specify if the application should check for updates and also when the updates should take place.

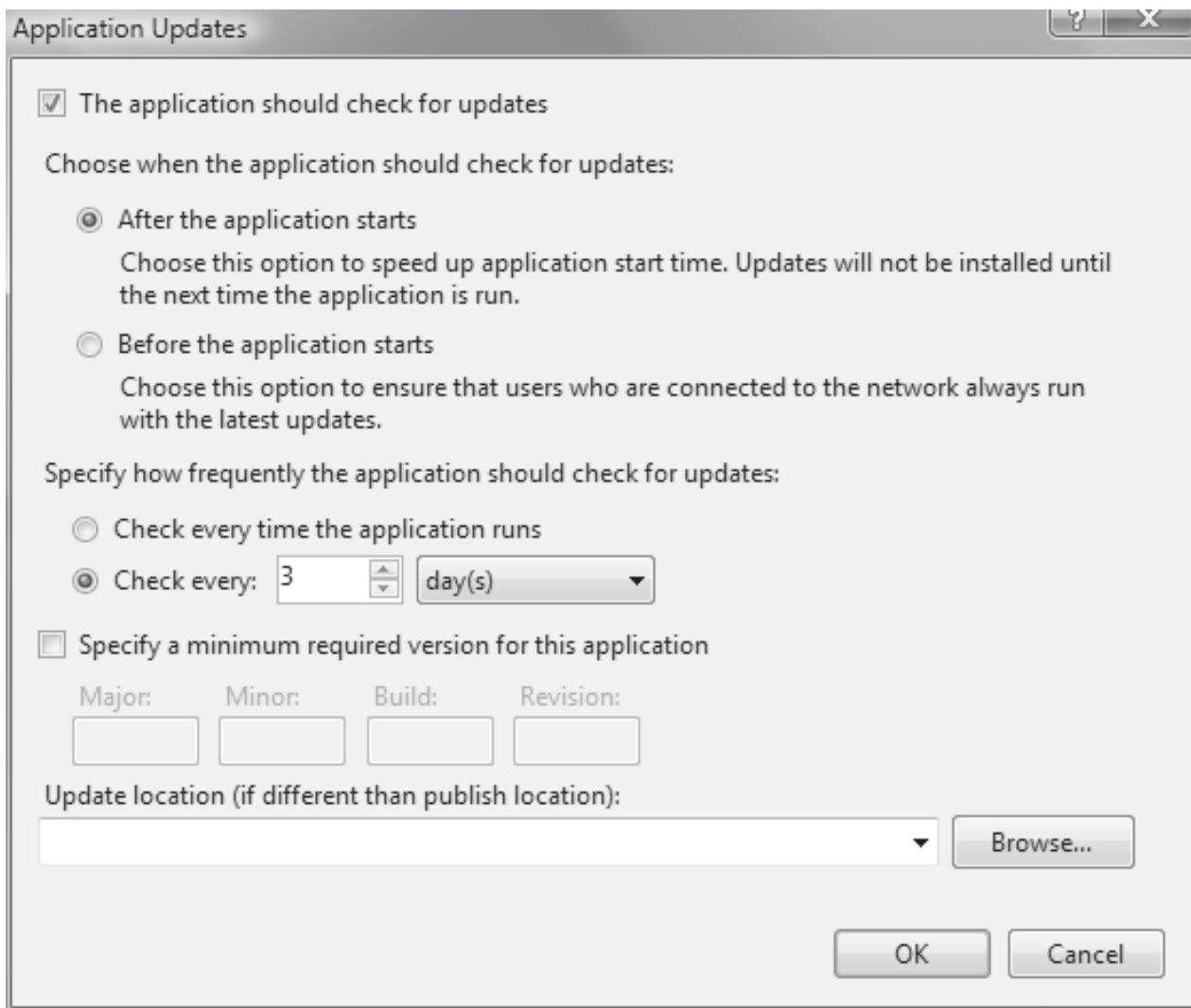


Figure 23.11: Application Updates dialog box

7. Specify the options as shown in figure 23.11 and click OK. This displays the Publish properties page once again.
8. Click Options. In the Publish Options dialog box that is displayed, you can configure publish options such as Publisher name, Product name, file associations, and so forth.

9. Type the Publisher name as Aptech Limited, Suite name, and Product name as BookStore as shown in figure 23.12.

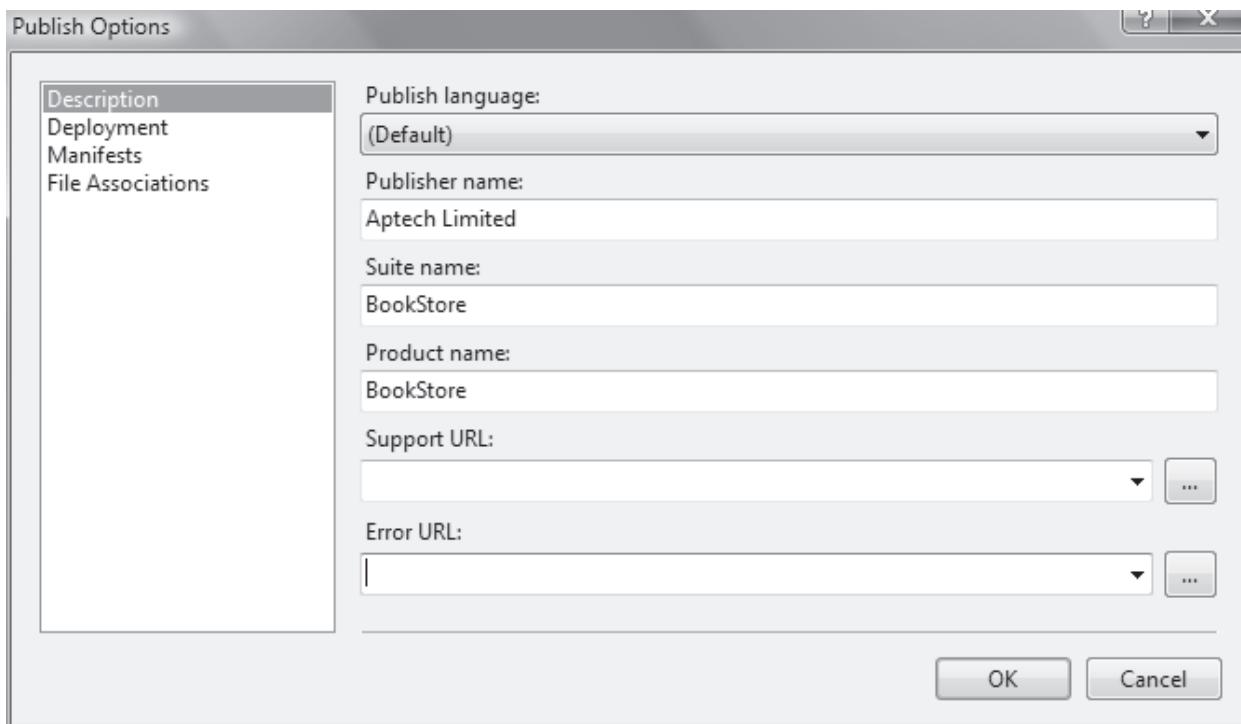


Figure 23.12: Publish Options dialog box

10. Click Deployment on the left pane. This shows deployment options as shown in figure 23.13.

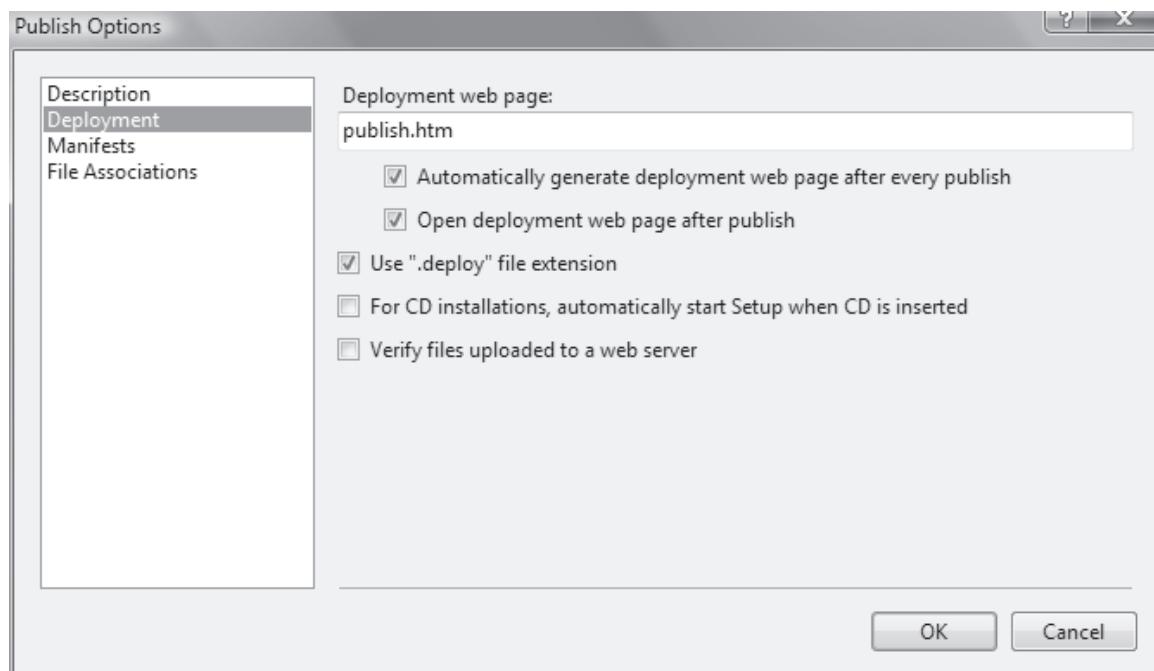


Figure 23.13: Deployment details

11. Retain the defaults and click OK. The Publish properties page is displayed.
12. Click Publish Wizard in the Publish properties page.

Clicking the Publish Wizard button launches the Publish Wizard page as shown in figure 23.14 which guides you through the ClickOnce publishing procedure.

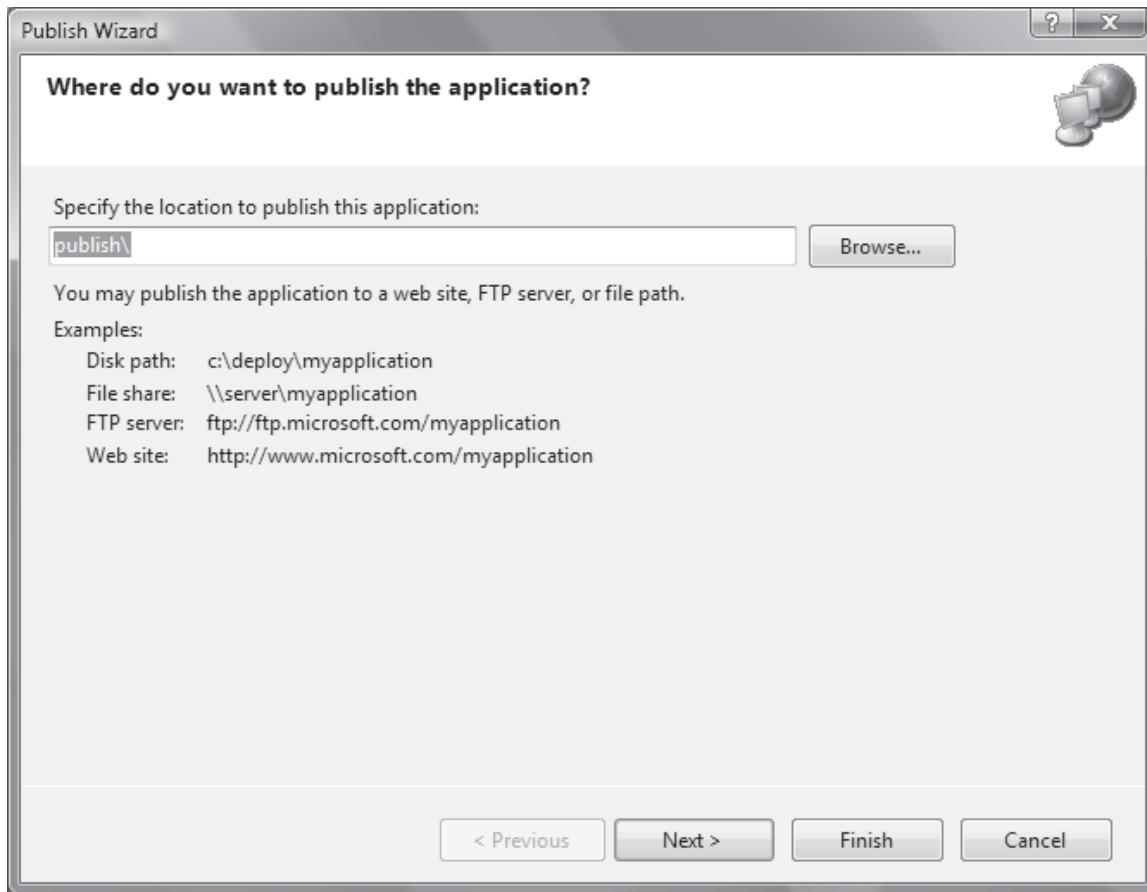


Figure 23.14: Publish Wizard

13. Specify the fileshare location as shown in figure 23.15.

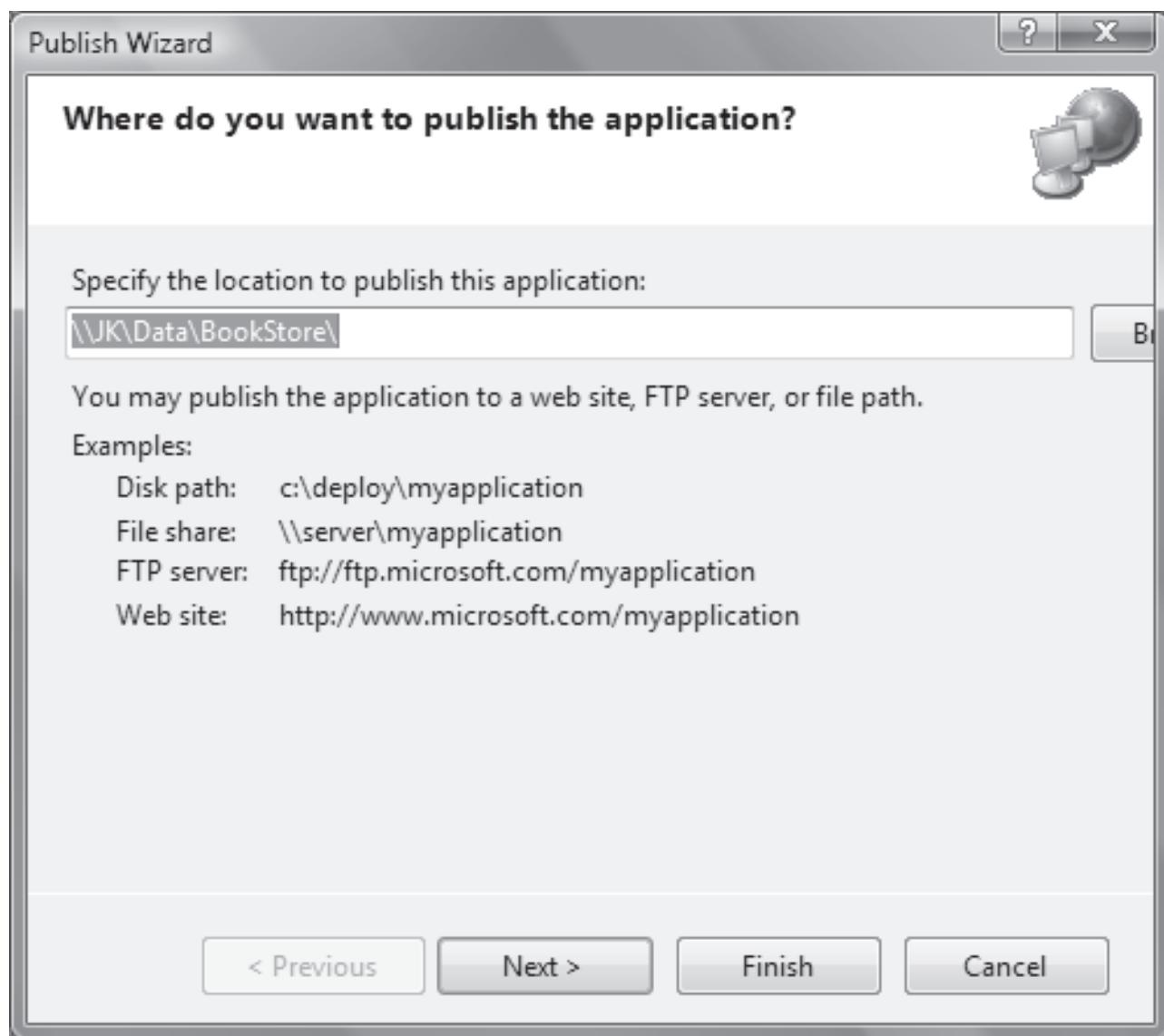


Figure 23.15: Specifying publish location

14. Click Next. The page for installation type is displayed. Here, you can specify how the users will install your application.

15. Select the fileshare option as shown in figure 23.16.

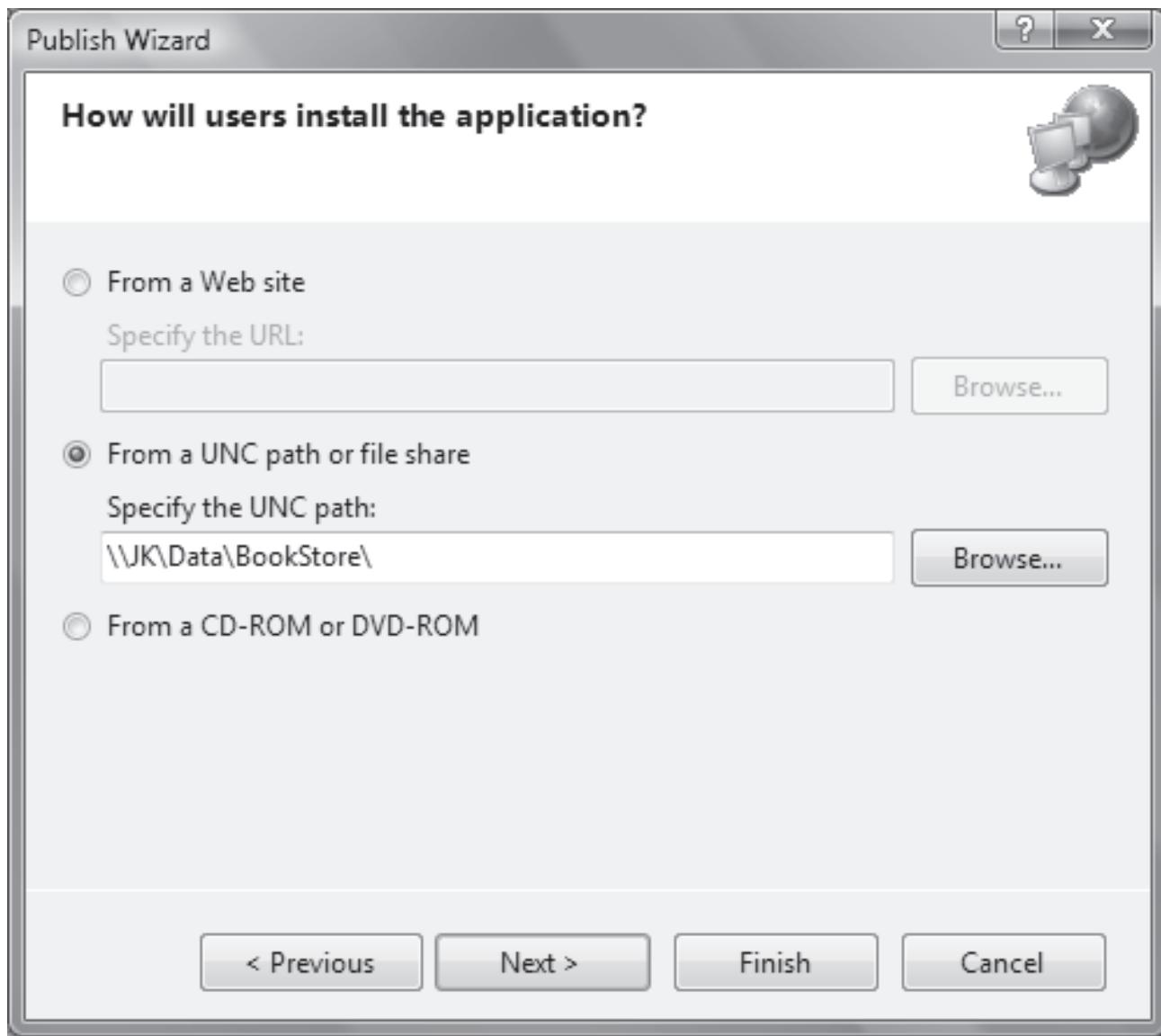


Figure 23.16: Specifying installation type

16. Click Next. The next page of the wizard is used to specify whether the application will be available online or offline.

17. Retain the default option, Yes, the application will be online or offline, as shown in figure 23.17.



Figure 23.17: Specifying online or offline availability

18. Click Next. This completes the wizard.

A message is displayed as seen in figure 23.18 to indicate that the application is ready to be published.

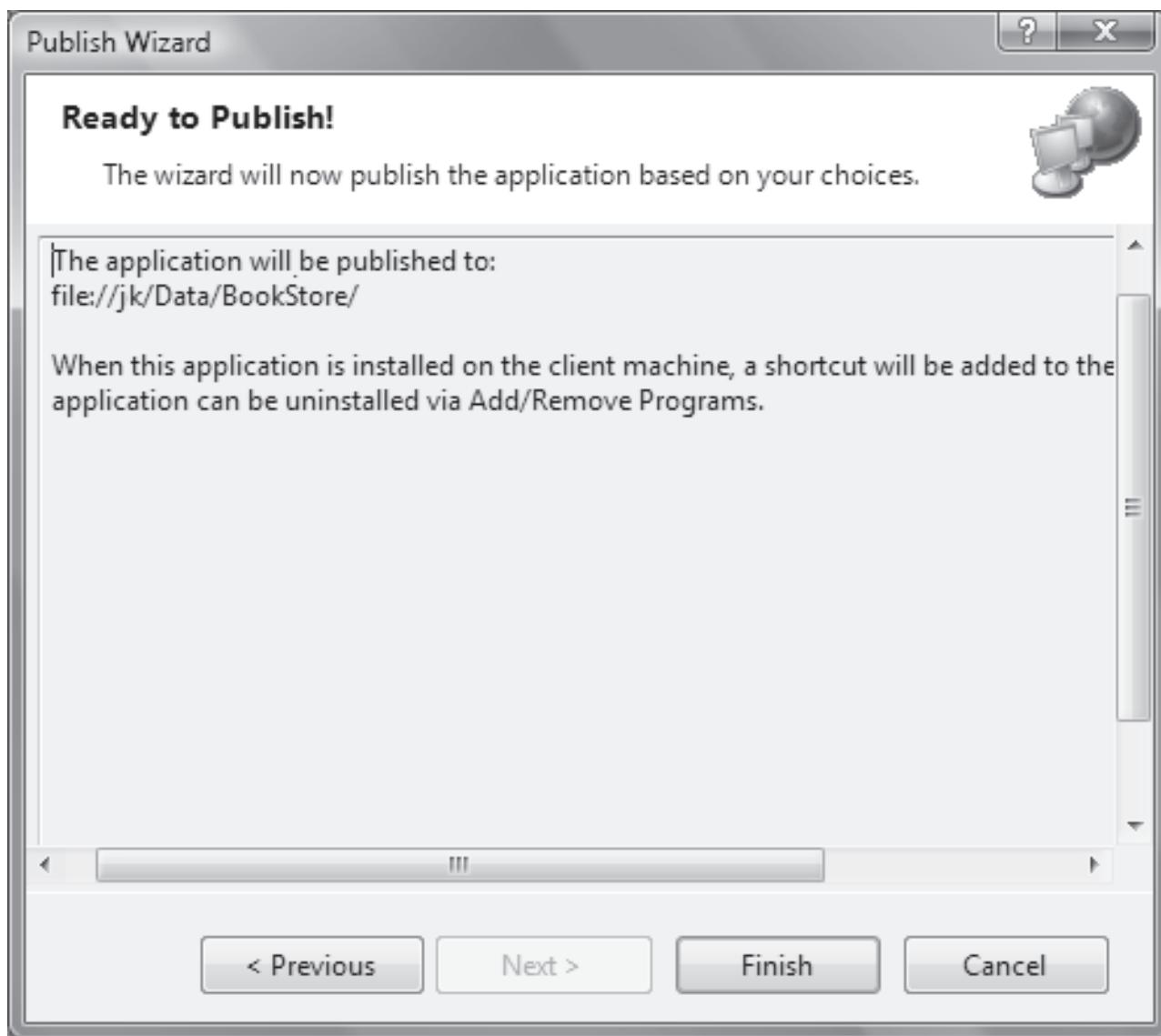


Figure 23.18: Final Page of Publish wizard

19. Click Finish.

When the publish.htm file is launched on the file share, the output will be as shown in figure 23.19.

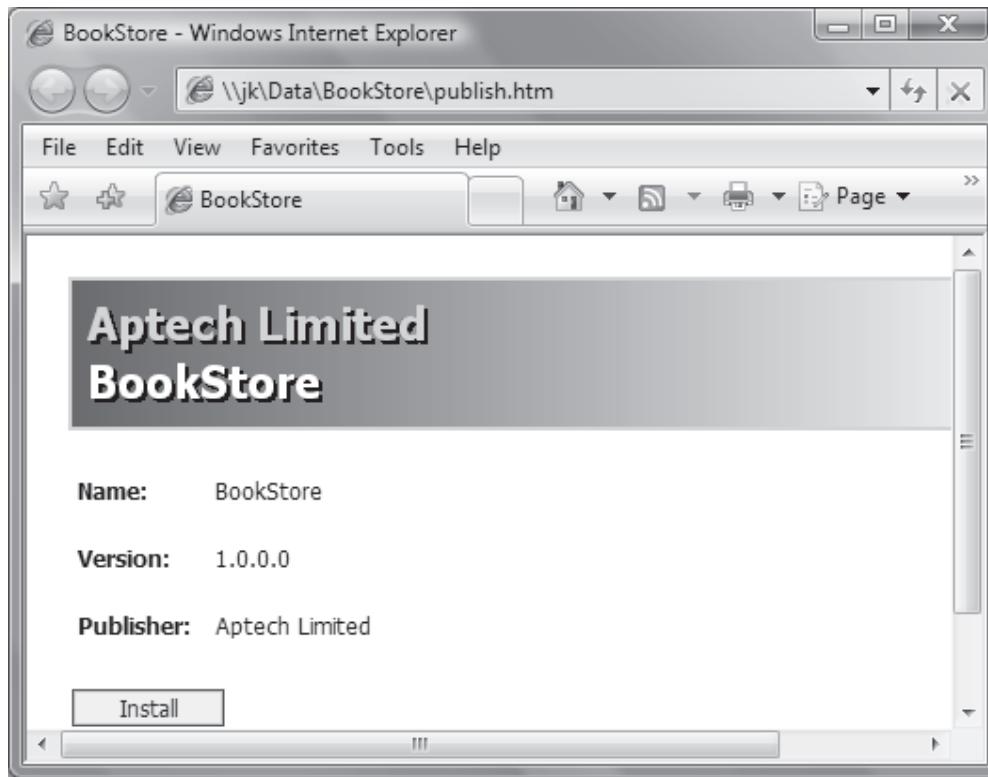


Figure 23.19: Publish.htm on the file share

20. Click **Install**. The application setup displays a prompt shown in figure 23.20 because the publisher is unknown.

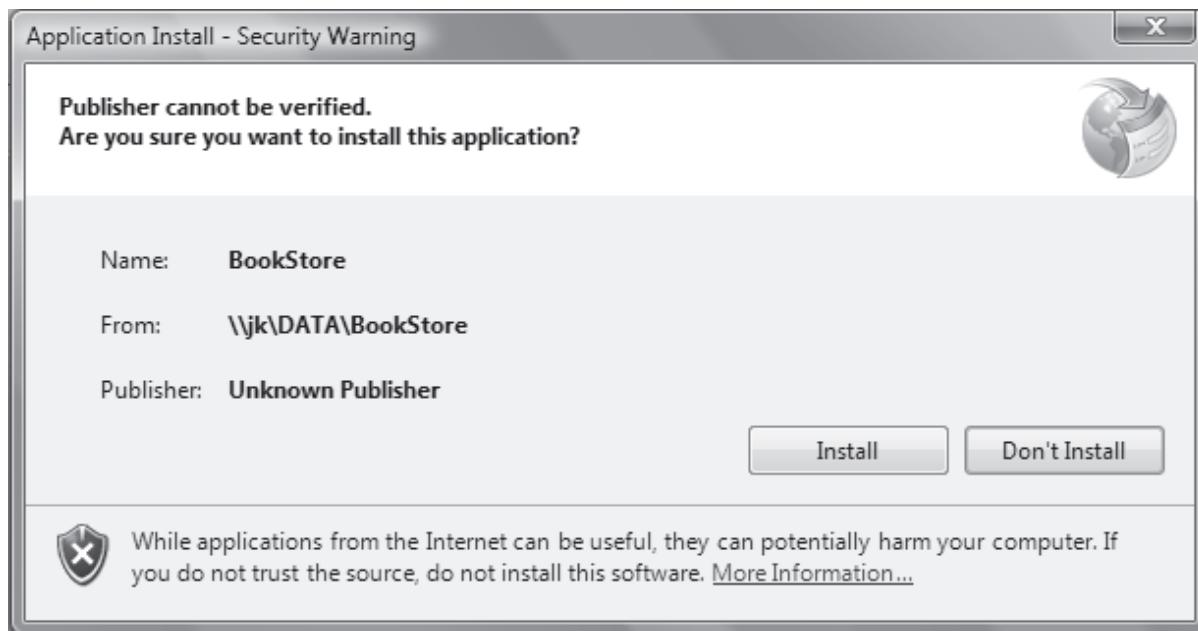


Figure 23.20: Security Warning for installing the application

**21. Click Install.**

The application is installed and you can execute various functionalities of the BookStore such as adding records, viewing records, and so forth.

When you attempt to close the Admin screen, an error occurs. This is depicted in figure 23.21.

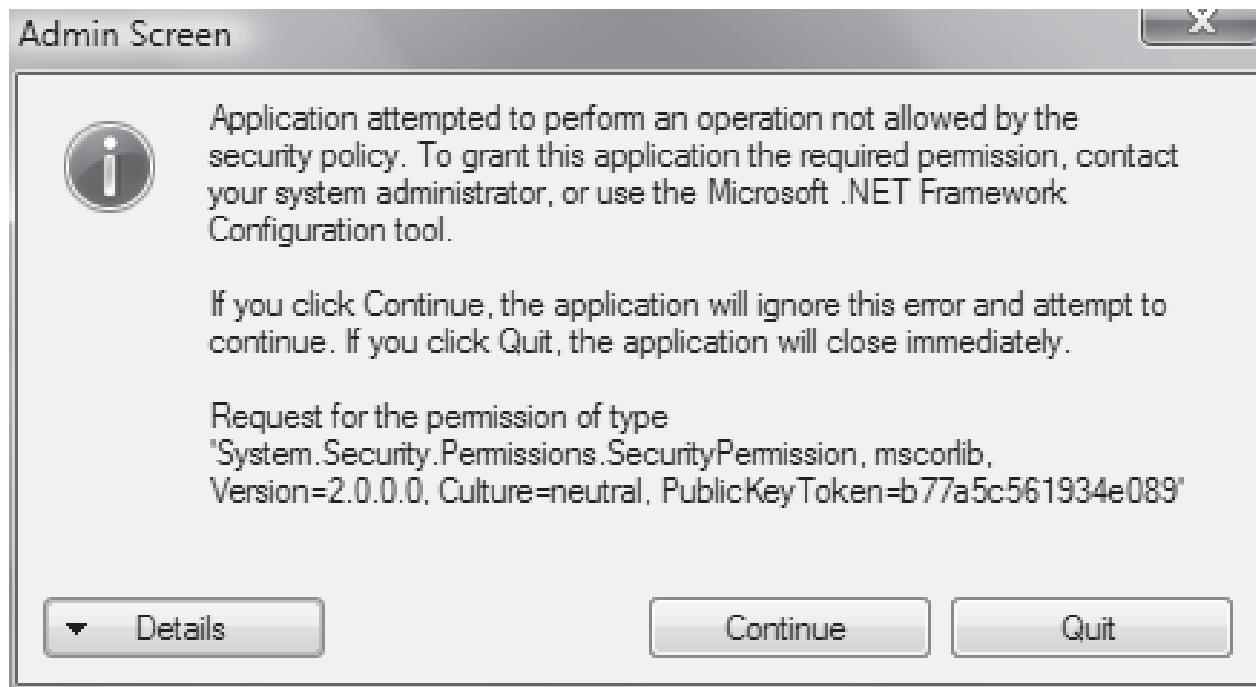


Figure 23.21: Security exception

This happens because of inadequate security permissions. Therefore, to resolve this issue, configure security permissions in your application.

**22. Click the Security tab in the Project Properties page.**

23. Select **SecurityPermission** and expand the options in the dropdown as shown in figure 23.22.

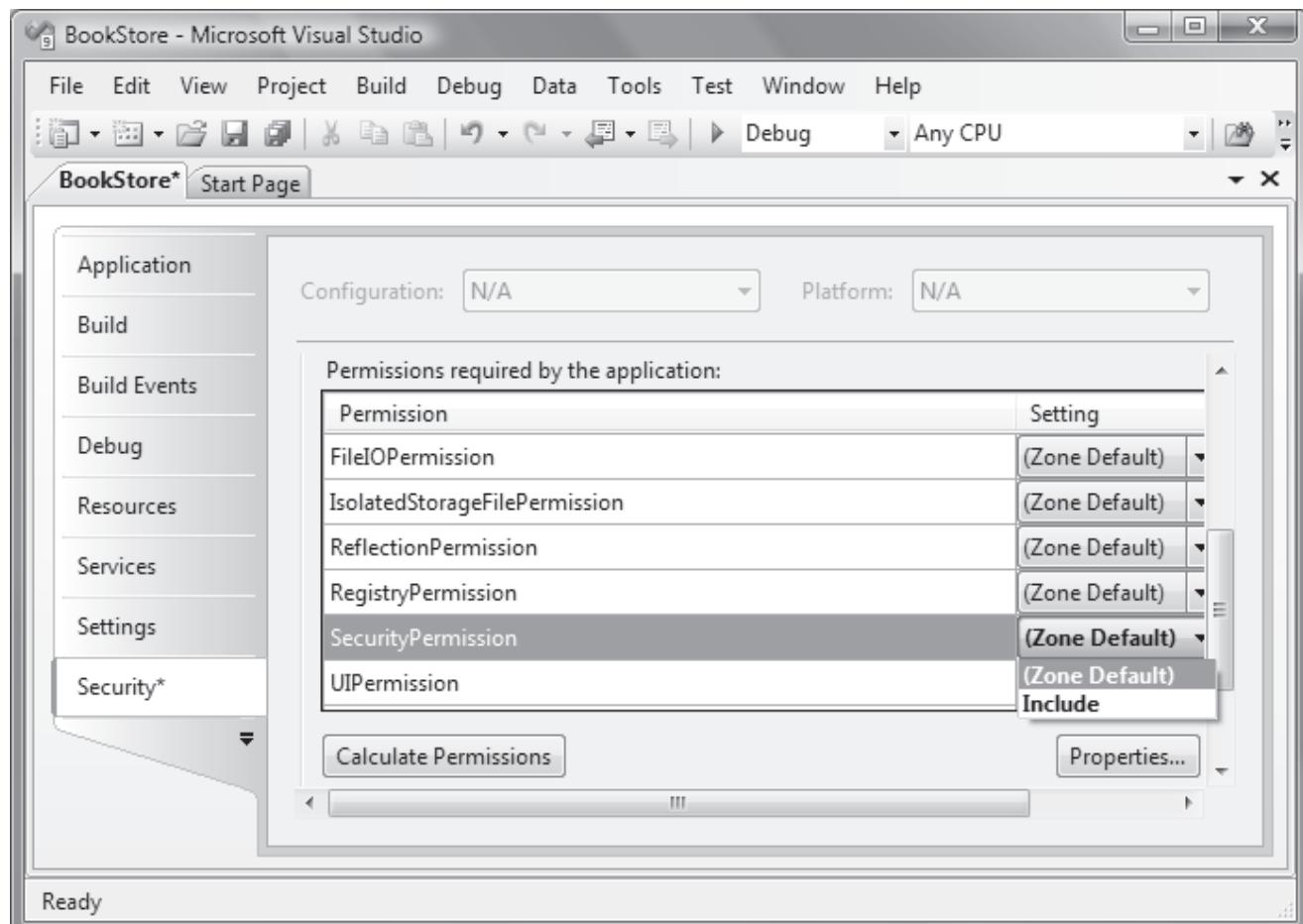


Figure 23.22: Setting SecurityPermission

24. Select the **Include** option. This will ensure the relevant security permissions (such as control over thread, application domain, and so forth) are granted.

When you republish the application, the version number changes automatically as seen in figure 23.23.

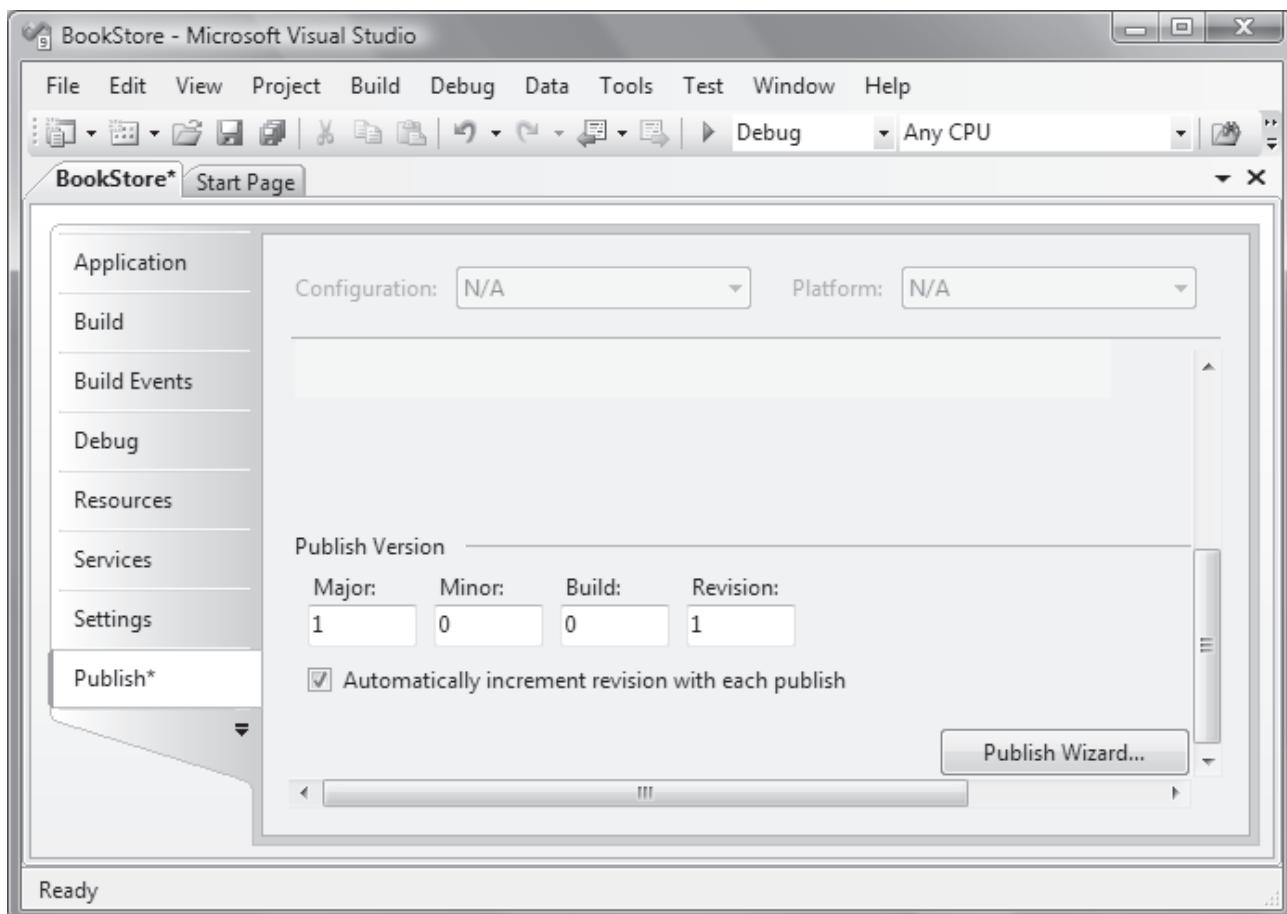


Figure 23.23: Change in Version Number

### Exercise

You are asked to create and publish a Word 2007 AddIn program using Visual Studio 2008.

### Solution:

#### Publishing a VSTO application

1. Create a new VSTO application by using File→New Project and then, select Office from project types.

2. Select Word 2007 Add-In as the template, and specify the name and location of the project as shown in figure 23.24.

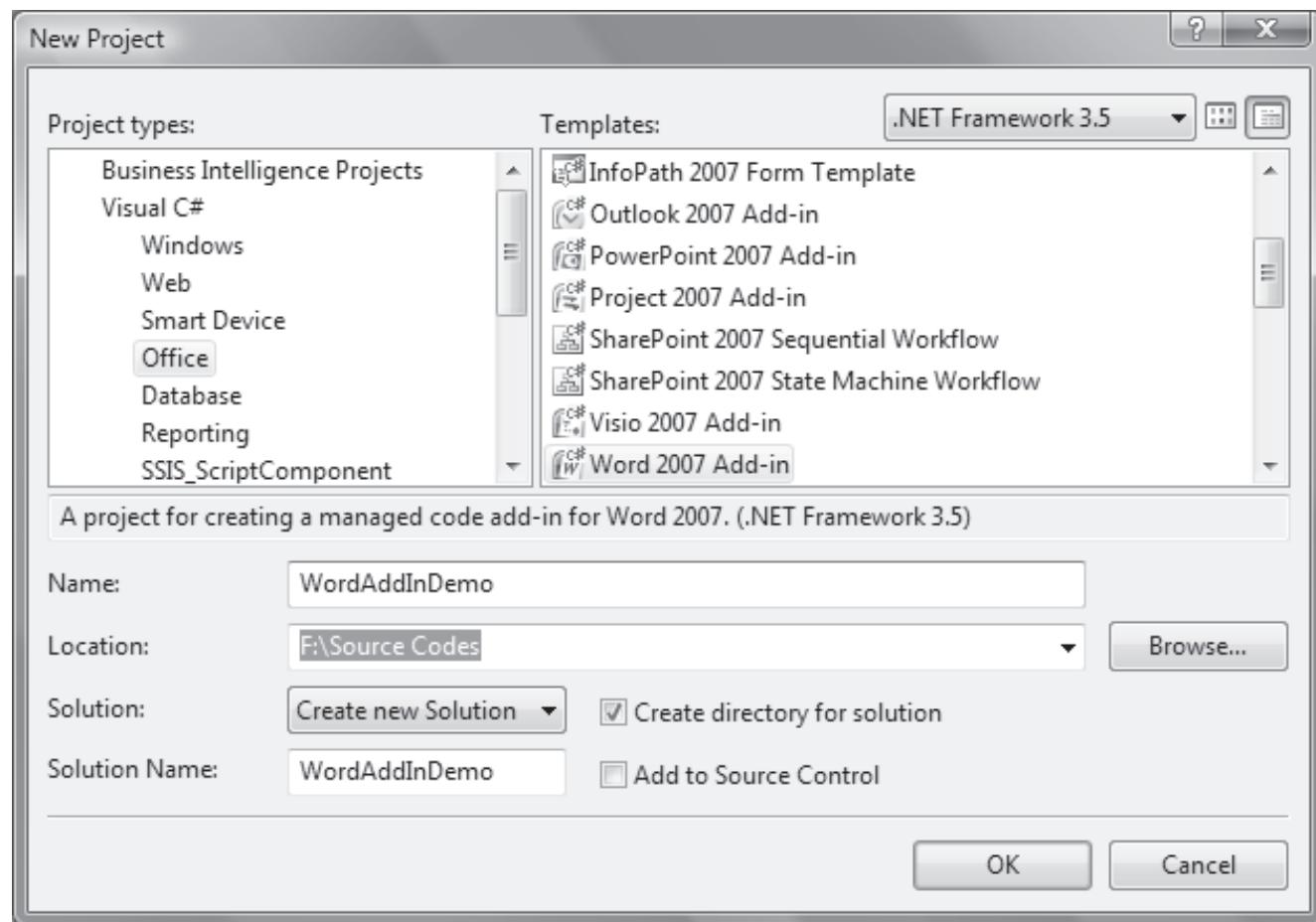
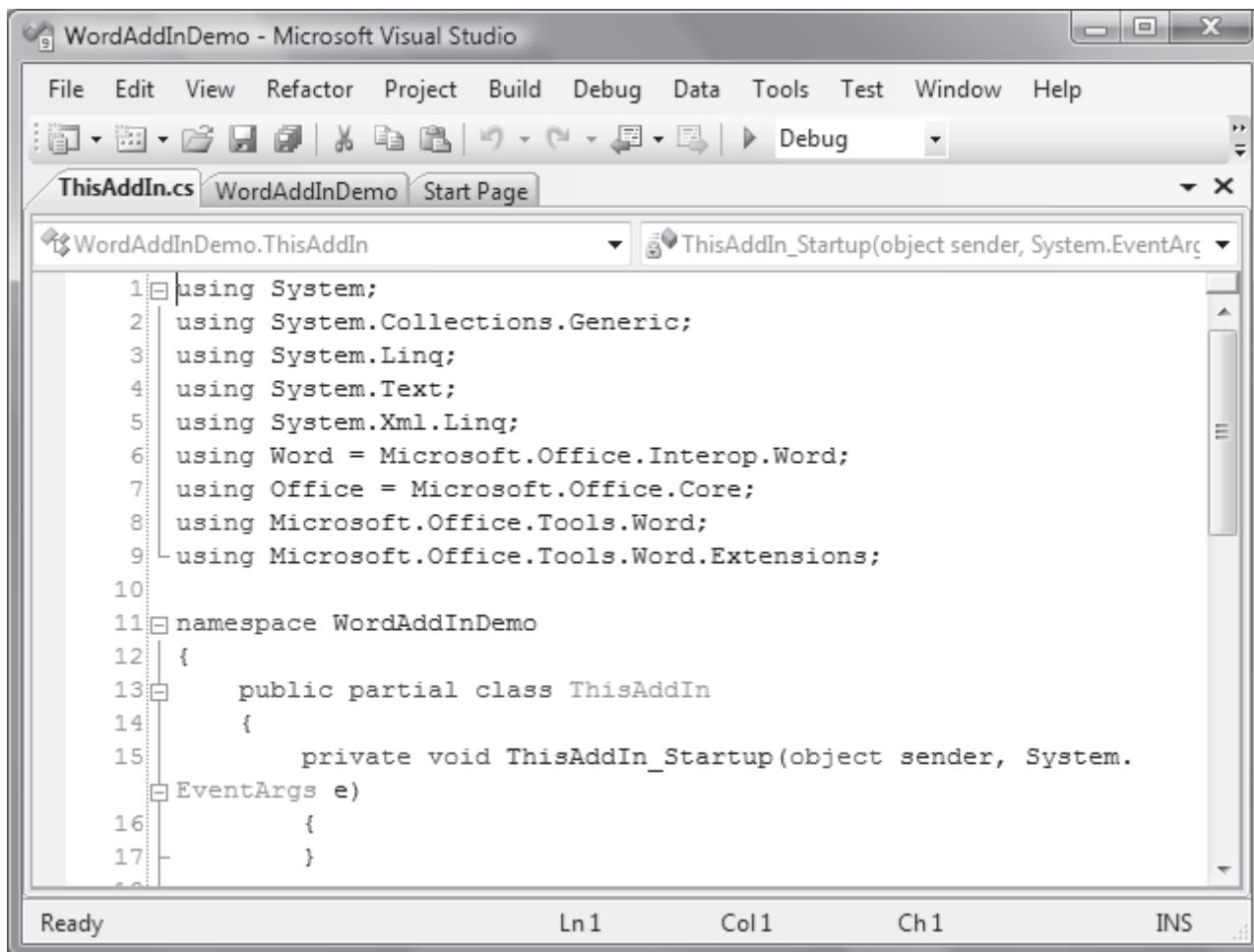


Figure 23.24: Creating VSTO application

3. Click OK. This creates the project and displays the code editor window as shown in figure 23.25.



The screenshot shows the Microsoft Visual Studio interface with the title bar "WordAddInDemo - Microsoft Visual Studio". The menu bar includes File, Edit, View, Refactor, Project, Build, Debug, Data, Tools, Test, Window, and Help. The toolbar below the menu bar has various icons for file operations like Open, Save, and Print. The status bar at the bottom shows "Ready", "Ln 1", "Col 1", "Ch 1", and "INS". The main code editor window displays the following C# code:

```
1  using System;
2  using System.Collections.Generic;
3  using System.Linq;
4  using System.Text;
5  using System.Xml.Linq;
6  using Word = Microsoft.Office.Interop.Word;
7  using Office = Microsoft.Office.Core;
8  using Microsoft.Office.Tools.Word;
9  using Microsoft.Office.Tools.Word.Extensions;
10
11 namespace WordAddInDemo
12 {
13     public partial class ThisAddIn
14     {
15         private void ThisAddIn_Startup(object sender, System.
16             EventArgs e)
17     }
}
```

Figure 23.25: VSTO application

Assume that you have added some functionality to the application and you want to publish it now.

4. Right-click the project name in the Solution Explorer and select Publish. The Publish Wizard is displayed.

5. Specify the location details where the application is to be published, as shown in figure 23.26.

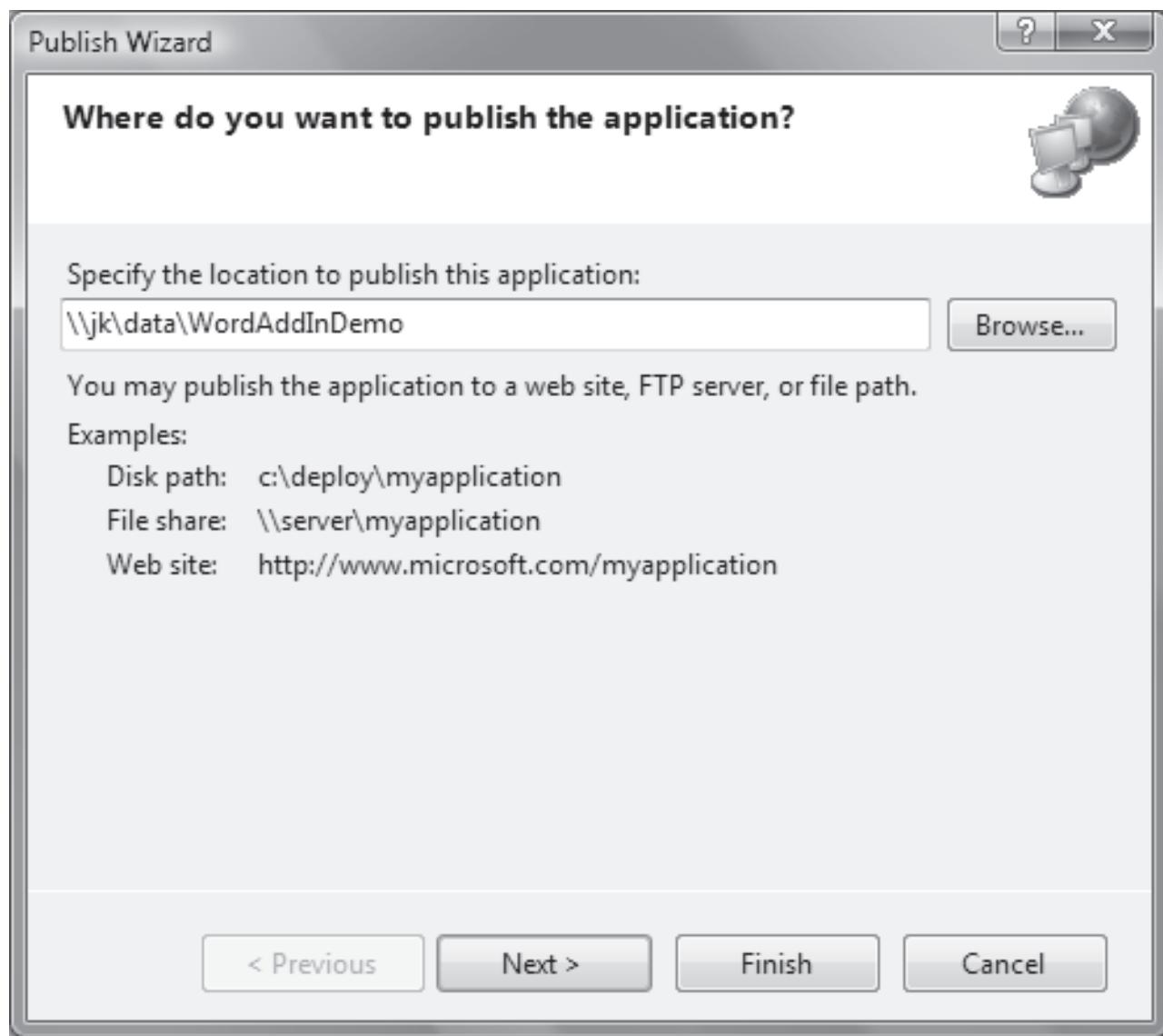


Figure 23.26: Specifying publish location

The next page asks you to specify the installation path as shown in figure 23.27.

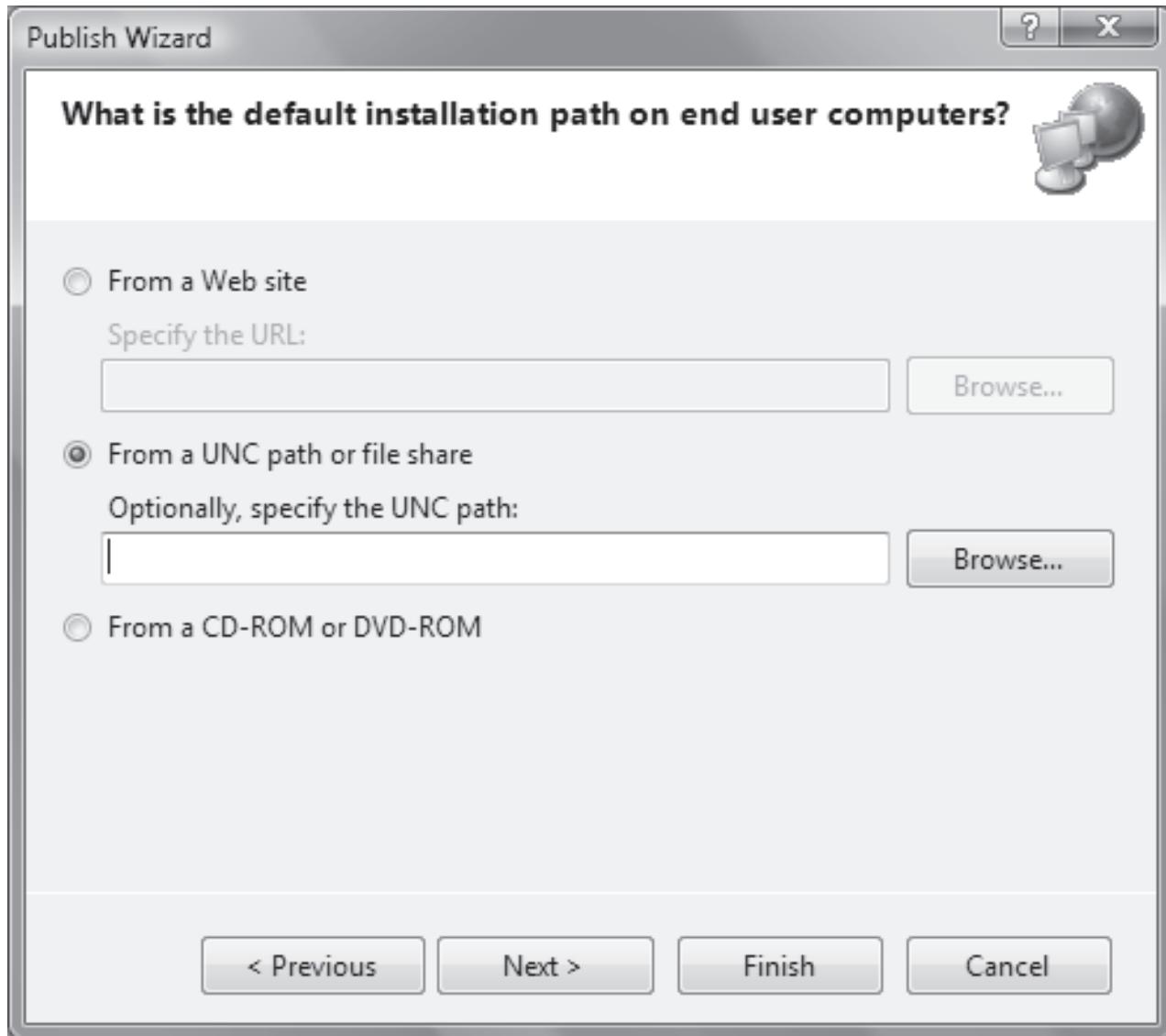


Figure 23.27: Specifying installation type

6. Specify the path as \\jk\data and click Next.

This completes the wizard and a message is displayed as seen in figure 23.28 to indicate that the application is ready to be published.

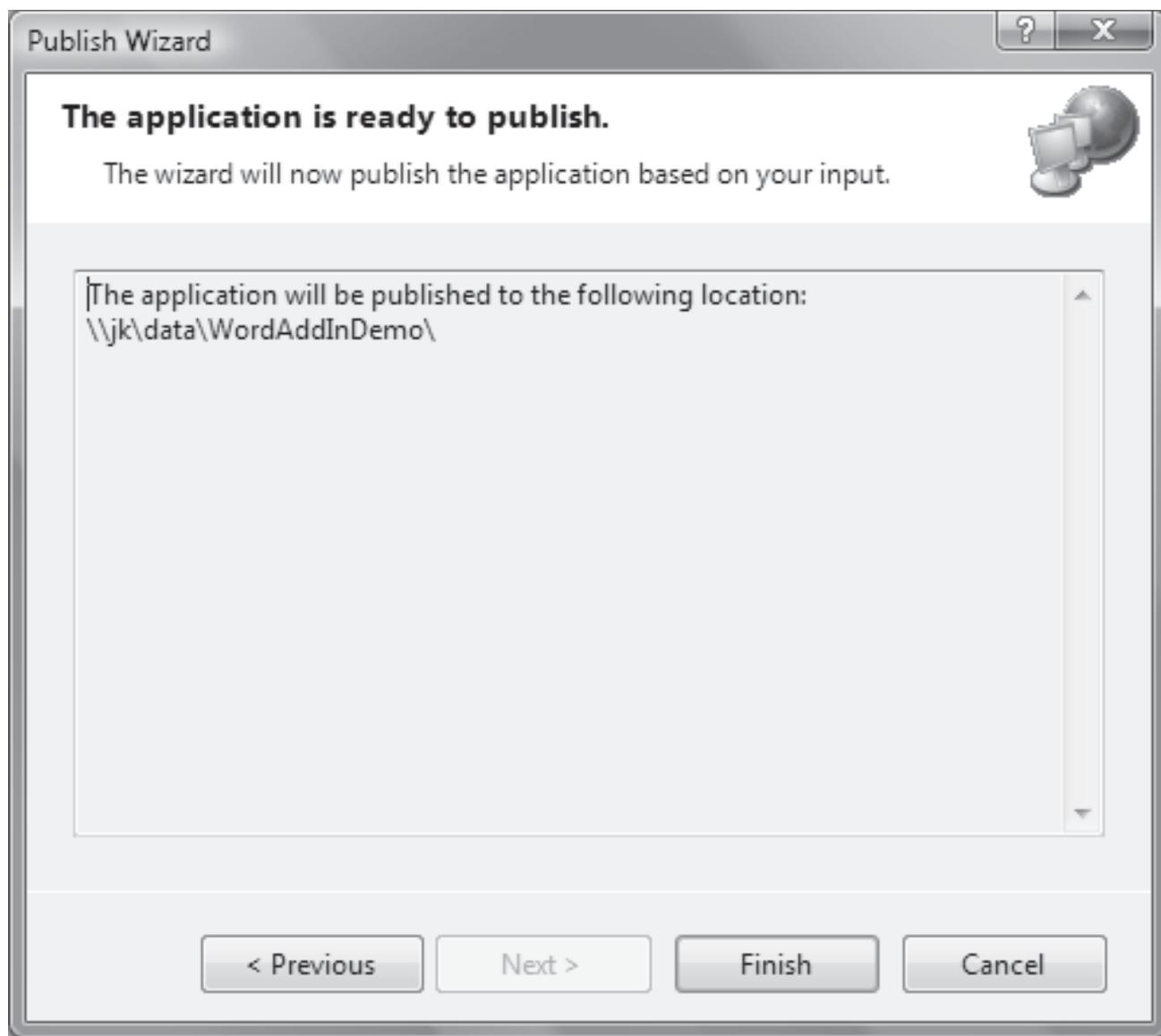


Figure 23.28: Final Page of Publish wizard

7. Click Finish.

On the file share machine, under the installation folder, you can observe a set of files as seen in figure 23.29.

Name	Type	Size
Application Files	File Folder	
Office2007PIARest	File Folder	
setup.exe	Application	992 KB
WordAddInDemo.vsto	VSTO Deployment...	6 KB

Figure 23.29: Setup files installed on client machine

8. Click **setup.exe** to begin installation.

## Part II 60 minutes

1. Create a setup project for the application developed in Lab Session 21.

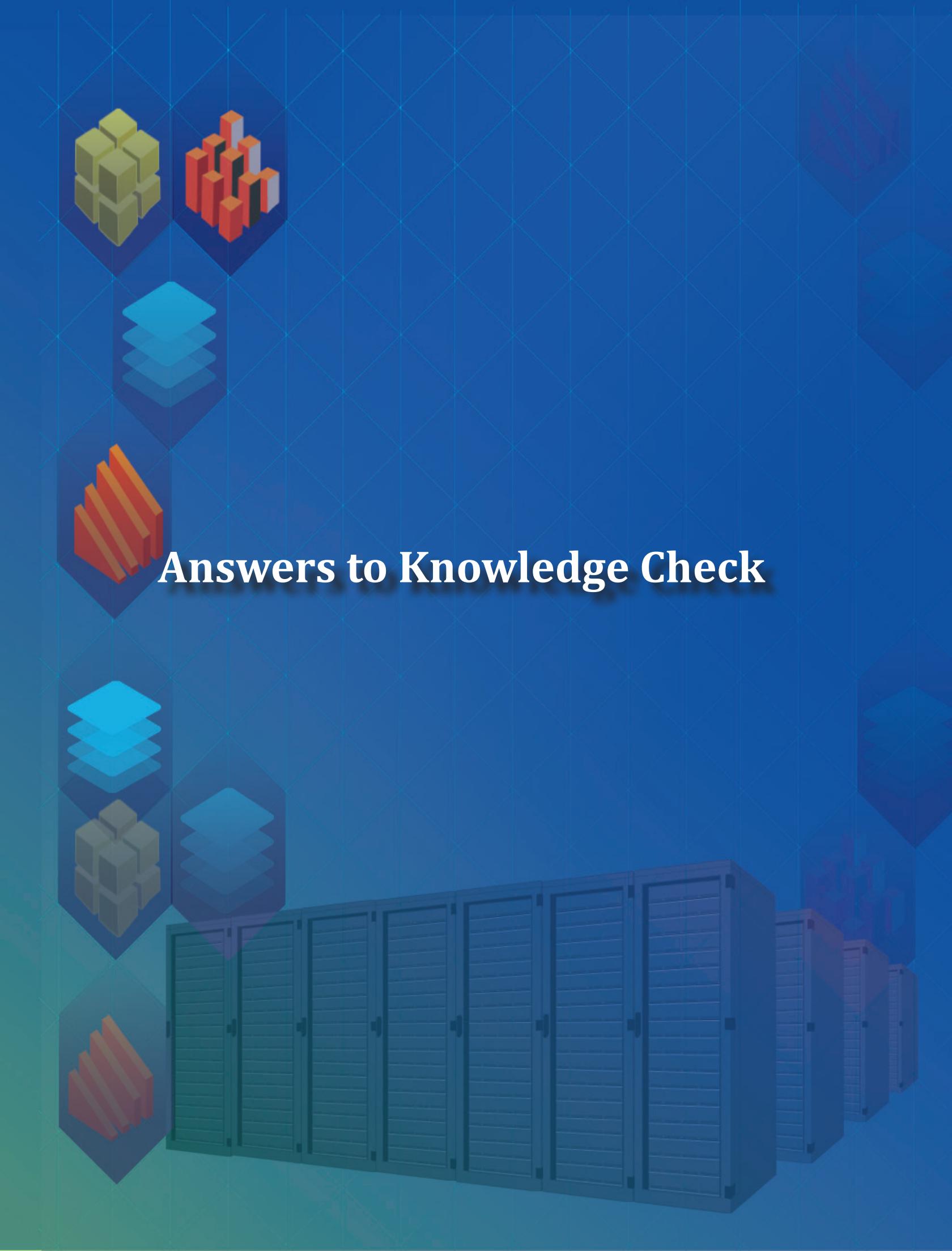
**Hints:**

- Select **Setup And Deployment** as the project type
- Click **Setup Project** in the Templates pane
- The setup project includes six editors to configure the contents and the behavior of the setup project. Make use of these to customize the setup



**Try it Yourself**

1. Publish and deploy the WorldCupFootBall WPF application created in Lab Session 19 using ClickOnce.



# Answers to Knowledge Check

## Module 1

### Knowledge Check 1

1.	D
2.	(A)-(4), (B)-(2), (C)-(5), (D)-(1), (E)-(3)

### Knowledge Check 2

1.	A
2.	(A)-(3), (B)-(1), (C)-(4), (D)-(5), (E)-(2)

### Knowledge Check 3

1.	C
2.	(A)-(5), (B)-(3), (C)-(1), (D)-(2), (E)-(4)
3.	(A)-(4), (B)-(2), (C)-(5), (D)-(1), (E)-(3)

### Knowledge Check 4

1.	(A)-(3), (B)-(1), (C)-(4), (D)-(5), (E)-(2)
2.	C

## Module 2

### Knowledge Check 1

1.	A-4, B-5, C-1, D-3, E-2
2.	B

**Knowledge Check 2**

1.	A, B, E
2.	A-4, B-2, C-5, D-2, E-3

**Knowledge Check 3**

1.	A-3, B-4, C-2, D-5, E-1
2.	B

**Knowledge Check 4**

1.	A, C
2.	B

**Module 3****Knowledge Check 1**

1.	C
2.	A-5, B-1, C-1, D-2, E-4

**Knowledge Check 2**

1.	A, B
2.	A, C, D,H

**Knowledge Check 3**

1.	A-2, B-1, C-4, D-5, E-3
2.	B

**Knowledge Check 4**

1.	C
2.	A, B, C, D

**Module 4****Knowledge Check 1**

1.	A, D
2.	A-4, B-3, C-2, D-1, E

**Knowledge Check 2**

1.	A-2, B-3, C-5, D-1, E-4
2.	B

**Knowledge Check 3**

1.	A, C, E
2.	D

**Module 5****Knowledge Check 1**

1.	C, D
----	------

**Knowledge Check 2**

1.	A
2.	A, C, D

**Knowledge Check 3**

1.	A, E, G, H
2.	A, B

**Knowledge Check 4**

1.	D
----	---

**Module 6****Knowledge Check 1**

1.	C, D
2.	A-3, B-5, C-4, D-2, E-1

**Knowledge Check 2**

1.	A, B, D
2.	A-5, B-1, C-4, D-3, E-2

**Knowledge Check 3**

1.	A, B
2.	A-4, B-3, C-5, D-1, E-2

**Knowledge Check 4**

1.	C
----	---

## *Module 7*

### **Knowledge Check 1**

1.	A-4, B-1, C-5, D-2, E-3
2.	A, C

### **Knowledge Check 2**

1.	A, C, D
2.	A-5, B-4, C-1, D-2, E-3

### **Knowledge Check 3**

1.	A-4, B-3, C-5, D-1, E-2
2.	C

## *Module 8*

### **Knowledge Check 1**

1.	A, C, E
----	---------

### **Knowledge Check 2**

1.	D
2.	C

### **Knowledge Check 3**

1.	A, C, D
2.	A-4, B-5, C-2, D-3, E-1

**Knowledge Check 4**

1.	A, B, D
2.	A-4, B-5, C-1, D-2, E-3

**Module 9****Knowledge Check 1**

1.	B
2.	A, D

**Knowledge Check 2**

1.	A-2, B-3, C-1, D-5, E-4
2.	A-4, B-3, C-1, D-2, E-5

**Knowledge Check 3**

1.	A-3, B-5, C-1, D-2, E-4
2.	A

**Knowledge Check 4**

1.	A, D, E
----	---------

**Module 10****Knowledge Check 1**

1.	A-5, B-4, C-1, D-2, E-3
2.	C, D, E

**Knowledge Check 2**

1.	B, C
2.	A

**Knowledge Check 3**

1.	B, D
2.	C

**Knowledge Check 4**

1.	D
----	---

**Module 11****Knowledge Check 1**

1.	C, E
----	------

**Knowledge Check 2**

1.	B, C
2.	A-4, B-5, C-1, D-2, E-3

**Knowledge Check 3**

1.	A, D, E
2.	A

**Knowledge Check 4**

1.	A-5, B-4, C-1, D-2, E-3
2.	C

**Knowledge Check 5**

1.	B, D
2.	A

**Module 12****Knowledge Check 1**

1.	C, E
----	------

**Knowledge Check 2**

1.	A-5, B-4, C-1, D-3, E-2
2.	B

**Knowledge Check 3**

1.	A, B, D
----	---------

**Knowledge Check 4**

1.	C, D, E
2.	B

**Module 13****Knowledge Check 1**

1.	A-3, B-4, C-1, D-5, E-2
----	-------------------------

**Knowledge Check 2**

1.	A, C, D
----	---------

**Knowledge Check 3**

1.	B
2.	A, D, E

**Knowledge Check 4**

1.	A, E
2.	A-2, B-1, C-4, D-5, E-3

**Module 14****Knowledge Check 1**

1.	A-5, B-4, C-1, D-2, E-3
2.	A, B
3.	C

**Knowledge Check 2**

1.	C, E
2.	A

**Knowledge Check 3**

1.	A, B
2.	B
3.	A-3, B-5, C-4, D-2, E-1

**Module 15****Knowledge Check 1**

1.	A, C, E
2.	A-2, B-4, C-5, D-1, E-3

**Knowledge Check 2**

1.	B, E
----	------

**Knowledge Check 3**

1.	A, B, E
2.	A-2, B-4, C-5, D-1, E-3

**Knowledge Check 4**

1.	B, D, E
2.	A-2, B-3, C-1, D-5, E-4

**Knowledge Check 5**

1.	A, D
----	------

**Module 16****Knowledge Check 1**

1.	A
2.	B
3.	C
4.	C
5.	B
6.	B
7.	A, C
8.	C

**Module 18****Knowledge Check 1**

1.	B, C
2.	C
3.	A, D
4.	A-2, B-5, C-1, D-3, E-4
5.	B
6.	B, D
7.	A, B
8.	A-3, B-1, C-5, D-4, E-2
9.	A

## *Module 20*

### **Knowledge Check 1**

1.	B, C
2.	B
3.	C
4.	A, B, D
5.	A
6.	A
7.	A

## *Module 22*

### **Knowledge Check 1**

1.	A, B, C
2.	A, C
3.	D
4.	C
5.	A
6.	B



Visit the  
**Frequently Asked Questions**  
section @