msdn® training

# Developing Microsoft® .NET Applications for Windows® (Visual C#™ .NET).

## Delivery Guide

Course Number: 2555A

Part Number: X08-82469
Released: 05/2002

Microsoft®

Course Number: 2555A
Part Number: X08-82469
Released: 05/2002

# Contents

## Introduction

## Module 1: Introducing Windows Forms

## Module 2: Working with Controls

## Module 3: Building Controls

## Module 4: Using Data in Windows Forms Applications

## Module 5: Interoperating with Managed Objects

## Module 6: Printing and Reporting in Windows Forms Applications

## Module 7: Asynchronous Programming

## Module 8: Enhancing the Usability of Applications

## Module 9: Deploying Windows Forms Applications

## Module 10: Securing Windows Forms Applications

## Appendix A: Using Filled Shapes and Images

# About This Course

This section provides you with a brief description of the course, audience, suggested prerequisites, and course objectives.

## Description

Windows Forms is the new platform for Microsoft® Windows® application development, based on the Microsoft .NET Framework. The .NET Framework provides a clear, object-oriented, extensible set of classes that enables developers to develop rich Windows Forms applications. Additionally, Windows Forms can act as the local user interface in a multi-tier distributed solution.

This three-day instructor-led course provides students with the skills required to build Windows Forms applications by using the .NET Framework.

## Audience

This course is intended for the intermediate programmer who is responsible for designing and building Windows Forms applications by using the .NET Framework. It is designed for developers who have Microsoft Visual C#™ .NET development skills.

Typically, these individuals perform the following key activities:

- Help with creation of functional specifications
- Design and develop user interfaces
- Create and test prototypes
- Write Windows Forms applications

## Student prerequisites

This course requires that students meet the following prerequisites:

- Experience with a .NET Framework language such as Visual C# .NET

  Microsoft MSDN® Training Course 2609A: *Introduction to C# Programming with Microsoft .NET* and Course 2124C: *Programming with C#* will help students gain skills in Visual C# .NET programming techniques and meet the prerequisites for this course.
- Experience developing applications

## Course objectives

After completing this course, the student will be able to:

- Create and populate Windows Forms.
- Organize controls on Windows Forms.
- Create menus in a Windows Forms application.
- Add code to form and control event procedures in a Windows Forms application.
- Create Multiple Document Interface (MDI) applications.
- Use dialog boxes in Windows Forms applications.
- Validate user input in a Windows Forms application.
- Create and use user controls in a Windows Forms application.
- Create licenses for controls.
- Bind Windows Forms applications to various data sources by using Microsoft ADO.NET.

- Consume XML Web services from Windows Forms applications.
- Use .NET and COM components in a Windows Forms application.
- Call Microsoft Win32® APIs from a Windows Forms application.
- Print documents in a Windows Forms application.
- Make asynchronous calls to methods from a Windows Forms application.
- Debug a Windows Forms application.
- Incorporate accessibility features in a Windows Forms application.
- Localize a Windows Forms application.
- Add support for help to localize a Windows Forms application.
- Create help files in a Windows Forms application.
- Deploy a Windows Forms application.
- Implement code access and role-based security in a Windows Forms application.
- Add deployment flexibility to applications by using shared assemblies.

# Course Timing

The following schedule is an estimate of the course timing. Your timing may vary.

## Day 1

| Start | End | Module |
|---|---|---|
| 9:00 | 9:30 | Introduction |
| 9:30 | 11:00 | Module 1: Introducing Windows Forms |
| 11:00 | 11:10 | Break |
| 11:10 | 11:40 | Lab 1.1: Creating Windows Forms |
| 11:40 | 12:30 | Lunch |
| 12:30 | 2:30 | Module 2: Working with Controls |
| 2:30 | 2:45 | Break |
| 2:45 | 3:15 | Lab 2.1: Working with Controls |
| 3:15 | 4:15 | Module 3: Building Controls |
| 4:15 | 4:45 | Lab 3.1: Building Controls |

## Day 2

| Start | End | Module |
|---|---|---|
| 8:30 | 9:00 | Day 1 review |
| 9:00 | 11:00 | Module 4: Using Data in Windows Forms Applications |
| 11:00 | 11:15 | Break |
| 11:15 | 11:45 | Module 4: Using Data in Windows Forms Applications (*continued*) |
| 11:45 | 12:30 | Lab 4.1: Accessing Data by Using ADO.NET |
| 12:30 | 1:00 | Lunch |
| 1:00 | 1:30 | Module 4: Using Data in Windows Forms Applications (*continued*) |
| 1:30 | 1:45 | Lab 4.2: Calling an XML Web Service |
| 1:45 | 2:00 | Break |
| 2:00 | 3:00 | Module 5: Interoperating with Managed Objects |
| 3:00 | 3:30 | Lab 5.1: Interoperating with COM and Calling Win32 APIs |
| 3:30 | 5:00 | Module 6: Printing and Reporting in Windows Forms Applications |
| 5:00 | 5:45 | Lab 6.1: Printing Formatted Documents |

## Day 3

| Start | End | Module |
|---|---|---|
| 8:30 | 10:00 | Module 7: Asynchronous Programming |
| 10:00 | 10:15 | Lab 7.1: Making Asynchronous Calls to an XML Web Service |
| 10:15 | 10:30 | Break |
| 10:30 | 11:30 | Module 8: Enhancing the Usability of Applications |
| 11:30 | 12:15 | Lunch |
| 12:15 | 12:45 | Lab 8.1: Enhancing the Usability of an Application |
| 12:45 | 2:45 | Module 9: Deploying Windows Forms Applications |
| 2:45 | 3:00 | Break |
| 3:00 | 3:30 | Lab 9.1: Deploying an Application |
| 3:30 | 4:45 | Module 10: Securing Windows Forms Applications |
| 4:45 | 5:15 | Lab 10.1: Adding and Testing Permission Requests |

# Trainer Materials Compact Disc Contents

The Trainer Materials compact disc contains the following files and folders:

- *Autorun.exe.* When the compact disc is inserted into the compact disc drive, or when you double-click the **Autorun.exe** file, this file opens the compact disc and allows you to browse the Student Materials or Trainer Materials compact disc.
- *Autorun.inf.* When the compact disc is inserted into the compact disc drive, this file opens Autorun.exe.
- *Default.htm*. This file opens the Trainer Materials Web page.
- *Readme.txt*. This file explains how to install the software for viewing the Trainer Materials compact disc and its contents and how to open the Trainer Materials Web page.
- *2555A_ms.doc.* This file is the Manual Classroom Setup Guide. It contains the steps for manually setting up the classroom computers.
- *2555A_sg.doc*. This file is the Automated Classroom Setup Guide. It contains a description of classroom requirements, classroom configuration, instructions for using the automated classroom setup scripts, and the Classroom Setup Checklist.
- *Powerpnt*. This folder contains the Microsoft PowerPoint® slides that are used in this course.
- *Ppview.* This folder contains the Microsoft PowerPoint Viewer 97, which can be used to display the PowerPoint slides if Microsoft PowerPoint 2002 is not available. Do not use this version in the classroom.
- *Setup*. This folder contains the files that install the course and related software to computers in a classroom setting.
- *StudentCD.* This folder contains the Web page that provides students with links to resources pertaining to this course, including additional reading, review and lab answers, lab files, multimedia presentations, and course-related Web sites.
- *Tools*. This folder contains files and utilities used to complete the setup of the instructor computer.
- *Webfiles*. This folder contains the files that are required to view the course Web page. To open the Web page, open Windows Explorer, and in the root directory of the compact disc, double-click **Default.htm** or **Autorun.exe**.

# Student Materials Compact Disc Contents

The Student Materials compact disc contains the following files and folders:

- *Autorun.exe.* When the compact disc is inserted into the CD-ROM drive, or when you double-click the **Autorun.exe** file, this file opens the compact disc and allows you to browse the Student Materials compact disc.
- *Autorun.inf.* When the compact disc is inserted into the compact disc drive, this file opens Autorun.exe.
- *Default.htm*. This file opens the Student Materials Web page. It provides you with resources pertaining to this course, including additional reading, review and lab answers, lab files, multimedia presentations, and course-related Web sites.
- *Readme.txt*. This file explains how to install the software for viewing the Student Materials compact disc and its contents and how to open the Student Materials Web page.
- *2555A_ms.doc*. This file is the Manual Classroom Setup Guide. It contains a description of classroom requirements, classroom setup instructions, and the classroom configuration.
- *Democode*. This folder contains files that are used in the instructor demonstrations.
- *Flash.* This folder contains the installer for the Macromedia Flash 5.0 browser plug-in.
- *Fonts*. This folder contains fonts that may be required to view Microsoft Word documents that are included with this course.
- *Inetpubs*. This folder contains files used by the sample Web-based applications in this course.
- *Labfiles*. This folder contains files that are used in the hands-on labs. These files may be used to prepare the student computers for the hands-on labs.
- *Media*. This folder contains files that are used in multimedia presentations for this course.
- *Mplayer*. This folder contains the setup file to install Microsoft Windows Media™ Player.
- *Practices*. This folder contains files that are used in the hands-on practices.
- *Sampapps*. This folder contains the sample applications associated with this course.
- *Sampcode.* This folder contains sample code that is accessible through the Web pages on the Student Materials compact disc.
- *Webfiles.* This folder contains the files that are required to view the course Web page. To open the Web page, open Windows Explorer, and in the root directory of the compact disc, double-click **Default.htm** or **Autorun.exe**.
- *Wordview*. This folder contains the Word Viewer that is used to view any Microsoft Word document (.doc) files that are included on the compact disc.

# Document Conventions

The following conventions are used in course materials to distinguish elements of the text.

| Convention | Use |
|---|---|
| **Bold** | Represents commands, command options, and syntax that must be typed exactly as shown. It also indicates commands on menus and buttons, dialog box titles and options, and icon and menu names. |
| *Italic* | In syntax statements or descriptive text, indicates argument names or placeholders for variable information. Italic is also used for introducing new terms, for book titles, and for emphasis in the text. |
| Title Capitals | Indicate domain names, user names, computer names, directory names, and folder and file names, except when specifically referring to case-sensitive names. Unless otherwise indicated, you can use lowercase letters when you type a directory name or file name in a dialog box or at a command prompt. |
| ALL CAPITALS | Indicate the names of keys, key sequences, and key combinations—for example, ALT+SPACEBAR. |
| `monospace` | Represents code samples or examples of screen text. |
| [ ] | In syntax statements, enclose optional items. For example, [*filename*] in command syntax indicates that you can choose to type a file name with the command. Type only the information within the brackets, not the brackets themselves. |
| { } | In syntax statements, enclose required items. Type only the information within the braces, not the braces themselves. |
| \| | In syntax statements, separates an either/or choice. |
| ► | Indicates a procedure with sequential steps. |
| ... | In syntax statements, specifies that the preceding item may be repeated. |
| .<br>.<br>. | Represents an omitted portion of a code sample. |

# Introduction

**Contents**

# Instructor Notes

**Presentation:**
**30 minutes**

The Introduction module provides students with an overview of the course content, materials, and logistics for Course 2555A, *Developing Microsoft .NET Applications for Windows® (Visual C#™ .NET)*.

**Required materials**

To teach this course, you need the following materials:

- Delivery Guide
- Trainer Materials compact disc

## How to Teach This Module

This section contains information that will help you to teach this module.

### Introduction

Welcome students to the course and introduce yourself. Provide a brief overview of your background to establish credibility.

Ask students to introduce themselves and provide their background, product experience, and expectations of the course.

Record student expectations on a whiteboard or flip chart that you can reference later in class.

### Course materials

Tell students that everything they will need for this course is provided at their desk.

Have students write their names on both sides of the name card.

Describe the contents of the student workbook and the Student Materials compact disc.

Tell students where they can send comments and feedback on this course.

Demonstrate how to open the Web page provided on the Student Materials compact disc by double-clicking **Autorun.exe** or **Default.htm** in the StudentCD folder on the Trainer Materials compact disc.

### Prerequisites

Describe prerequisites for this course. This is an opportunity for you to identify students who may not have the appropriate background or experience to attend this course.

### Course outline

Briefly describe each module and what students will learn.

Explain how this course will meet students' expectations by relating the information covered in individual modules to their expectations.

### Demonstration: Expense Report Application

It is very important that you go through this demonstration before teaching the rest of the course. The Expense Report application is one of two primary scenarios for practices and labs throughout the course. Demonstrating the Expense Report application is also a good way to introduce students to the skills that they will acquire in the course. If a student arrives after you have done the demonstration, have the student review the steps for this demonstration before doing any of the labs.

### Demonstration: Purchase Order Application

It is very important that you go through this demonstration before teaching the rest of the course. The Purchase Order application is one of two primary scenarios for practices and labs throughout the course. Demonstrating the Purchase Order application is also a good way to introduce students to the skills that they will acquire in the course. If a student arrives after you have done the demonstration, have the student review the steps for this demonstration before doing any of the labs.

### Microsoft Certified Professional program

Inform students about the Microsoft Certified Professional (MCP) program and the various certification options.

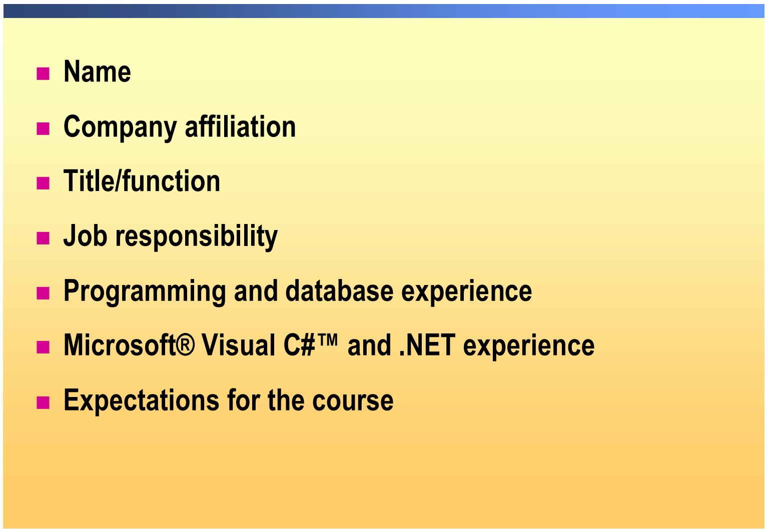
### Facilities

Explain the class hours, extended building hours for labs, parking, restroom location, meals, phones, message posting, and where smoking is or is not allowed.

Let students know if your facility has Internet access that is available for them to use during class breaks.

Also, make sure that the students are aware of the recycling program if one is available.

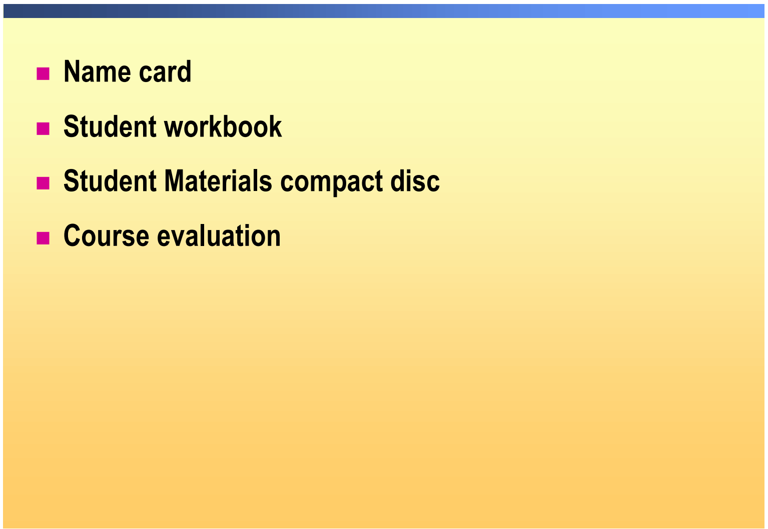
# Introduction



- Name
- Company affiliation
- Title/function
- Job responsibility
- Programming and database experience
- Microsoft® Visual C#™ and .NET experience
- Expectations for the course

Your instructor will ask you to introduce yourself and provide a brief overview of your background, addressing the bulleted items on the slide as appropriate.

# Course Materials



- Name card
- Student workbook
- Student Materials compact disc
- Course evaluation

The following materials are included with your kit:

- *Name card.* Write your name on both sides of the name card.
- *Student workbook.* The student workbook contains the material covered in class, in addition to the hands-on lab exercises.
- *Student Materials compact disc.* The Student Materials compact disc contains the Web page that provides you with links to resources pertaining to this course, including additional readings, review and lab answers, lab files, multimedia presentations, and course-related Web sites.

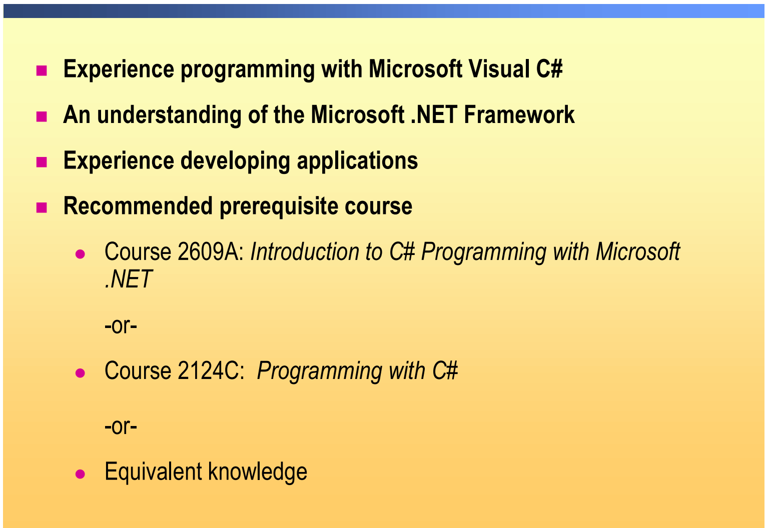
  **Note** To open the Web page, insert the Student Materials compact disc into the CD-ROM drive, and then in the root directory of the compact disc, double-click **Autorun.exe** or **Default.htm**.

  **Important** There are starter and solution files associated with the labs in this course. If you perform a default installation, the starter and the solution files install to C:\Program Files\Msdntrain\2555. However, if you install to a different location, you must reset the assembly references in the starter and solution projects.

- *Course evaluation.* To provide feedback on the course, training facility, and instructor, you will have the opportunity to complete an online evaluation near the end of the course.

  To provide additional comments or inquire about the Microsoft Certified Professional program, send e-mail to mcphelp@microsoft.com.

# Prerequisites



- **Experience programming with Microsoft Visual C#**
- **An understanding of the Microsoft .NET Framework**
- **Experience developing applications**
- **Recommended prerequisite course**
  - Course 2609A: *Introduction to C# Programming with Microsoft .NET*

    -or-
  - Course 2124C: *Programming with C#*

    -or-
  - Equivalent knowledge

This course requires that you meet the following prerequisites:

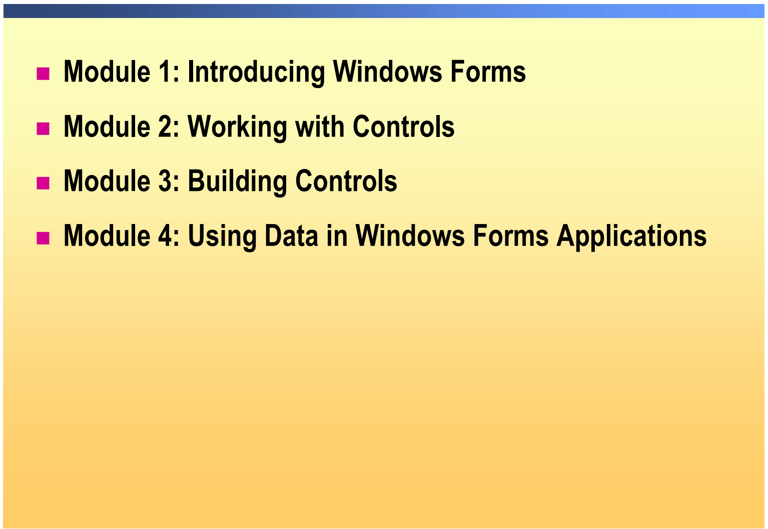
- Experience programming with Microsoft Visual C#
- An understanding of the Microsoft .NET Framework
- Experience developing applications

  –and–
- Completion of Microsoft MSDN® Training Course 2609A, *Introduction to C# Programming with Microsoft .NET*

  –or–
- Completion of MSDN Training Course 2124C, *Programming with C#*

  –or–
- Equivalent knowledge

# Course Outline



- Module 1: Introducing Windows Forms
- Module 2: Working with Controls
- Module 3: Building Controls
- Module 4: Using Data in Windows Forms Applications

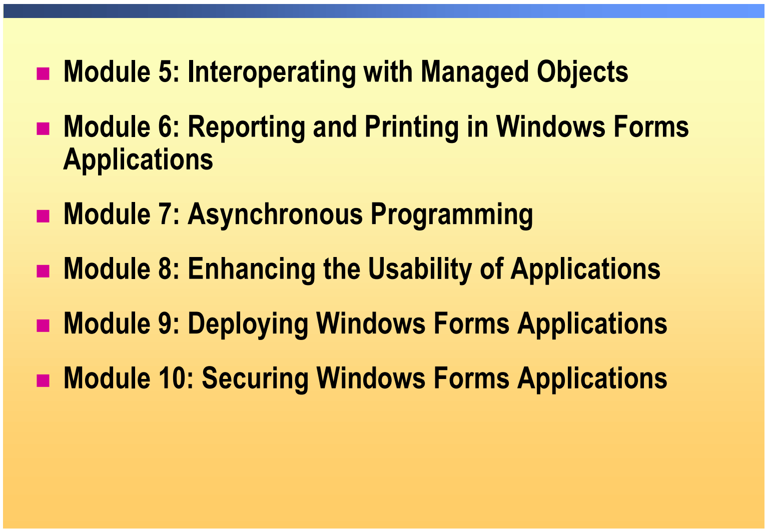
Module 1, "Introducing Windows Forms," introduces Windows Forms and controls, which are part of the Microsoft .NET framework. It explains how to create and populate base forms and inherited forms by using Microsoft Visual Studio® .NET. It also covers organizing controls on a form and creating Multiple Document Interface (MDI) applications. This module is meant mainly to be a review of concepts that you are familiar with but also presents some new concepts, such as how to use Visual Studio .NET tools for organizing controls on a Windows form.

Module 2, "Working with Controls," explains how to code for event procedures associated with different controls. The module covers how to use some of the Windows Forms intrinsic controls in an application. It also explains how to use dialogs, validation controls, and menus in a Windows Forms application. The module also includes a section on the controls collection and how to add controls at run time.

Module 3, "Building Controls," describes the options for building your own controls. It explains how to extend the functionality of an existing Windows Forms control, combine multiple existing controls into a composite control, and build a new custom control. It also covers how to add design-time attributes and licensing support to a control.

Module 4, "Using Data in Windows Forms Applications," describes how to bind Windows Forms to various data sources by using Microsoft ADO.NET and the Bindings collection. The module also provides an overview of the XML Web services programming model and covers how to create applications that use XML Web services. The module also provides an overview of how to persist data to and read data from files and isolated storage.

# Course Outline *(continued)*



- Module 5: Interoperating with Managed Objects
- Module 6: Reporting and Printing in Windows Forms Applications
- Module 7: Asynchronous Programming
- Module 8: Enhancing the Usability of Applications
- Module 9: Deploying Windows Forms Applications
- Module 10: Securing Windows Forms Applications

Module 5, “Interoperating with Managed Objects,” explains how to use .NET and COM components in your Windows Forms application. You will also learn how to call Microsoft Win32® APIs in your Windows Forms application.

Module 6, “Reporting and Printing in Windows Forms Applications,” explains how to create reports in a Windows Forms application by using Crystal Reports. The module also covers how to implement printing in a Windows Forms application.

Module 7, “Asynchronous Programming,” explains how to use the techniques of asynchronous programming and multithreading to avoid blocking the user interface of an application.

Module 8, “Enhancing the Usability of Applications,” explains how to use the accessibility, Help, and localization features available in the .NET Framework.

Module 9, “Deploying Windows Forms Applications,” explains assemblies and the use of strong-named assemblies and the global assembly cache in the .NET Framework. It also covers how to configure and deploy your Windows Forms applications.

Module 10, “Securing Windows Forms Applications,” explains how to use these security features in the .NET Framework in Windows Forms applications.

# Demonstration: Expense Report Application

In this demonstration, you will see how to use the Expense Report application

**Introduction**

In this demonstration, you will see how to use the Expense Report application.

**Note** If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

**Instructions**

► **To run the demonstration**

1. Open the InternalBusinessApp.sln solution file from *install_folder*\Sampapps\Business Application Shell.
2. Mention that the business expense reporting application consists of two major parts:
   - A Windows Forms-based client application that resides on the user's local computer.
   - An XML Web services component that returns information about existing expense reports and receives and stores information for newly submitted expense reports.

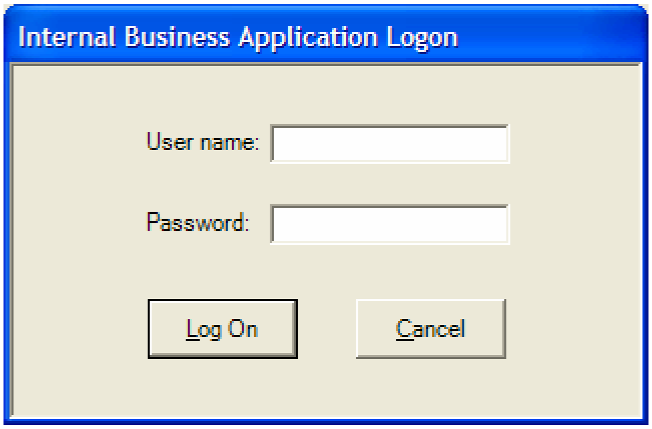
   All users of the application can create, submit, and view their own expense reports. Users who are managers can also view expense reports for their reporting employees and approve or disapprove those reports.

   While disconnected from the network, users can create new reports or view saved reports. Users must be connected to the network to submit reports or retrieve information about reports that do not reside on their local computer.

3. Describe the user interface.

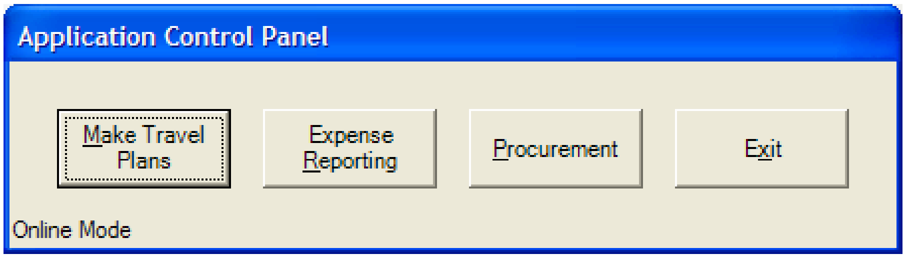
   The user interface of the client application consists of several forms.

   The main control panel form for the Business application will display a **Logon** form in its **Load** event handler.



4. Log on to the application. Specify **mario** for the user name and **P@ssw0rd** for the password.

   After the user has successfully logged on, the main control panel form for the Business application appears.

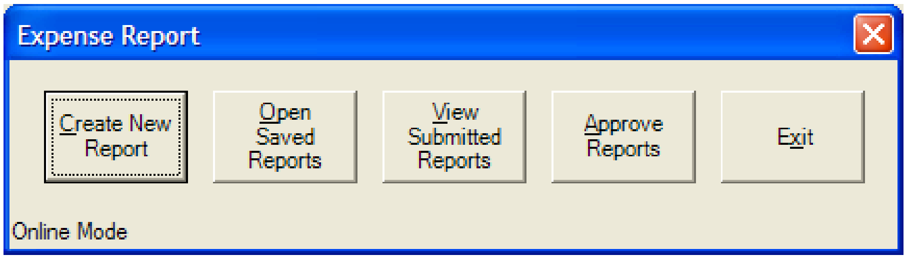


5. Describe the buttons on the main page.

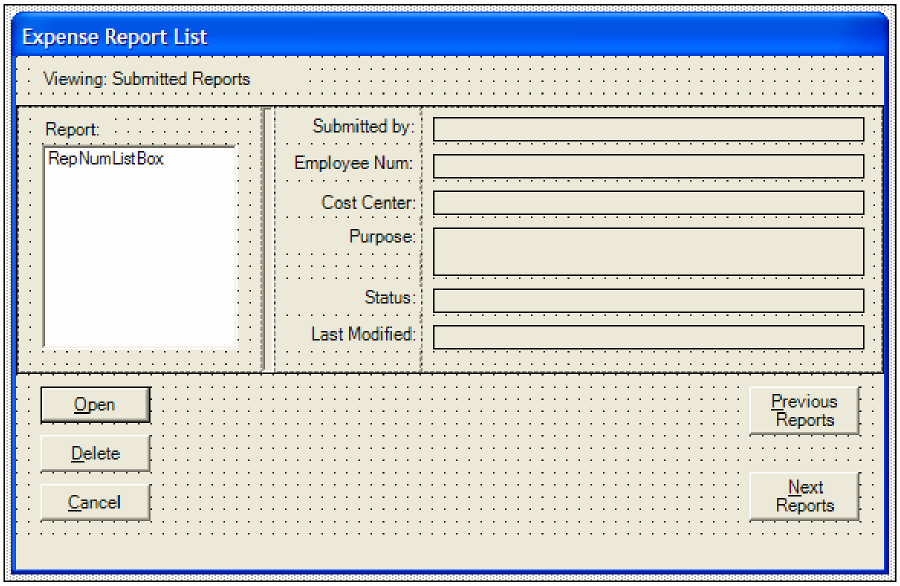
   The **Make Travel Plans** and **Procurement** buttons are just stubs with no real functionality. Students will work on implementing various parts of the Expense Reporting functionality. The **Exit** button closes the application.

6. Click **Expense Reporting**. Also demonstrate the multithreading feature of the application.

   When the user clicks the **Expense Reporting** button, an Expense Report control panel form appears.

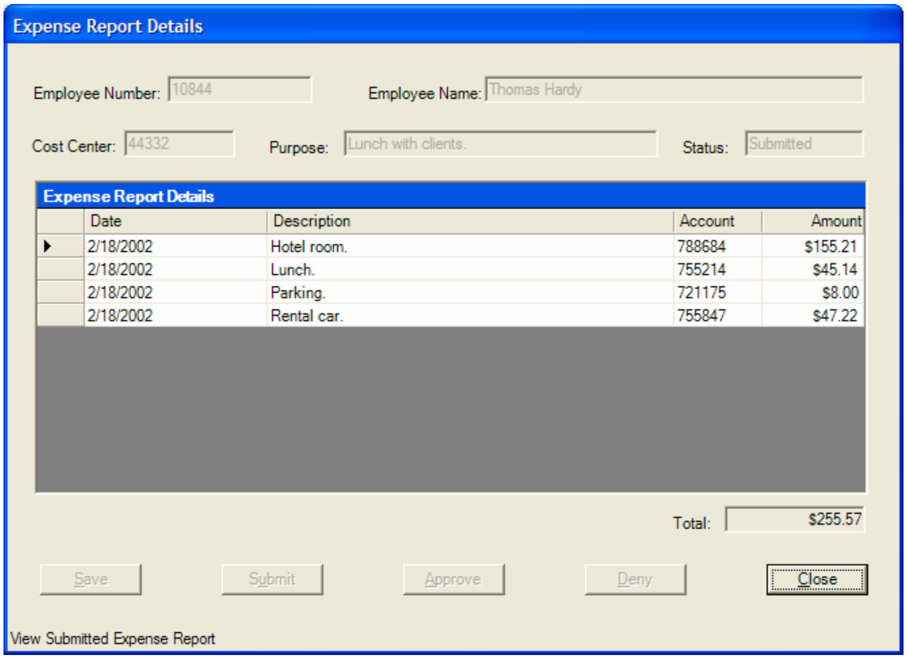


An Expense Report summary form displays a list of reports and summary information.



7. Double-click a given expense report in the list.

When the user double-clicks a given expense report in the list, an **Expense Report Details** screen appears and shows the details for that report.



Employee Number: 10844 Employee Name: Thomas Hardy

Cost Center: 44332 Purpose: Lunch with clients. Status: Submitted

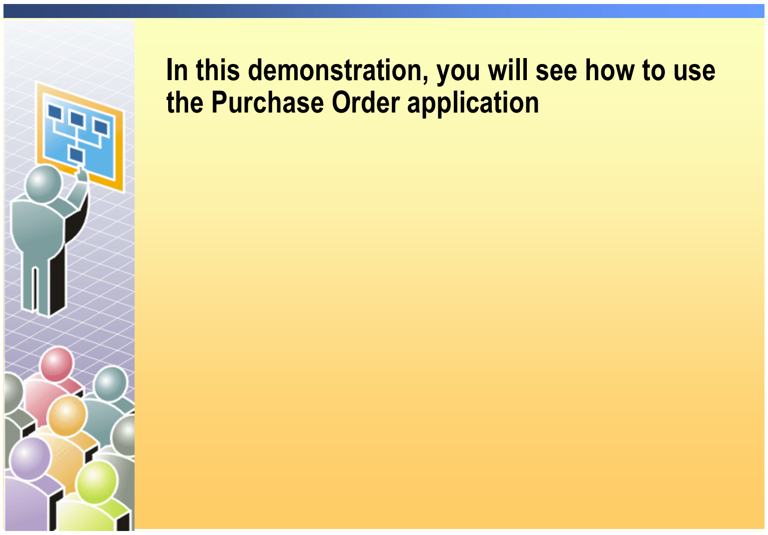
Expense Report Details

| | Date | Description | Account | Amount |
|---|---|---|---|---|
| ▶ | 2/18/2002 | Hotel room. | 788684 | $155.21 |
| | 2/18/2002 | Lunch. | 755214 | $45.14 |
| | 2/18/2002 | Parking. | 721175 | $8.00 |
| | 2/18/2002 | Rental car. | 755847 | $47.22 |

Total: $255.57

View Submitted Expense Report

# Demonstration: Purchase Order Application



In this demonstration, you will see how to use the Purchase Order application

**Introduction**

In this demonstration, you will see how to use the Purchase Order application.

**Instructions**

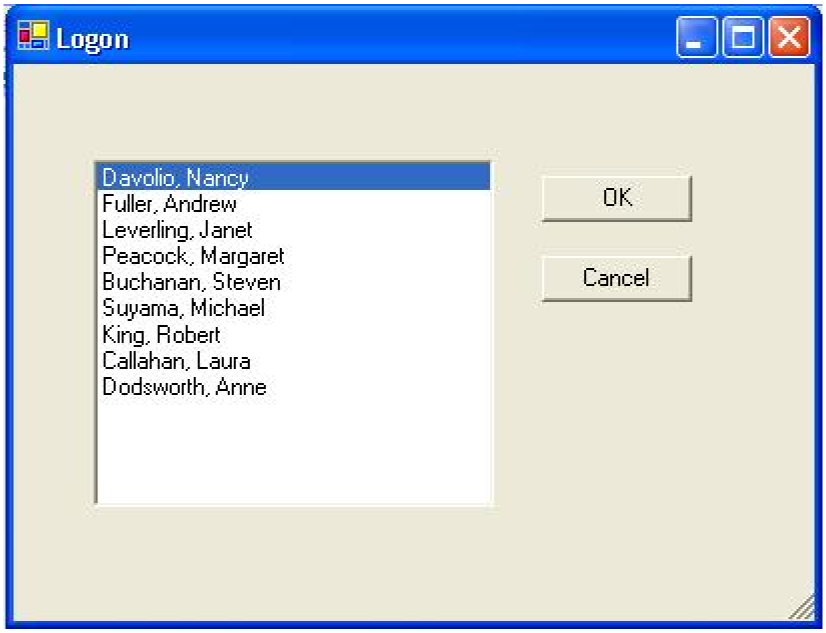
► **To run the demonstration**

1. Open the Purchase Order application from *install_folder*\Sampapps\OrderApplication.
2. Mention that the Purchase Order application consists of:
   - A Windows Forms-based client application.
   - A DataSet containing customer, product, and order information from the Northwind database. The DataSet is stored as a local XML file, and the user can choose to refresh this file when connected to the database. The DataSet is bound to controls in the main form and to a custom composite control.
   - A DataSet containing new order and order detail information that is stored as a local XML file and is used to update the database when connected.
   - A custom composite control that displays product information and exposes properties, methods, and events implemented in the main form.
   - A Crystal Report, bound to the local DataSet, that displays the history of customer orders.
   - Printing abilities that allow users to preview a print document, specify page settings, and print a document.

3. Describe the user interface.

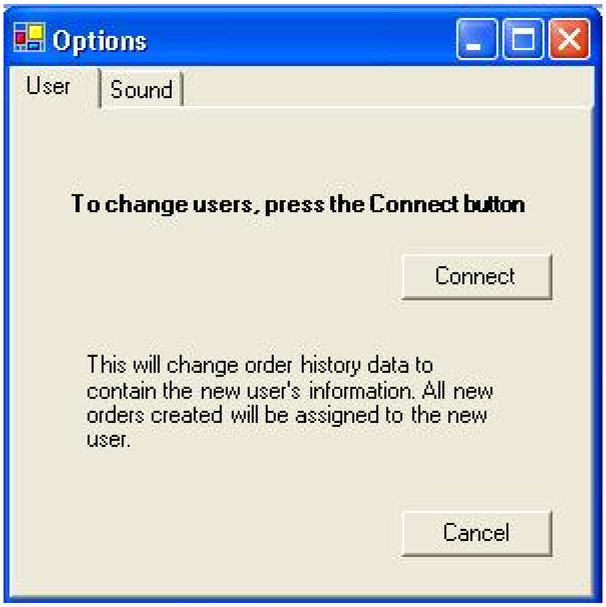
   The user interface of the client application consists of several screens.

   The Logon form is displayed when an employee chooses to refresh data and the **EmployeeID** is unknown. The form is also displayed when the user chooses to change their identity by using the **Option** menu on **MainForm**.



Davolio, Nancy
Fuller, Andrew
Leverling, Janet
Peacock, Margaret
Buchanan, Steven
Suyama, Michael
King, Robert
Callahan, Laura
Dodsworth, Anne

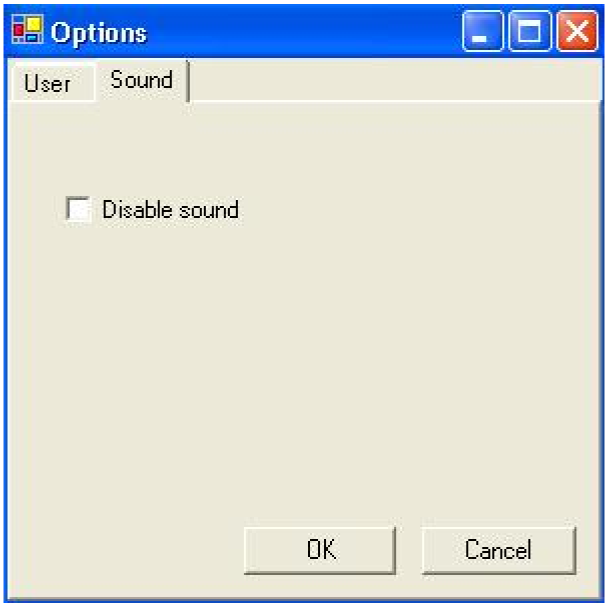
4. Open the Options form.



To change users, press the Connect button

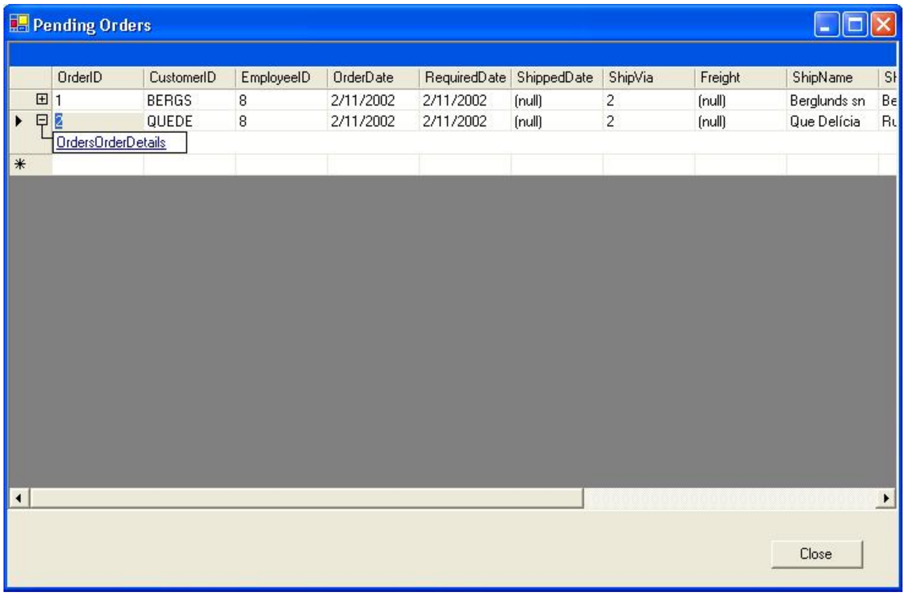
This will change order history data to contain the new user's information. All new orders created will be assigned to the new user.

Users can change the employee information of the Purchase Order application and can also turn the sound effects on or off. The Options form includes two tabs: the first is used to change the **EmployeeID**, and the second is used to turn the sound on or off.



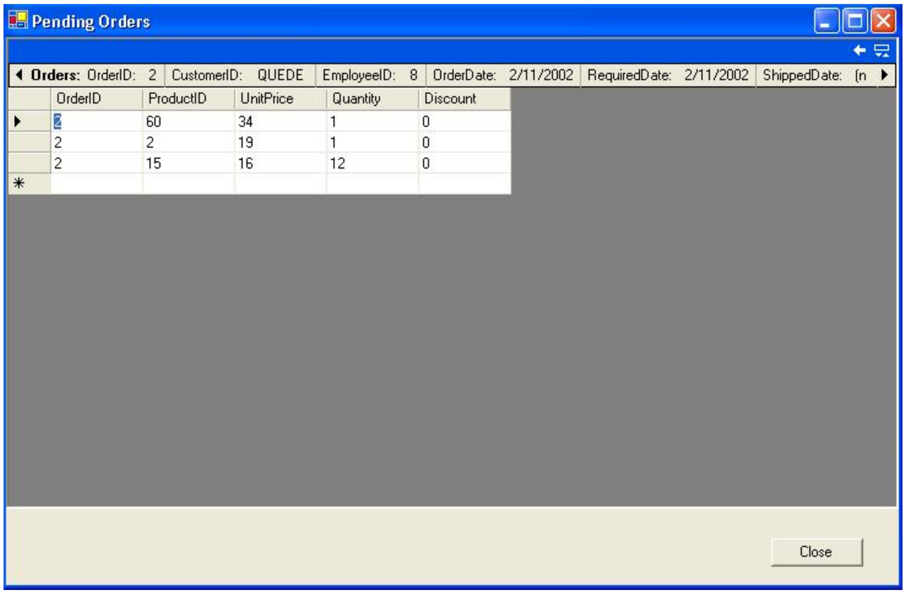
☐ Disable sound

5. On the **View** menu, click **View Unsubmitted Orders** to open the Pending Orders form.



| OrderID | CustomerID | EmployeeID | OrderDate | RequiredDate | ShippedDate | ShipVia | Freight | ShipName | Sh |
|---|---|---|---|---|---|---|---|---|---|
| 1 | BERGS | 8 | 2/11/2002 | 2/11/2002 | (null) | 2 | (null) | Berglunds sn | Be |
| 2 | QUEDE | 8 | 2/11/2002 | 2/11/2002 | (null) | 2 | (null) | Que Delícia | Ru |

The Pending Orders form allows users to view and edit orders that have not been submitted. Clicking the **OrdersOrderDetails** link for an order displays the individual order items of a particular order.



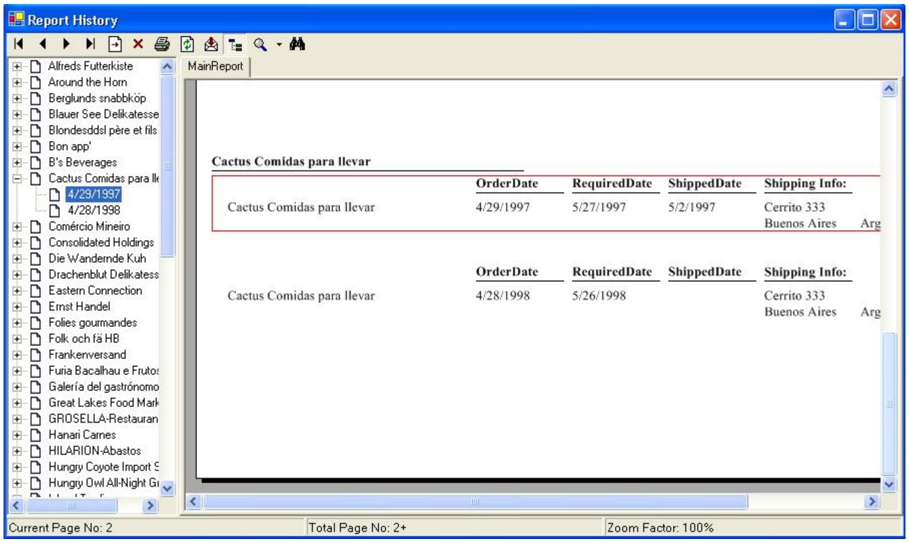
Orders: OrderID: 2 CustomerID: QUEDE EmployeeID: 8 OrderDate: 2/11/2002 RequiredDate: 2/11/2002 ShippedDate: (n

| OrderID | ProductID | UnitPrice | Quantity | Discount |
|---|---|---|---|---|
| 2 | 60 | 34 | 1 | 0 |
| 2 | 2 | 19 | 1 | 0 |
| 2 | 15 | 16 | 12 | 0 |

You can navigate from the parent and child tables by using the navigational controls provided by the **datagrid** control. Modifications can be made and are persisted when the Pending Orders form is closed.

In both views, some of the data columns are set to ReadOnly to maintain data integrity.

6. On the **View** menu, click **Submitted Orders** to open the Report History form.

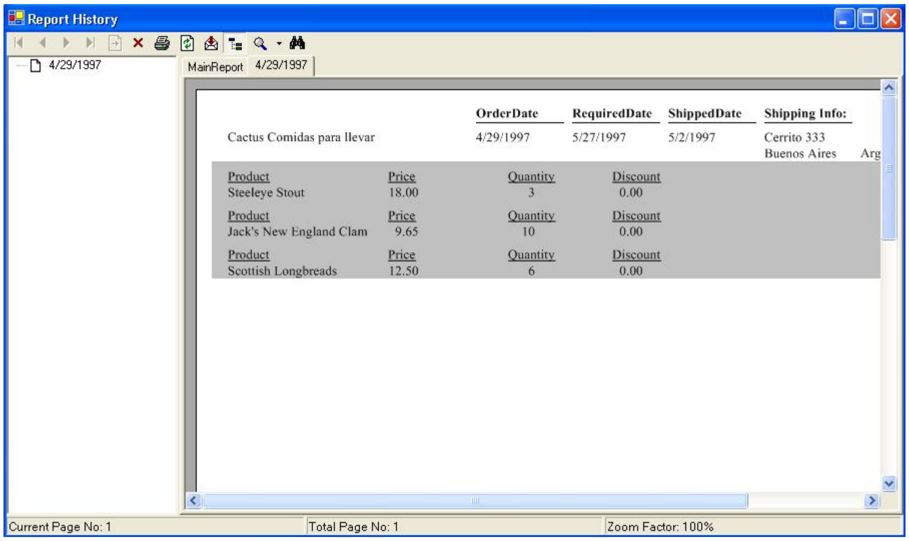


**Cactus Comidas para llevar**

| | OrderDate | RequiredDate | ShippedDate | Shipping Info: |
|---|---|---|---|---|
| Cactus Comidas para llevar | 4/29/1997 | 5/27/1997 | 5/2/1997 | Cerrito 333<br>Buenos Aires Arg |

| | OrderDate | RequiredDate | ShippedDate | Shipping Info: |
|---|---|---|---|---|
| Cactus Comidas para llevar | 4/28/1998 | 5/26/1998 | | Cerrito 333<br>Buenos Aires Arg |

This form uses a Crystal Report Viewer to display all order history for a given employee. Order information is displayed by **CustomerName** and by **OrderDate**. You can also click each order in the report to display order details.

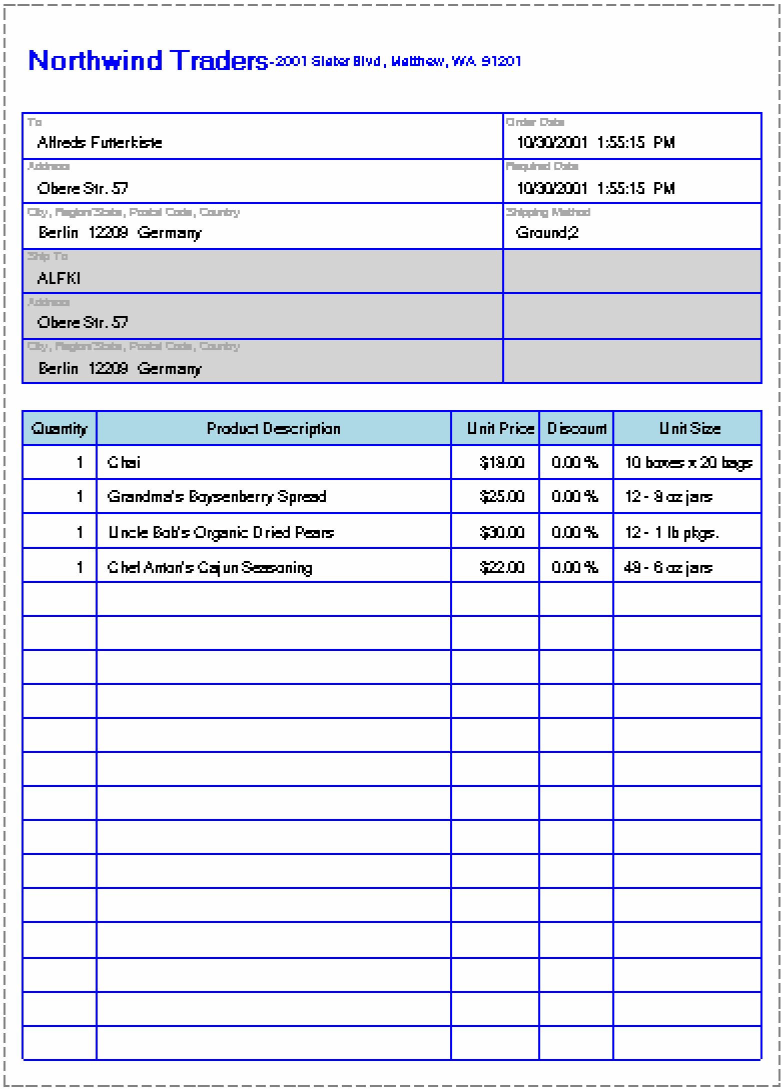


| | OrderDate | RequiredDate | ShippedDate | Shipping Info: |
|---|---|---|---|---|
| Cactus Comidas para llevar | 4/29/1997 | 5/27/1997 | 5/2/1997 | Cerrito 333<br>Buenos Aires Arg |

| Product | Price | Quantity | Discount |
|---|---|---|---|
| Steeleye Stout | 18.00 | 3 | 0.00 |
| **Product** | **Price** | **Quantity** | **Discount** |
| Jack's New England Clam | 9.65 | 10 | 0.00 |
| **Product** | **Price** | **Quantity** | **Discount** |
| Scottish Longbreads | 12.50 | 6 | 0.00 |

Current Page No: 1 | Total Page No: 1 | Zoom Factor: 100%

7. Show the print features by clicking the **PrintPreview** button on the toolbar.



Northwind Traders-2001 Slater Blvd, Matthew, WA 91201

| | |
|---|---|
| To<br>Alfreds Futterkiste | Order Date<br>10/30/2001 1:55:15 PM |
| Address<br>Obere Str. 57 | Required Date<br>10/30/2001 1:55:15 PM |
| City, Region/State, Postal Code, Country<br>Berlin 12209 Germany | Shipping Method<br>Ground2 |
| Ship To<br>ALFKI | |
| Address<br>Obere Str. 57 | |
| City, Region/State, Postal Code, Country<br>Berlin 12209 Germany | |

| Quantity | Product Description | Unit Price | Discount | Unit Size |
|---|---|---|---|---|
| 1 | Chai | $19.00 | 0.00 % | 10 boxes x 20 bags |
| 1 | Grandma's Boysenberry Spread | $25.00 | 0.00 % | 12 - 8 oz jars |
| 1 | Uncle Bob's Organic Dried Pears | $30.00 | 0.00 % | 12 - 1 lb pkgs. |
| 1 | Chef Anton's Cajun Seasoning | $22.00 | 0.00 % | 48 - 6 oz jars |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |
| | | | | |

# Microsoft Certified Professional Program

http://www.microsoft.com/traincert/

The Microsoft Certified Professional program is a leading certification program that validates your experience and skills to keep you competitive in today's changing business environment. The following table describes each certification in more detail.

| Certification | Description |
|---|---|
| MCSA on Microsoft Windows 2000 | The Microsoft Certified Systems Administrator (MCSA) certification is designed for professionals who implement, manage, and troubleshoot existing network and system environments based on Microsoft Windows 2000 platforms, including the Windows .NET Server family. Implementation responsibilities include installing and configuring parts of the systems. Management responsibilities include administering and supporting the systems. |
| MCSE on Microsoft Windows 2000 | The Microsoft Certified Systems Engineer (MCSE) credential is the premier certification for professionals who analyze the business requirements and design and implement the infrastructure for business solutions based on the Microsoft Windows 2000 platform and Microsoft server software, including the Windows .NET Server family. Implementation responsibilities include installing, configuring, and troubleshooting network systems. |
| MCSD | The Microsoft Certified Solution Developer (MCSD) credential is the premier certification for professionals who design and develop leading-edge business solutions with Microsoft development tools, technologies, platforms, and the Microsoft Windows DNA architecture. The types of applications MCSDs can develop include desktop applications and multi-user, Web-based, N-tier, and transaction-based applications. The credential covers job tasks ranging from analyzing business requirements to maintaining solutions. |

**(*continued*)**

| Certification | Description |
|---|---|
| MCDBA on Microsoft SQL Server™ 2000 | The Microsoft Certified Database Administrator (MCDBA) credential is the premier certification for professionals who implement and administer Microsoft SQL Server databases. The certification is appropriate for individuals who derive physical database designs, develop logical data models, create physical databases, create data services by using Transact-SQL, manage and maintain databases, configure and manage security, monitor and optimize databases, and install and configure SQL Server. |
| MCP | The Microsoft Certified Professional (MCP) credential is for individuals who have the skills to successfully implement a Microsoft product or technology as part of a business solution in an organization. Hands-on experience with the product is necessary to successfully achieve certification. |
| MCT | Microsoft Certified Trainers (MCTs) demonstrate the instructional and technical skills that qualify them to deliver Microsoft Official Curriculum through Microsoft Certified Technical Education Centers (Microsoft CTECs). |

## Certification Requirements

The certification requirements differ for each certification category and are specific to the products and job functions addressed by the certification. To become a Microsoft Certified Professional, you must pass rigorous certification exams that provide a valid and reliable measure of technical proficiency and expertise.

**For More Information** See the Microsoft Training and Certification Web site at http://www.microsoft.com/traincert/.

You can also send e-mail to mcphelp@microsoft.com if you have specific certification questions.

## Acquiring the Skills Tested by an MCP Exam

Microsoft Official Curriculum (MOC) and MSDN® Training Curriculum can help you develop the skills that you need to do your job. They also complement the experience that you gain while working with Microsoft products and technologies. However, no one-to-one correlation exists between MOC and MSDN Training courses and MCP exams. Microsoft does not expect or intend for the courses to be the sole preparation method for passing MCP exams. Practical product knowledge and experience is also necessary to pass the MCP exams.

To help prepare for the MCP exams, use the preparation guides that are available for each exam. Each Exam Preparation Guide contains exam-specific information, such as a list of the topics on which you will be tested. These guides are available on the Microsoft Training and Certification Web site at http://www.microsoft.com/traincert/.

# Facilities

- Class hours
- Building hours
- Parking
- Restrooms
- Meals
- Phones
- Messages
- Smoking
- Recycling

msdn® training

# Module 1: Introducing Windows Forms

**Contents**

Microsoft®

# Instructor Notes

**Presentation:**
**90 minutes**

**Lab:**
**30 minutes**

This module provides students with an overview of using Windows Forms, which is part of the new Microsoft® .NET Framework. Students will create Windows Forms and set their properties and controls to them. They will create inherited forms and also learn how to organize controls on a form. In the module, students also learn how to create Multiple Document Interface (MDI) applications.

After completing this module, students will be able to:

- Create a form and add controls to it.
- Create an inherited form by using Visual Inheritance.
- Organize controls on a form.
- Create MDI applications.

**Required materials**

To teach this module, you need the following materials: Microsoft PowerPoint® file 2555A_01.ppt.

**Preparation tasks**

To prepare for this module:

- Read all of the materials for this module.
- Complete the demonstration, practices, and lab.

# How to Teach This Module

This section contains information that will help you to teach this module. The following are some tips on how to teach this module:

- If students are interested in referencing code in other languages, point them to "Language Equivalents" in the Help documentation for the Microsoft Visual Studio® .NET development system. This section provides examples in languages such as Microsoft Visual Basic® .NET, Microsoft Visual C# ™, and Java.
- Lab 1.1: Creating Windows Forms is based on the Expense Report application in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)* and is intended to simulate a real-world environment in which students will demonstrate what they learned during the lecture and practice portions of the module. The lab does not provide step-by-step detailed instructions; instead, the students are given tasks to complete in the left column and a list of resources that they can use (if they need help) in the right column. Students get the hands-on experience that they need by completing the practice activities at the end of each lesson.

# Lesson: Creating a Form

This section describes the instructional methods for teaching this lesson.

**Windows Forms vs. Web Forms**

Although the course is about Windows Forms, students often have the question in their minds as to which is better—Windows Forms or Web Forms. This topic explains the differences between Windows Forms and Web Forms. Do not get too much into the details about Web Forms.

**How to Create a Form**

In this topic, demonstrate how to create a new form in Visual Studio .NET.

**How to Set Form Properties**

In this topic, demonstrate a few properties that can be set by using the Properties window.

**How to Handle Form Events**

This topic introduces students to how to create event handlers. Do not get into the details of event handlers. Event handlers are covered in greater detail in Module 2, "Using Windows Forms Controls" in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*

**Windows Forms Designer-Generated Code**

This topic shows the default code generated by the Windows Forms Designer. The main purpose of this topic is to show the code generated by the Designer and to advise students against modifying or deleting this code.

# Lesson: Adding Controls to a Form

This section describes the instructional methods for teaching this lesson.

**How to Add Controls to a Form**

This topic covers how to use the Toolbox to add controls to a form. Demonstrate how to drag and drop controls (buttons, labels, text boxes, and so on) from the Toolbox to a form.

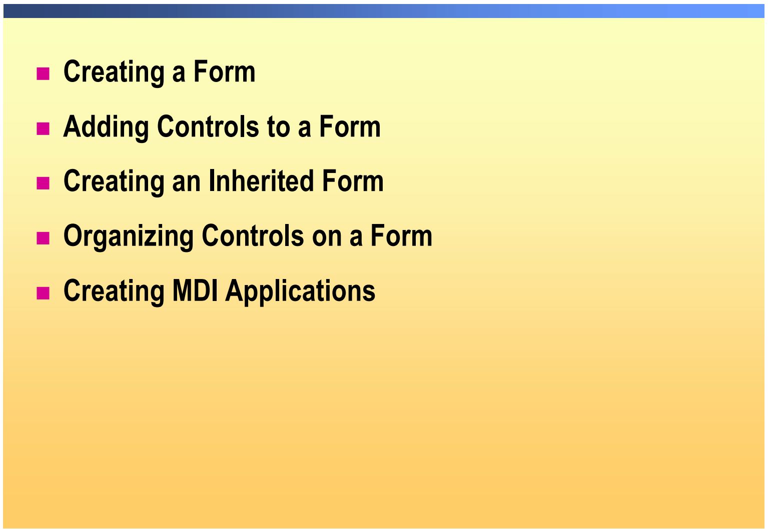
**How to Customize the Controls Toolbox**

Explain why students will need to customize the Toolbox. Demonstrate how to customize the Toolbox by adding a control that is not present in the Toolbox. Show students how they can remove controls from the Toolbox.

# Lab 1.1: Creating Windows Forms

- Make sure that you have demonstrated the two lab applications—the Expense Report application and the Purchase Order application—in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)* before students begin the lab. To see how to demonstrate lab scenarios, see the Introduction module in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- Practice exercises will enable students to successfully complete the lab exercises. Therefore, make sure that students have completed all practice exercises before they begin the lab.

# Overview



- Creating a Form
- Adding Controls to a Form
- Creating an Inherited Form
- Organizing Controls on a Form
- Creating MDI Applications

**Introduction**

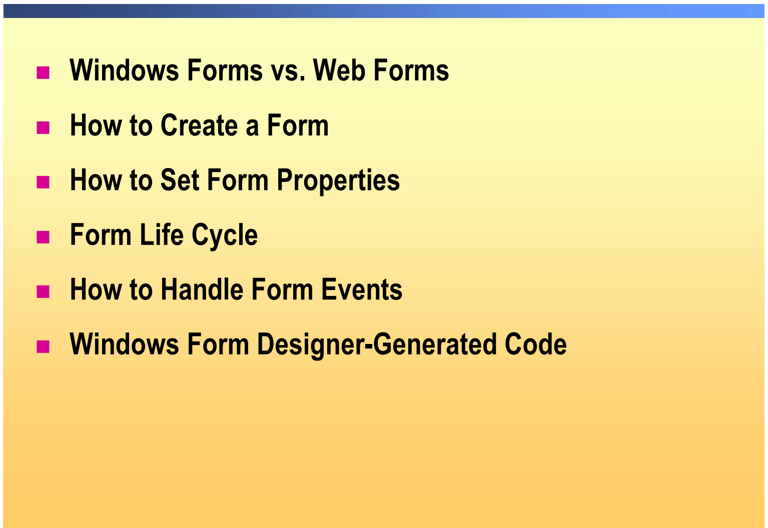
Windows Forms is part of the new Microsoft® .NET Framework, and it uses many new technologies including a common application framework, managed execution environment, integrated security, and object-oriented design principles. In addition, Windows Forms offers full support for quickly and easily connecting to Extensible Markup Language (XML) Web services and building rich, data-aware applications based on the ADO.NET data model. With the new shared development environment in the Microsoft Visual Studio® .NET development system, developers are able to create Windows Forms applications by using any of the languages supporting the .NET platform, including the Microsoft Visual C#™ .NET and Microsoft Visual Basic® .NET development systems.

**Objectives**

After completing this module, you will be able to:

- Create a form and add controls to it.
- Create an inherited form by using Visual Inheritance.
- Organize controls on a form.
- Create Multiple Document Interface (MDI) applications.

# Lesson: Creating a Form



- Windows Forms vs. Web Forms
- How to Create a Form
- How to Set Form Properties
- Form Life Cycle
- How to Handle Form Events
- Windows Form Designer-Generated Code

**Introduction**

Forms are the basic element of the user interface (UI) in applications created for the Microsoft Windows® operating system. They provide a framework that you can use throughout your application to give it a consistent look and feel. A form in Windows-based applications is used to present information to the user and to accept input from the user.

Forms expose properties that define their appearance, methods that define their behavior, and events that define their interaction with the user. By setting the properties of the form and writing code to respond to its events, you customize the form to meet the requirements of your application. A form is a control derived from the **Form** class, which in turn derives from the **Control** class. The framework also allows you to inherit from existing forms to add functionality or modify existing behavior. When you add a form to your project, you can choose whether it inherits from the **Form** class provided by the .NET Framework or from a form you created previously.

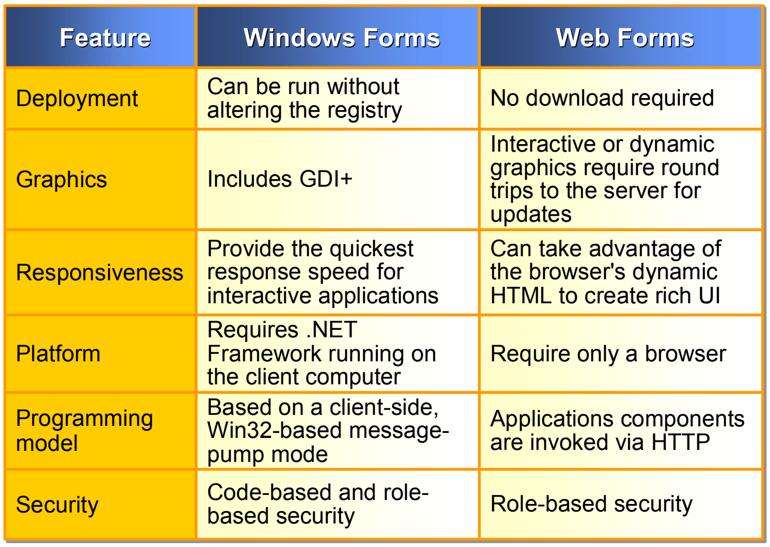
This lesson covers the basic concepts of forms and how to add controls to forms.

**Lesson objectives**

After completing this lesson, you will be able to:

- Describe a form.
- Determine whether to use Windows Forms or Web Forms in a scenario.
- Create a form.
- Set the properties of a form.
- Describe the events and methods in the forms life cycle.

# Windows Forms vs. Web Forms



| Feature | Windows Forms | Web Forms |
|---|---|---|
| Deployment | Can be run without altering the registry | No download required |
| Graphics | Includes GDI+ | Interactive or dynamic graphics require round trips to the server for updates |
| Responsiveness | Provide the quickest response speed for interactive applications | Can take advantage of the browser's dynamic HTML to create rich UI |
| Platform | Requires .NET Framework running on the client computer | Require only a browser |
| Programming model | Based on a client-side, Win32-based message-pump mode | Applications components are invoked via HTTP |
| Security | Code-based and role-based security | Role-based security |

### Introduction

When designing applications that involve a user interface, you have two choices: Windows Forms and Web Forms. Both have full design-time support in the development environment and can provide a rich user interface and advanced application functionality to solve business problems. When you have multiple options, it is important for you to know which option to use when.

### Windows Forms

Windows Forms are used to develop applications where the client is expected to shoulder a significant amount of the processing burden in an application. These include classic desktop applications for the Microsoft Win32® application programming interface. Examples include drawing or graphics applications, data-entry systems, point-of-sale systems, and games. All of these applications rely on the power of the desktop computer for processing and high-performance content display.

### Web Forms

ASP.NET Web Forms are used to create applications in which the primary user interface is a browser. This includes applications intended to be available publicly on the World Wide Web, such as e-commerce applications.

## Windows Forms vs. Web Forms

The following table provides a comparison of different application criteria and how Windows Forms and Web Forms technologies address these criteria.

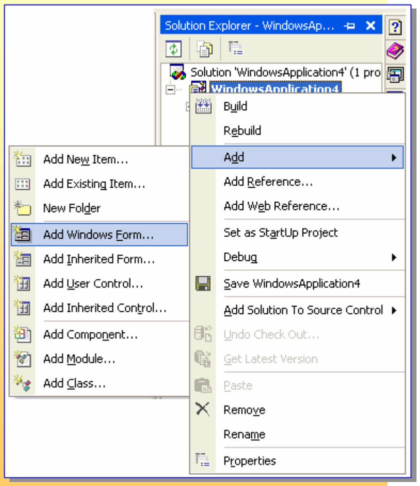
| Feature/criterion | Windows Forms | Web Forms |
|---|---|---|
| Deployment | Applications can be downloaded, installed, and run directly on the users' computers without any alteration of the registry. | Have no client deployment; the client requires only a browser. The server must be running Microsoft .NET Framework. Updates to the application are made by updating code on the server. |
| Graphics | Windows Forms include GDI+, which allows sophisticated graphics to be used for games and other extremely rich graphical environments. | Interactive or dynamic graphics require round trips to the server for updates when used on Web Forms. GDI+ can be used on the server to create custom graphics. |
| Responsiveness | Windows Forms can run entirely on the client computer; they can provide the quickest response speed for applications requiring a high degree of interactivity. | If you know that users will have Microsoft Internet Explorer 5 or later, a Web Forms application can take advantage of the browser's dynamic HTML (DHTML) capabilities to create a rich, responsive UI. If users have other browsers, most processing (including UI-related tasks such as validation) requires a round trip to the Web server, which can affect responsiveness. |
| Platform | Windows Forms require the .NET Framework to be running on the client computer. | Web Forms require only a browser. DHTML-capable browsers can take advantage of extra features, but Web Forms can be designed to work with all browsers. The Web server must be running .NET Framework. |

(***continued***)

| **Feature/criterion** | **Windows Forms** | **Web Forms** |
|---|---|---|
| Programming model | Windows Forms are based on a client-side, Win32-based message-pump mode, in which instances of components are created, used, and discarded by the developer. | Web Forms rely on a largely asynchronous, disconnected model, in which components are loosely coupled to the application front end. Typically, application components are invoked by HTTP. This model may not be suitable for applications that require extreme throughput from the user end or for those with high-volume transactions. Similarly, Web Forms applications may not be suitable for database applications that require high levels of concurrency control (for example, pessimistic locking). |
| Security | Windows Forms use granular permissions in its implementation of code access security to protect computer resources and sensitive information. This allows careful exposure of functionality, while retaining security. | Web Forms allow you to control the identity under which server application code is executed. Applications can execute code by using the identity of the requesting entity, which is known as impersonation. Applications can also dynamically tailor content based on the requestor's identity or role. For example, a manager could receive access to a site or to content that requires a higher level of security than someone with lesser credentials. |

# How to Create a Form

- A base form is created when you create a new project
- To create a new form
  1. Right-click the project in Solution Explorer
  2. Click Add
  3. Click Add Windows Forms



****************************ILLEGAL FOR NON-TRAINER USE******************************

## Introduction

In a Windows-based application, the form is the primary element for user interaction. By combining controls and your own actions, you can request information from the user and respond to it.

In Visual Studio .NET, a form is a window used in your application. When you create a new Windows Application project, Visual Studio .NET provides a Designer view that contains a form. The default form contains the minimum elements used by most forms: a title bar, a control box, and **Minimize**, **Maximize**, and **Close** buttons.

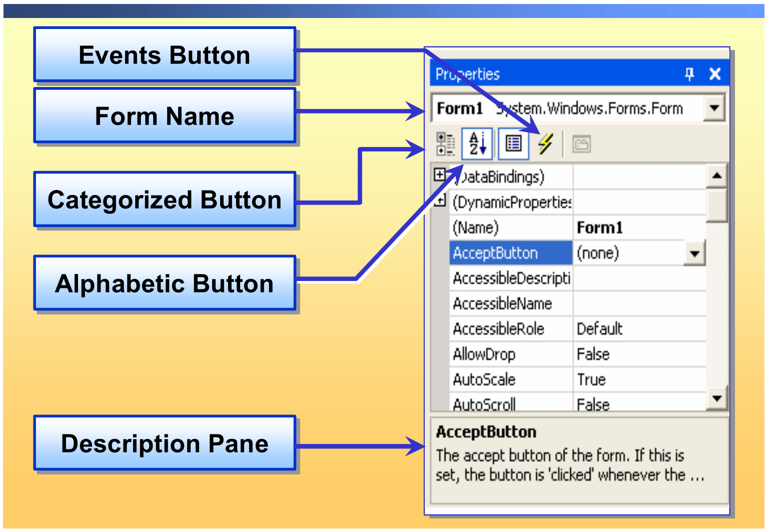
## Procedure: Creating forms

Most applications require more than one window. You must add a form to your project for every window that your application requires.

To add additional forms to your project:

1. If Solution Explorer is not open, on the **View** menu, click **Solution Explorer**.
2. In Solution Explorer, right-click the project name, point to **Add**, and then click **Add Windows Form**.
3. In the **Add New Item** dialog box, in the **Name** box, type an appropriate name for the form, and then click **Open**.

# How to Set Form Properties



Form1 System.Windows.Forms.Form

| | |
|---|---|
| (DataBindings) | |
| (DynamicProperties | |
| (Name) | Form1 |
| AcceptButton | (none) |
| AccessibleDescripti | |
| AccessibleName | |
| AccessibleRole | Default |
| AllowDrop | False |
| AutoScale | True |
| AutoScroll | False |

**AcceptButton**
The accept button of the form. If this is set, the button is 'clicked' whenever the ...

*****************************ILLEGAL FOR NON-TRAINER USE******************************

## Introduction

When you are building the user interface of a Windows-based application, you must set the properties for the objects that you create.

## Common form properties

The following table describes some common form properties that you typically set at design time.

| Property | Description | Default setting |
|---|---|---|
| **(Name)** | Sets the name of the form in your project. (This is not the name that is displayed to the user in the caption bar but rather the name that you will use in your code to reference the form.)<br>**Important:** If you change the **(Name)** property of your form, you must set the startup object for your project to the new name or the project will not start correctly. For information about how to change the startup object, see Form Life Cycle in this lesson in this module. | Form1 (Form2, Form3, and so on) |
| **AcceptButton** | Sets which button is clicked when the user presses the ENTER key.<br>**Note:** You must have at least one button on your form to use this property. | None |
| **CancelButton** | Sets which button is clicked when the user presses the ESC key.<br>**Note:** You must have at least one button on your form to use this property. | None |

**(*continued*)**

| Property | Description | Default setting |
|---|---|---|
| **ControlBox** | Determines whether a form displays a control box in the caption bar. The control box can contain the **Minimize** button, **Maximize** button, **Help** button, and the **Close** button. | True |
| **FormBorderStyle** | Controls the appearance of the border for the form. This will also affect how the caption bar appears and what buttons appear on it. | Sizable |
| **MaximizeBox** | Determines whether a form has a **Maximize** button in the upper right corner of its caption bar. | True |
| **MinimizeBox** | Determines whether a form has a **Minimize** button in the upper right corner of its caption bar. | True |
| **StartPosition** | Determines the position of a form on the screen when it first appears. | WindowsDefaultLocation |
| **Text** | Sets the text displayed in the caption bar of the control. | Form1 (Form2, Form3, and so on) |

### Procedure: Setting form properties

You can set form properties either by writing code or by using the Properties window. Any property settings that you establish at design time are used as the initial settings each time your application runs.

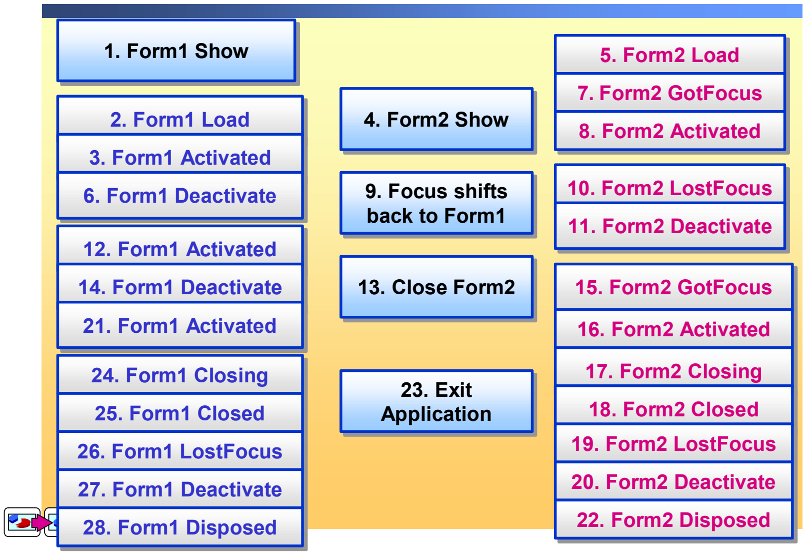
To set form properties at design time:

1. If the Properties window is not open, on the **View** menu, click **Properties Window**.
2. In Design view, click the form for which you want to set a property. The name of the form appears in the **Object** list at the top of the Properties window.
3. Use the **Alphabetic** and **Categorized** buttons in the Properties window to choose whether to view the form properties alphabetically or by category.
4. In the **Properties** window, click the property that you want to set.

**Note** When you select a property, a description of the property appears at the bottom of the Properties window, in the Description pane.

5. Type or select the property setting that you want.

# Form Life Cycle



## Introduction

After adding the necessary forms to your project and setting the startup form, you must determine which events and methods to use. The entire life cycle of a form uses several methods and events.

## Form events and methods

When the **Show()** method is called, the form events and methods are generally triggered in the following order:

1. Load
2. GotFocus
3. Activated
4. Closing
5. Closed
6. Deactivate
7. LostFocus
8. Dispose()

### New

The **Initialize** event is typically used to prepare an application for use. Variables are assigned to initial values, and controls may be moved or resized to accommodate initialization data.

In .NET, initialization code must be added to the form constructor after the call to **InitializeComponent()** as shown in the following example:

```
public CalcUI()
{
  //
  // Required for Windows Form Designer support
  //

  InitializeComponent();

  //
  // Add your initialization code here
  //
}
```

### Show

The **Show** method includes an implied **Load**; this means that if the specified form is not already loaded when the **Show** method is called, the application automatically loads the form into memory and then displays it to the user. The **Show** method can display forms as modal or modeless.

```
FrmSplash.Show();
```

You can use the **ShowDialog()** method to show a form as a dialog box.

### Load

The **Load** event is used to perform actions that must occur before the form displays. It is also used to assign default values to the form and its controls.

The **Load** event occurs each time that a form is loaded into memory. A form's **Load** event can run multiple times during an application's life. **Load** fires when a form starts as the result of the **Load** statement, **Show** statement, or when a reference is made to an unloaded form's properties, methods, or controls.

### Activated/Deactivate

When the user moves among two or more forms, you can use the **Activated** and **Deactivate** events to define the forms' behaviors. The **Activated** event occurs when the form is activated in code or by the user. To activate a form at run time by using code, call the **Activate** method. You can use this event for tasks such as updating the contents of the form based on changes made to the form's data when the form was not activated.

The **Activated** event fires when the form receives focus from another form in the same project. This event fires only when the form is visible. For example, a form loaded by using the **Load** statement isn't visible unless you use the **Show** method, or set the form's **Visible** property to **True**. The **Activated** event fires before the **GotFocus** event.

Use the following code to set the focus to a form.

```
FrmSplash.Focus();
```

**Deactivate** fires when the form loses focus to another form. This event fires after the **LostFocus** event.

Both the **Activated** and **Deactivate** events fire only when focus is changing within the same application. If you switch to a different application and then return to the program, neither event fires.

**Important** If you need to add code that executes either when the form is being displayed or when the form is being hidden, add the code to the **Activated** and **Deactivate** event handlers instead of to the **GotFocus** and **LostFocus** event handlers.

### Closing

The **Closing** event is useful when you need to know how the user is closing the form. The **Closing** event occurs when the form receives a request to close. Data validation can occur at this time. If there is a need to keep the form open (for example, if data validation fails), the closing event can be canceled.

### Closed

The **Closed** event occurs when the form is closed and before the **Dispose** event. Use the **Closed** event procedure to verify that the form should be closed or to specify actions that take place when closing the form. You can also include form-level validation code for closing the form or saving data to a file.

### Dispose

The .NET framework does not support the **Terminate** event. Termination code must execute inside the **Dispose** method, before the call to **base.Dispose()**.

```
protected override void Dispose( bool disposing )
{
  // Termination code goes here.
  if( disposing )
  {
      if (components != null)
      {
          components.Dispose();
      }
  }
  base.Dispose( disposing );
}
```

The **Dispose** method is called automatically for the main form in an application; you must call it explicitly for any other form.

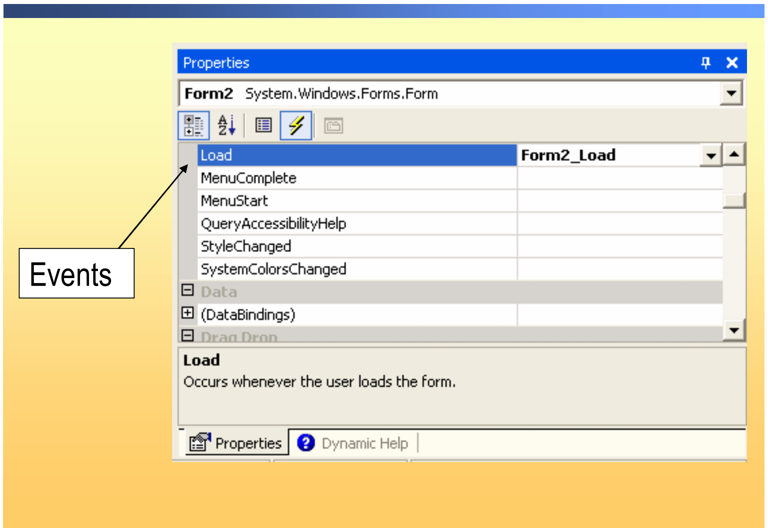
### Hide

The **Hide** method removes a form from the screen without removing it from memory. A hidden form's controls are not accessible to the user, but they are available to the running application. When a form is hidden, the user cannot interact with the application until all code in the event procedure that caused the form to be hidden has finished executing.

If the form is not already loaded into memory when the **Hide** method is called, the **Hide** method loads the form but doesn't display it.

```
frmMyForm.Hide();
```

# How to Handle Form Events



**Load**
Occurs whenever the user loads the form.

*******************************ILLEGAL FOR NON-TRAINER USE********************************

### Introduction

An *event handler* is a segment of code that is called when a corresponding event occurs. For example, you can write code in an event handler for the **Activated** event of a form to perform operations such as updating the data displayed in the controls of the form when the form is activated.

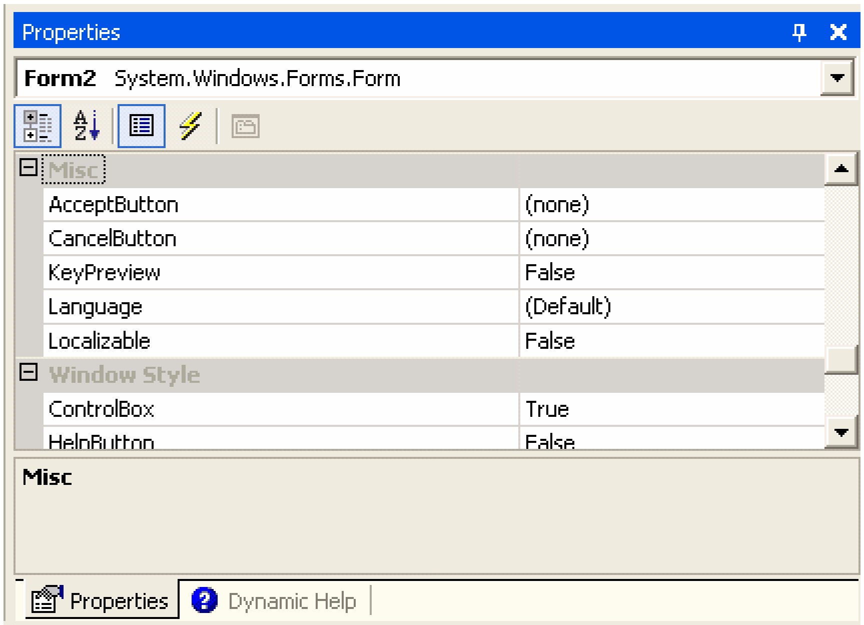
The .NET Framework uses a standard naming convention for event handlers. The convention is to combine the name of the object that sends the event, an underscore, and the name of the event. For example, the **Click** event of a form named Form1 would be named **Form1_Click**.

**Procedure**

To add an event handler:

1. Open the Properties windows for the form for which you want to add an event handler.

2. Click the Event icon ⚡ in the Properties window to view the events.

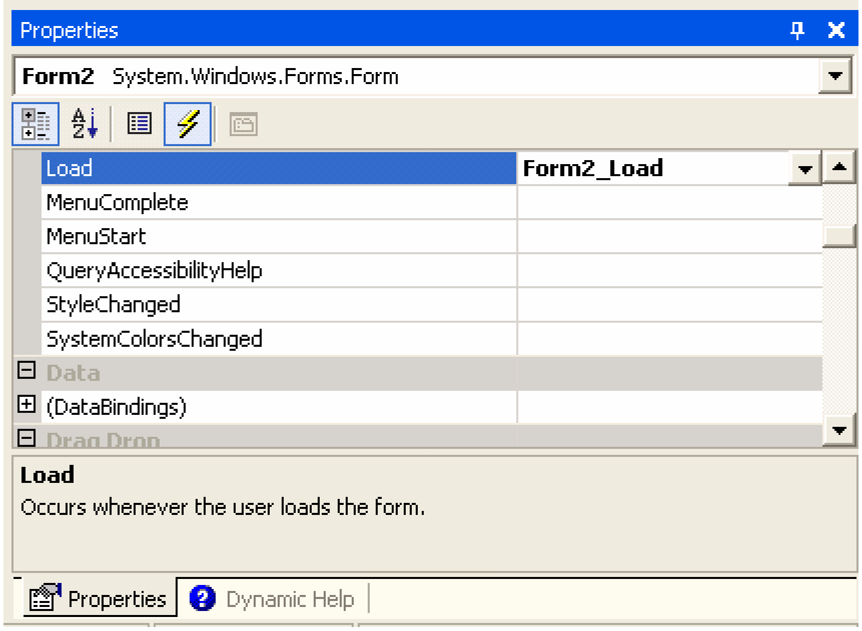


Form2 System.Windows.Forms.Form

| Misc | |
|---|---|
| AcceptButton | (none) |
| CancelButton | (none) |
| KeyPreview | False |
| Language | (Default) |
| Localizable | False |
| **Window Style** | |
| ControlBox | True |
| HelpButton | False |

**Misc**

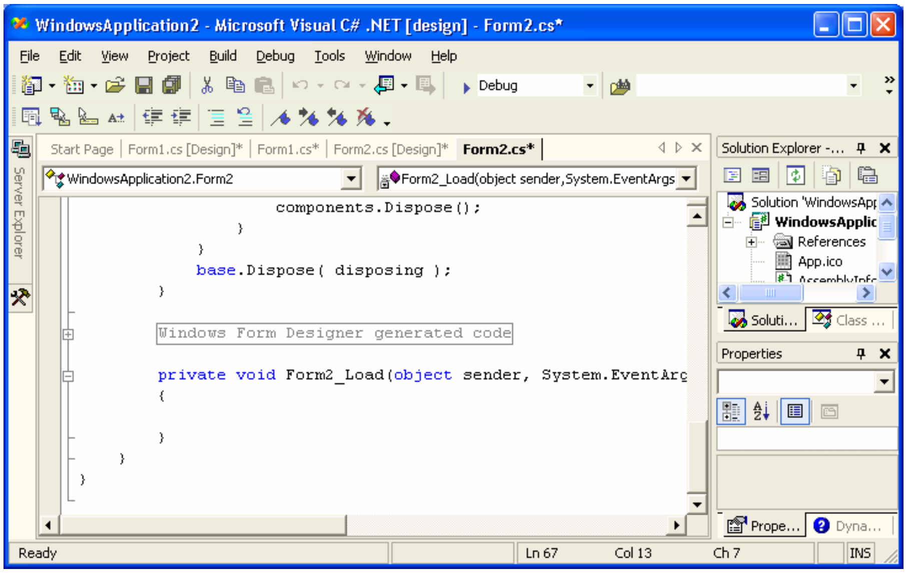
3. Double-click the event to add an event handler.



Form2 System.Windows.Forms.Form

| Load | **Form2_Load** |
|---|---|
| MenuComplete | |
| MenuStart | |
| QueryAccessibilityHelp | |
| StyleChanged | |
| SystemColorsChanged | |
| **Data** | |
| (DataBindings) | |
| **Drag Drop** | |

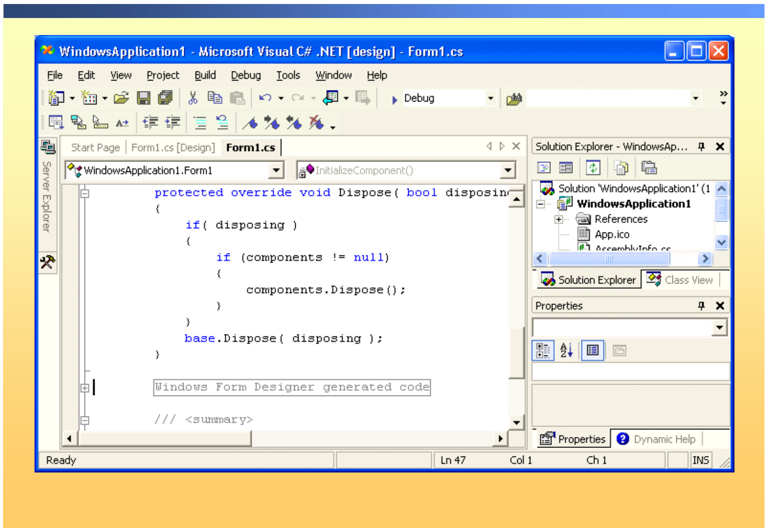
**Load**
Occurs whenever the user loads the form.


WindowsApplication2 - Microsoft Visual C# .NET [design] - Form2.cs*
File Edit View Project Build Debug Tools Window Help
Debug
Server Explorer
Start Page | Form1.cs [Design]* | Form1.cs* | Form2.cs [Design]* | Form2.cs*
WindowsApplication2.Form2
Form2_Load(object sender,System.EventArgs
Solution Explorer -...
Solution 'WindowsApp
WindowsApplic
References
App.ico
AssemblyInfo
Soluti... Class ...
Properties
Prope... Dyna...
Ready
Ln 67
Col 13
Ch 7
INS

```
                    components.Dispose();
                }
            }
            base.Dispose( disposing );
        }

        Windows Form Designer generated code

        private void Form2_Load(object sender, System.EventArg
        {

        }
    }
}
```

You will learn more about using events and event handlers in the .NET Framework in Module 2, “Working with Controls” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.

# Windows Forms Designer-Generated Code


WindowsApplication1 - Microsoft Visual C# .NET [design] - Form1.cs
File Edit View Project Build Debug Tools Window Help
Debug
Server Explorer
Start Page | Form1.cs [Design] | Form1.cs
WindowsApplication1.Form1
InitializeComponent()
Solution Explorer - WindowsAp...
Solution 'WindowsApplication1' (1
WindowsApplication1
References
App.ico
AssemblyInfo.cs
Solution Explorer
Class View
Properties
Properties
Dynamic Help
Ready
Ln 47
Col 1
Ch 1
INS

```
protected override void Dispose( bool disposin
{
    if( disposing )
    {
        if (components != null)
        {
            components.Dispose();
        }
    }
    base.Dispose( disposing );
}

Windows Form Designer generated code

/// <summary>
```

### Introduction

When you create a form by using Windows Forms Designer, the Designer generates a lot of code that you would have to write if you were creating a form on your own.

### Designer-generated code

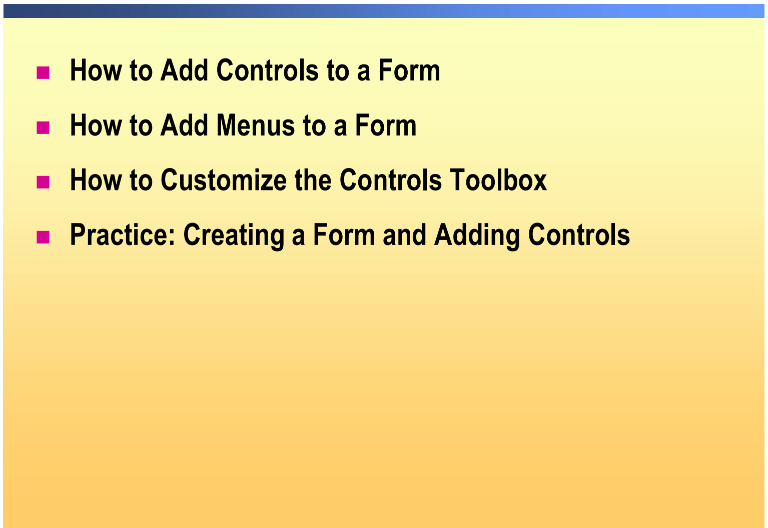
If you look at the default code for the form, you will find the following code generated by the Designer.

```
#region Windows Form Designer generated code
      /// <summary>
      /// Required method for Designer support - do not modify
      /// the contents of this method with the code editor.
      /// </summary>
      private void InitializeComponent()
      {
          //
          // Form1
          //
          this.AutoScaleBaseSize = new System.Drawing.Size(5,
13);
          this.ClientSize = new System.Drawing.Size(292, 266);
          this.Name = "Form1";
          this.Text = "Form1";
          this.Load += new
System.EventHandler(this.Form1_Load);

      }
      #endregion
```

Notice the InitializeComponent section. This code is used by the development environment to persist the property values you set in the Windows Forms Designer.

**Important** Avoid modifying or deleting the Windows Forms Designer-generated code. Modifying or deleting this code can result in errors in your project.

# Lesson: Adding Controls to a Form



- How to Add Controls to a Form
- How to Add Menus to a Form
- How to Customize the Controls Toolbox
- Practice: Creating a Form and Adding Controls

**Introduction**

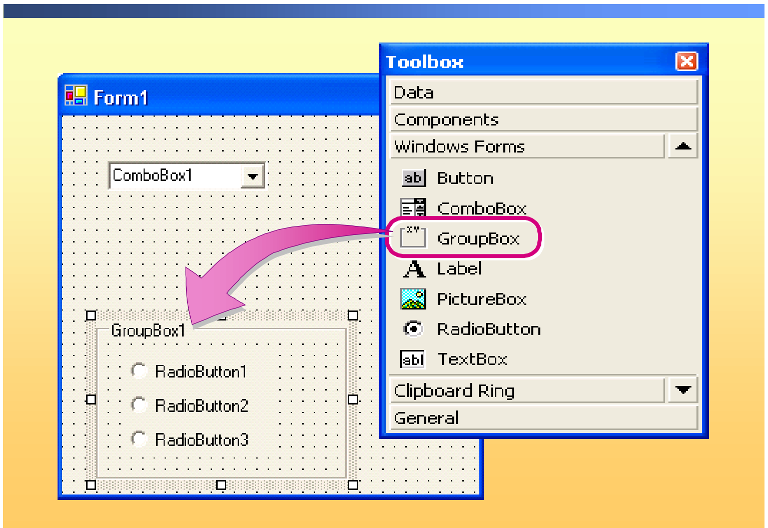
To create a user interface for an application, you must add controls to a form. This lesson covers adding controls to a form.

**Lesson objectives**

After completing this lesson, you will be able to:

- Add controls to a form.
- Add menus to a form.
- Customize the Controls Toolbox.

# How to Add Controls to a Form



### Introduction

Controls are objects that are contained in form objects. Buttons, text boxes, and labels are examples of controls.

### Procedure: Adding controls to a form

There are two ways to add controls to a form. The first way allows you to add several controls quickly and then size and position them individually. The second way gives you more initial control over the size and position of the control.

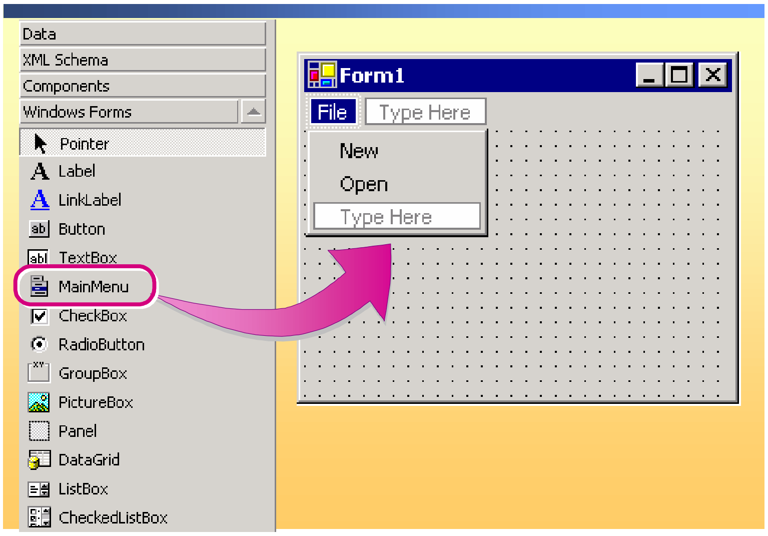
To add controls to a form and then size and position them:

1. If the Toolbox is not open, on the **View** menu, click **Toolbox**.
2. In the Toolbox, double-click the control that you want to add. This places an instance of the control at the default size in the upper left corner of the active object. When adding multiple controls in this manner, they are placed on top of each other.
3. After the controls are added, you can reposition and resize them:
   a. To reposition the control, click the control to select it, and then drag the control to the correct position.
   b. To resize the control, click the control to select it, drag one of the eight sizing handles until the control is properly sized.

To size and position controls while you add them to a form:

1. If the Toolbox is not open, on the **View** menu, click **Toolbox**.
2. In the Toolbox, click the control that you want to add.
3. Move the mouse pointer over the form. The pointer symbol changes to a crosshair.
4. Position the crosshair where you want the upper left corner of the control.
5. Click and drag the crosshair where you want the lower right corner. A rectangle that indicates the control's size and location is drawn on the screen.
6. When the control is correctly sized, release the mouse button. The sized control appears in the correct location on the form.
7. You can reposition or resize the control after you have released the mouse button:
   a. To reposition the control, click the control to select it, and then drag the control to the correct position.
   b. To resize the control, click the control to select it, and drag one of the eight sizing handles until the control is properly sized.

# How to Add Menus to a Form



### Introduction

Menus provide a structured way for users to access the commands and tools contained in an application. Proper planning and design of menus and toolbars is essential and ensures proper functionality and accessibility of your application to users.

A menu control has many properties such as **Name**, **Caption**, and **Index**.

- The **Name** property identifies the menu control in code.
- The **Index** property identifies controls that share the same name.
- The **Caption** property is the text that appears on the menu bar at run time.

### Procedure: Adding menus to a form

To add menus to a form:

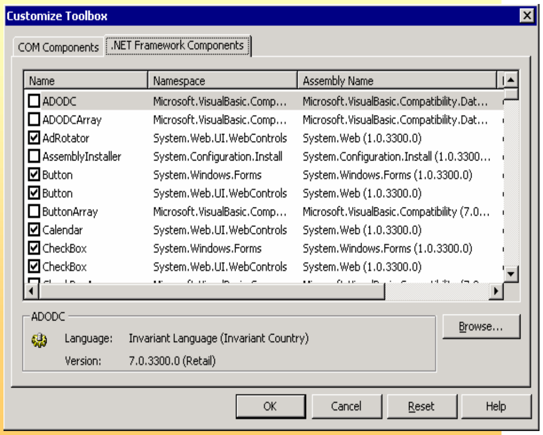
1. If the Toolbox is not open, on the **View** menu, click **Toolbox**.
2. In the Toolbox, double-click the **MainMenu** control.
3. In the **Caption** box, on the newly created menu, type the text for the first menu's title. This title will appear on the menu bar.
4. In the **Name** box, in the Properties window, type the name that you will use to refer to the menu control in code.

# How to Customize the Controls Toolbox

1 Right-click the Toolbox

2 Click Customize Toolbox

3 Select the required control on the .NET Framework Components page



Customize Toolbox

COM Components | .NET Framework Components

| Name | Namespace | Assembly Name |
|---|---|---|
| ☐ ADODC | Microsoft.VisualBasic.Comp… | Microsoft.VisualBasic.Compatibility.Dat… |
| ☐ ADODCArray | Microsoft.VisualBasic.Comp… | Microsoft.VisualBasic.Compatibility.Dat… |
| ☑ AdRotator | System.Web.UI.WebControls | System.Web (1.0.3300.0) |
| ☐ AssemblyInstaller | System.Configuration.Install | System.Configuration.Install (1.0.3300… |
| ☑ Button | System.Windows.Forms | System.Windows.Forms (1.0.3300.0) |
| ☑ Button | System.Web.UI.WebControls | System.Web (1.0.3300.0) |
| ☐ ButtonArray | Microsoft.VisualBasic.Comp… | Microsoft.VisualBasic.Compatibility (7.0… |
| ☑ Calendar | System.Web.UI.WebControls | System.Web (1.0.3300.0) |
| ☑ CheckBox | System.Windows.Forms | System.Windows.Forms (1.0.3300.0) |
| ☑ CheckBox | System.Web.UI.WebControls | System.Web (1.0.3300.0) |

ADODC

Language: Invariant Language (Invariant Country)

Version: 7.0.3300.0 (Retail)

**Introduction**

The Toolbox displays a variety of items for use in Visual Studio .NET projects. The items available include .NET components, Component Object Model (COM) components, Hypertext Markup Language (HTML) objects, code fragments, and text. The Toolbox contains a variety of controls that you can use to add art work, labels, buttons, list boxes, scroll bars, menus, and geometric shapes to a user interface.

Each control that you add to a form becomes a programmable user interface object in your application. These objects are visible to the user when the application runs, and they operate like the standard objects in any Windows-based application. However, there are some controls that are visible on the Toolbox by default. You must customize the Toolbox to display such controls on the Toolbox. For example, the **StatusBarPanel** control is not visible on the Toolbox by default.

You can customize the Toolbox by adding and removing items from it. The **Customize Toolbox** dialog box displays tabbed lists of components that are recognized on your machine. Use the **Customize Toolbox** dialog box to add controls to the Toolbox or remove controls from it.

**Note** The **Customize Toolbox** dialog box replaces the **Customize** dialog box in previous versions of Microsoft Visual C++®.

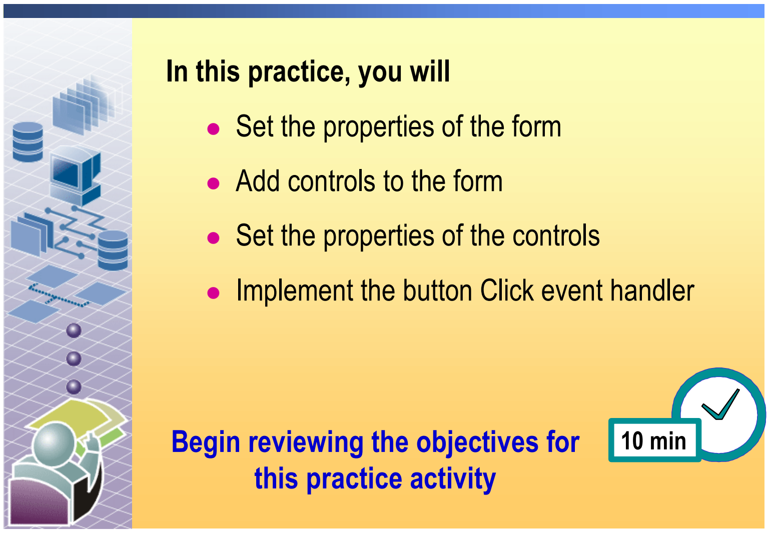
### Procedure: Customizing the Controls Toolbox

To customize the Controls Toolbox:

1. Right-click the Toolbox.
2. Click **Customize Toolbox**.
3. Click the .**NET Framework Components** tab or the **COM Components** tab, and select the required controls.

**Note** You can add code fragments to the Toolbox by selecting the code fragment or text and dragging it to the Toolbox. A new entry beginning with **text** will appear in the Controls Toolbox.

# Practice: Creating a Form and Adding Controls



**In this practice, you will**

- Set the properties of the form
- Add controls to the form
- Set the properties of the controls
- Implement the button Click event handler

Begin reviewing the objectives for this practice activity

10 min

**Introduction**

In this practice, you will open an existing project and modify the properties on the default form. You will also add controls to the form and implement the **Click** event for two buttons.

**Instructions**

### ▶ Open the practice project

1. Use Windows Explorer and browse to *install_folder*\Practices\Mod01\Mod01_01\Starter.

> **Note** If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

2. Double-click the CreatingForms.sln solution file to open the project.

### ▶ Set the properties of a form

1. Open the CreatingForms.cs file in Design view.
2. If the Properties window is not visible, on the **View** menu, click **Properties Window**.
3. Set the remaining form properties to the indicated values.

| Property | Value |
|---|---|
| **ControlBox** | **false** |
| **Font** | **Trebuchet MS, 10pt** |
| **FormBorderStyle** | **Fixed3D** |
| **Size** | **300, 175** |
| **Text** | **Hello World** |

### ► Add controls to the form

1. If the Toolbox is not visible, on the **View** menu, click **Toolbox**.
2. In the Toolbox, double-click the **Label** control to add it to the form.
3. Double-click the **Button** control to add it to the form.
4. Double-click the **Button** control again to add a second button to the form.
5. Position the **Label** control near the top center of the form.
6. Position the buttons next to each other near the bottom of the form—with **button1** on the left and **button2** on the right.

### ► Set the properties of the controls

1. Click the **Label1** control.
2. Set the following properties for the **Label1** control to the values provided.

| Property | Value |
|---|---|
| (Name) | OutputLabel |
| BorderStyle | Fixed3D |
| Font | Trebuchet MS, 10pt, Bold |
| ForeColor | ActiveCaption |
| Location | 14,30 |
| Size | 264, 23 |
| Text | (Delete existing text and leave it blank) |
| TextAlign | MiddleCenter |

3. Click **Button1**.
4. Set the following properties for the **Button1** control to the values provided in the following table.

| Property | Value |
|---|---|
| (Name) | HelloButton |
| Location | 57, 87 |
| Size | 75,25 |
| Text | &Say Hello |

5. Click **Button2**.
6. Set the following properties for the **Button2** control to the values provided in the following table.

| Property | Value |
|---|---|
| (Name) | ExitButton |
| Location | 161, 87 |
| Size | 75,25 |
| Text | E&xit |

7. Double-click the **Say Hello** button to create the **Click** event handler.
8. In the **Click** event handler for **HelloButton**, add the following line of code:

```
OutputLabel.Text = "Hello, World!";
```

9. Switch back to Design view of the form.
10. Double click the **Exit** button to create the **Click** event handler.
11. In the **Click** event handler for **ExitButton**, add the following line of code:

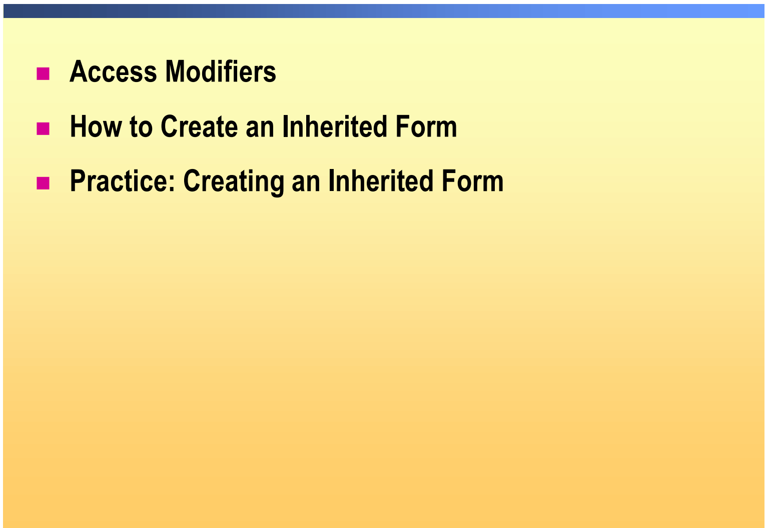
```
this.Close();
```

12. Switch back to Design view of the form.
13. Set the **AcceptButton** property to **HelloButton** and the **CancelButton** property to **ExitButton**.

### ► Build and run the application

1. To build the application, click the **Build** menu, and then click **Build Solution**.
2. Run the application by pressing F5.

# Lesson: Creating an Inherited Form



- Access Modifiers
- How to Create an Inherited Form
- Practice: Creating an Inherited Form

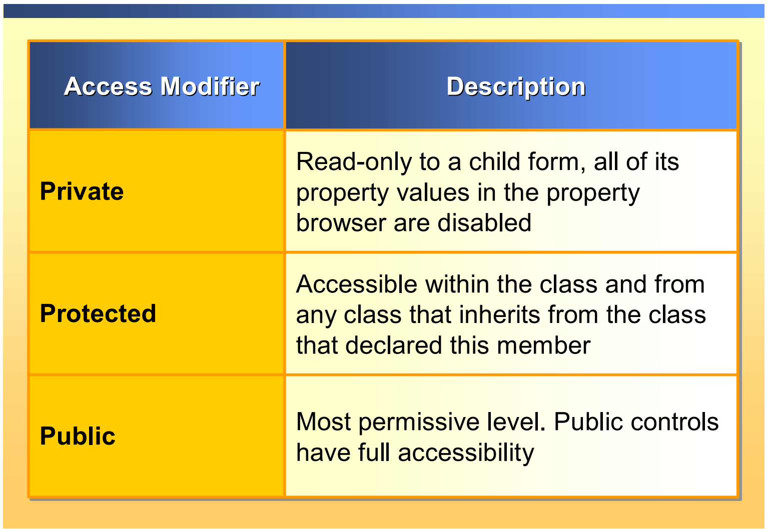
**Introduction**

Creating new Windows Forms by inheriting from base forms is an efficient way to duplicate your best efforts without going through the process of entirely recreating a form every time you require it. The process of inheriting a form from an existing one is called *Visual Inheritance*. This lesson covers how to inherit from an existing form by using Visual Inheritance.

**Lesson objective**

After completing this lesson, you will be able to create an inherited form.

# Access Modifiers



| Access Modifier | Description |
|---|---|
| **Private** | Read-only to a child form, all of its property values in the property browser are disabled |
| **Protected** | Accessible within the class and from any class that inherits from the class that declared this member |
| **Public** | Most permissive level. Public controls have full accessibility |

****************************ILLEGAL FOR NON-TRAINER USE******************************

### Introduction

Not only can you inherit controls and properties from a base form, you can also inherit code. This means that a code library can be built to enhance code reuse. The major advantage of using Inheritance is that you can override the code if it is not applicable in a particular circumstance.

To be able to override an event in the child form, you must make it visible to the child. This means that it must be defined as **Public** or **Protected** (protected is the default for events), and its **Modifiers** keyword cannot be private.

### Access modifiers

The **Modifiers** property determines the accessibility level of a control, such as how a control behaves and what functionality it has when its form is used as a base form.  An inherited form displays the controls in a different shade of grey depending upon the value of this property. Some of the values of the **Modifiers** property include **Public**, **Private**, and **Protected**.

The values of the control's properties are the same as those on the parent object, and when they are altered on the child form, that property becomes bold in the Properties window. To reset all values to those held by the parent, right-click the Properties window, and then click **Reset**.

- **Private**

  A **Private** control is read-only to a child form. Because a **Private** control cannot be modified, all of its property values in the Properties window are disabled. Copying this control elsewhere on the form or project produces a fully editable version.

- **Protected**

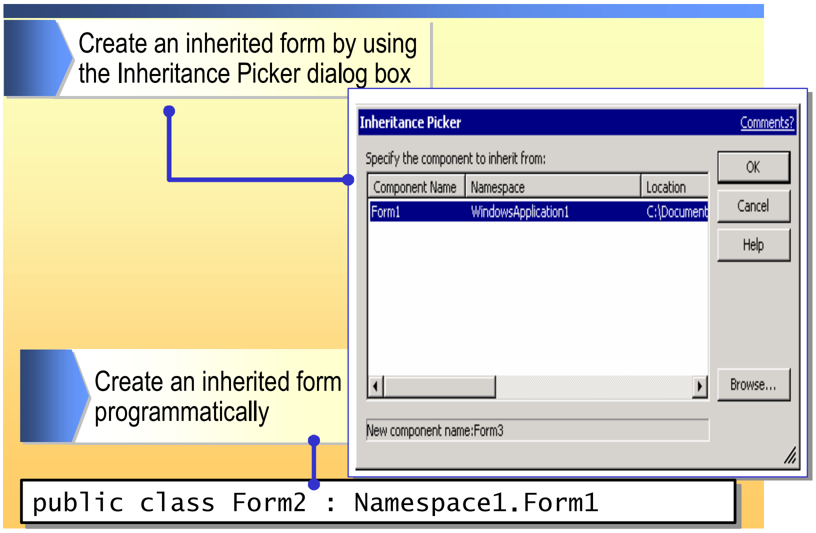
  A protected member is accessible within the class and from any class that inherits from the class that declared this member. If a change is made to a **Protected** control on a child form, then those changes will remain even if a change is made to the parent form. For the other types of controls, changes to the parent form will override those made to the child.

- **Public**

  **Public** is the most permissive level. **Public** controls have full accessibility.

**Note** The **Public**, **Protected**, and **Private** property values are the three property values that are common across all the .NET Language projects. C# supports two other values, **Internal** and **Protected Internal**.

# How to Create an Inherited Form



Create an inherited form by using the Inheritance Picker dialog box

Specify the component to inherit from:

| Component Name | Namespace | Location |
|---|---|---|
| Form1 | WindowsApplication1 | C:\Document |

New component name:Form3

Create an inherited form programmatically

```
public class Form2 : Namespace1.Form1
```

****************************ILLEGAL FOR NON-TRAINER USE******************************

**Introduction**

Visual Inheritance offers many advantages to developers. If you already have designed a form for a different project that is similar to the one you need in the current project, you can inherit from the earlier form. It also means that you can create a base form as a template to use later. It is a useful way of duplicating the main functionality of particular forms without having to recreate them from the beginning. Changes to the base form will be reflected in those forms that are inherited from it; so changes to the underlying template form will change all forms based on it.

**Procedure: Inheriting from an existing form**

There are two ways of implementing Visual Inheritance in a Visual Studio .NET project.

To create an inherited form programmatically:

1. Create a new project in Visual Studio .NET.
2. Add another form, or view the code of Form1, which is created by default.
3. In the class definition, add a reference to the form to inherit from. The reference should include the namespace that contains the form, followed by a period, and then the name of the base form itself.

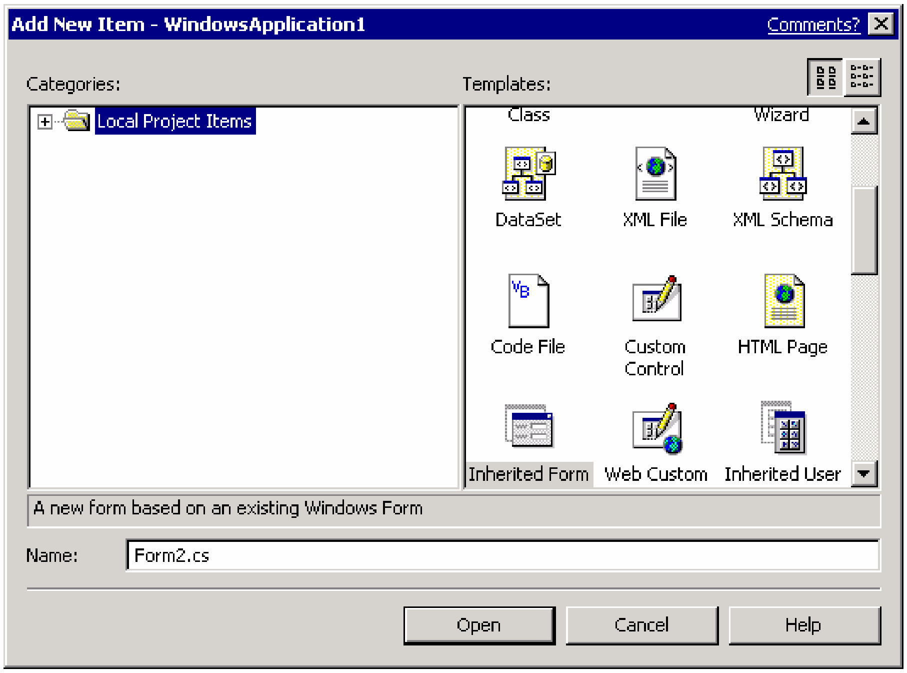
   ```
   public class Form2 : Namespace1.Form1
   ```

   The form now takes on the characteristics of the inherited form. It contains the controls that were on the inherited form, the code, and the properties.

To create an inherited form by using the **Inheritance Picker** dialog box:

**Note** Make sure that you build the solution before you inherit from an existing form in the project.

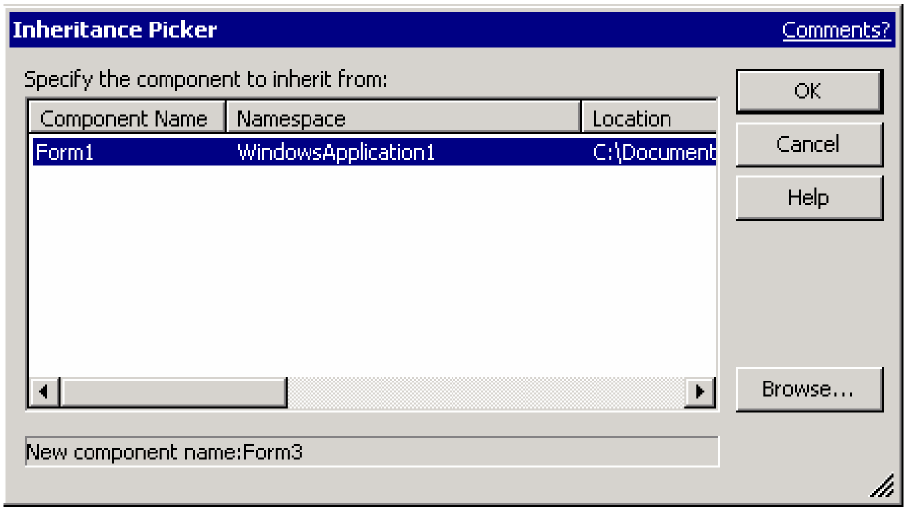
1. On the **Project** menu, click **Add Inherited Form**. The dialog box is identical for C# and for Visual Basic.



A new form based on an existing Windows Form

Name: Form2.cs

2. In the Categories pane, click **Local Project Items**, and in the Templates pane, click **Inherited Form**. In the **Name** box, type a name, and then click **Open**. This will open the **Inheritance Picker** dialog box.



Specify the component to inherit from:

| Component Name | Namespace | Location |
|---|---|---|
| Form1 | WindowsApplication1 | C:\Document |

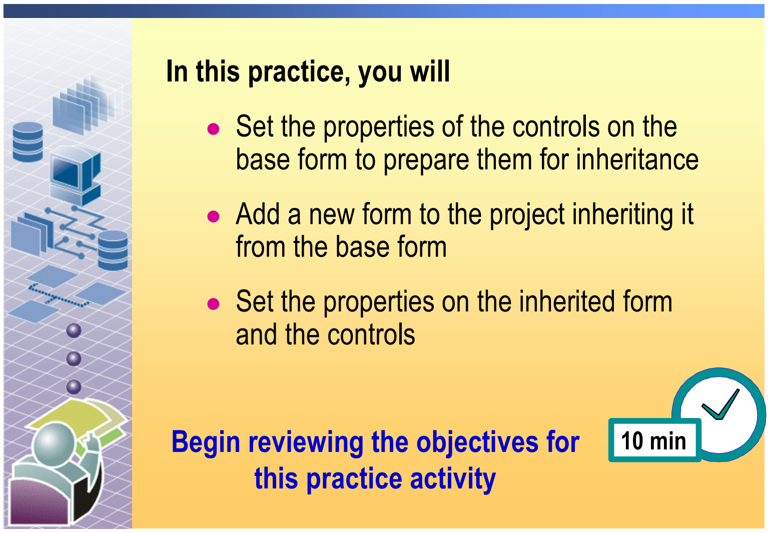
New component name:Form3

3. Click **Browse**, and locate the compiled executable of the project that contains your form. Click **OK**.

The new form should now be added into the project and be based on your inherited form. It contains the controls on the base form.

**Important** After the new form has been added and has been inherited from the base form, the project must be rebuilt to complete the inheritance relationship. The new form can then be displayed in Design view.

# Practice: Creating an Inherited Form



In this practice, you will

- Set the properties of the controls on the base form to prepare them for inheritance
- Add a new form to the project inheriting it from the base form
- Set the properties on the inherited form and the controls

Begin reviewing the objectives for this practice activity

**Introduction**

In this practice, you will modify an existing form to allow it to be inherited by other forms. You will create a new form and inherit it from the base form. You will then modify the inherited form to customize it for a particular application.

**Instructions**

### ► Open the project file for this practice activity

1. Use Windows Explorer, and browse to *install_folder*\Practices\Mod01\Mod01_02\Starter.
2. Double-click the WindowsCalculator.sln solution file to open the project.

### ► Modify a form's properties

1. If Solution Explorer is not visible, on the **View** menu, click **Solution Explorer**.
2. If the Properties window is not visible, on the **View** menu, click **Properties Window**.
3. Open BaseAboutForm.cs in Design view.
4. Click the **Product Name** label, and ensure that the **Modifiers** property is set to **Protected**.
5. Ensure that the **Modifiers** property is set to the given values for the following controls.

| Control | Modifiers Property |
|---|---|
| **Version <1.0.0000>** Label | **Protected** |
| **Copyright © yyyy Contoso, Ltd.** Label | **Protected** |
| **All Rights Reserved** Label | **Protected** |
| **OK** button | **Protected** |

6. Save the project.

### ► Add an inherited form

1. From the **Project** menu, click **Add New Item**.
2. In the Categories pane, click **Local Project Items**.
3. In the Templates pane, click **Inherited Form**.
4. In the **Name** field, type **AboutForm.cs** and then click **Open** to open the form.
5. In the **Inheritance Picker** dialog box, select **BaseAboutForm**.
6. Build the project.

   This creates the inheritance relationship between the two forms, including the access modifiers for the forms and controls.
7. Open **AboutForm.cs** and switch to Design view.
8. Set the form's **Size** property to **504,216**.
9. Set the **BackColor** property of the form to **Control**.
10. Set the **Text** property to **About Simple Windows Calculator**.
11. Click the **Product Name** label. Set the **Text** property to **Simple Windows Calculator**.
12. Set the properties for the following controls.

| Control | Properties |
|---|---|
| **Version <1.0.0000>** Label | Text: **Version 2.53.1892** |
| **Copyright © yyyy Contoso, Ltd.** Label | Text: **Copyright © 2002 Contoso, Ltd.** |

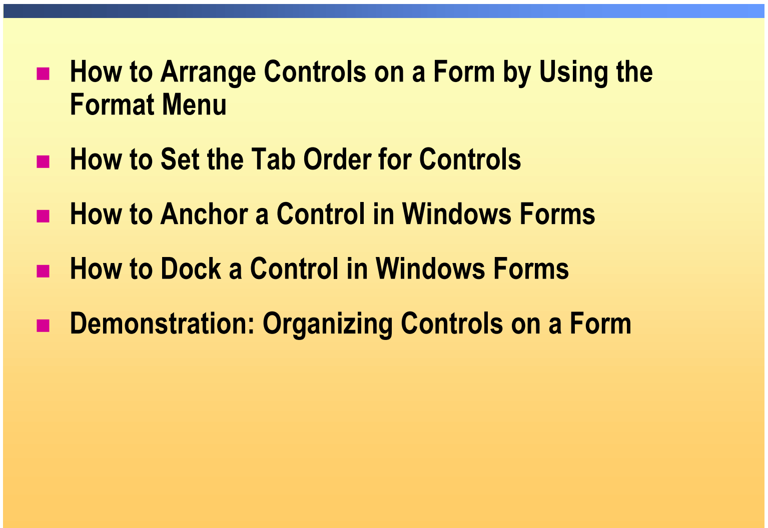
### ► Implement the About menu item and test the application

1. Open the source code for CalcUI.cs.
2. On the **View** menu, click **Show Tasks**, and then click **Comment**.
3. In the Task List window, locate the TODO comment. Add code to show the About window when the menu item is clicked:

```
AboutForm aboutForm = new AboutForm();
aboutForm.ShowDialog();
```

4. Build and run the application.
5. When the Simple Calculator window appears, on the **Help** menu, click **About**. The inherited **About** box will appear. Notice that the background of the **System Info** button is still the background set on the base form, but the other controls have the background color set in the inherited form.

# Lesson: Organizing Controls on a Form



- How to Arrange Controls on a Form by Using the Format Menu
- How to Set the Tab Order for Controls
- How to Anchor a Control in Windows Forms
- How to Dock a Control in Windows Forms
- Demonstration: Organizing Controls on a Form

### Introduction

Often you want to change the position and dimensions of controls at run time. Currently, you handle the **resize** event (WM_SIZE for Application Programming Interface [API] programmers), calculate the new position, width, and height, and call some methods like **Move** or **SetWindowPos**. The Windows Forms library offers two very helpful concepts to simplify these proceedings: anchoring and docking. In addition, Visual Studio .NET provides the **Format** menu, which allows you to arrange controls on a form.

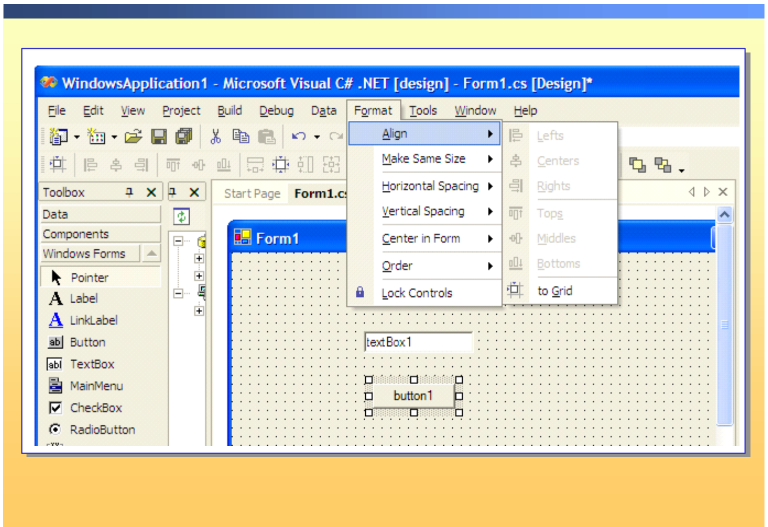
This lesson covers the **Format** menu and how to anchor and dock controls on a form. The lesson also covers setting the tab order for controls.

### Lesson objectives

After completing this lesson, you will be able to:

- Arrange controls on a form by using the **Format** menu.
- Set the tab order for the controls on a form.
- Anchor controls to a form.
- Dock controls on a form.

# How to Arrange Controls on a Form by Using the Format Menu



****************************ILLEGAL FOR NON-TRAINER USE******************************

## Introduction

You can use the **Format** menu or the Layout Toolbar in the Visual Studio Integrated Development Environment (IDE) to align, layer, and lock controls on a form.

## Format menu options

The **Format** menu provides many options for organizing controls. When you use the **Format** menu options for organizing controls, select controls so that the last control that you select is the primary control to which the others are aligned. The primary control has dark sizing handles whereas other controls have light sizing handles around them.

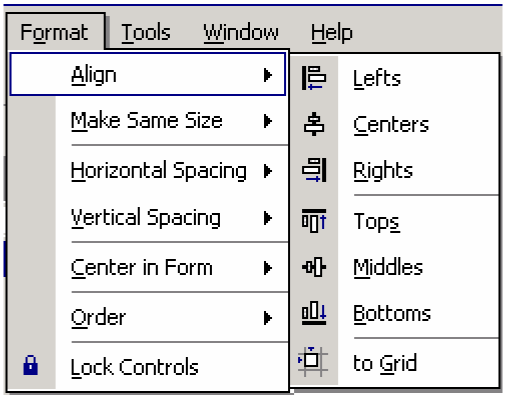
The choices and their functions are listed in the following table.

| Choice | Description |
|---|---|
| Align | Aligns all the controls with respect to the primary control |
| Make Same Size | Resizes multiple controls on a form |
| Horizontal Spacing | Increases horizontal spacing between controls |
| Vertical Spacing | Increases vertical spacing between controls |
| Center in Form | Centers the controls on a form |
| Order | Layers controls on a form |
| Lock Controls | Locks all controls on a form |

**Procedure: Arranging controls on a form**

To align multiple controls:

1. In the Windows Forms Designer, open the form that contains the controls that you want to position.
2. Select the controls that you want to align so that the last control that you select is the primary control to which the others are aligned.
   - On the **Format** menu, point to **Align**, and then click any of the seven choices available.

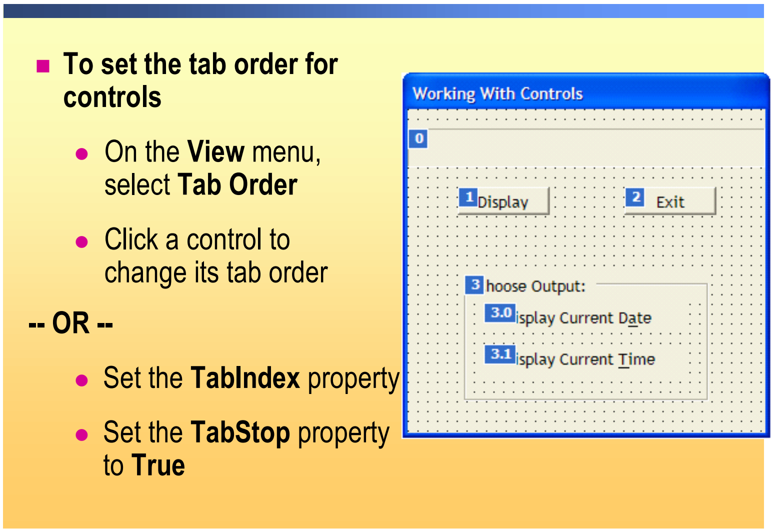


When creating complex user interfaces, you may want to layer controls on a form. To layer controls on a form:

1. Select a control.
2. On the **Format** menu, point to **Order**, and then click **Bring To Front** or **Send To Back**.

You can lock all controls on a form. This prevents any accidental moving or resizing of controls if you are setting other properties for controls. To lock all controls on a form, on the **Format** menu, click **Lock Controls**.

# How to Set the Tab Order for Controls



- **To set the tab order for controls**
  - On the **View** menu, select **Tab Order**
  - Click a control to change its tab order

-- OR --

  - Set the **TabIndex** property
  - Set the **TabStop** property to **True**

## Introduction

The tab order is the order in which a user moves focus from one control to another by pressing the TAB key. Each form has its own tab order. By default, the tab order is the same as the order in which you created the controls. Tab order numbering begins with zero.

## Procedure: Setting the tab order

You can set tab order in the Properties window by using the **TabIndex** property. The **TabIndex** property of a control determines where it is positioned in the tab order. By default, the first control drawn has a TabIndex value of **0**, the second has a TabIndex of **1**, and so on.

To set the tab order by using the **View** menu:

1. On the **View** menu, click **Tab Order**.
2. Click the controls sequentially to establish the tab order that you want.
3. When you are finished, on the **View** menu, click **Tab Order**.

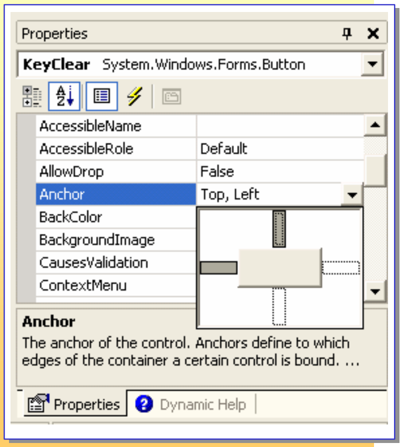
To set the tab order by using the **TabIndex** property:

1. Select the control.
2. Set the **TabIndex** property to the required value.
3. Set the **TabStop** property to **True**.

   By turning off the **TabStop** property, you enable a control be passed over in the tab order of the form. A control in which the **TabStop** property has been set to **False** still maintains its position in the tab order, even though the control is skipped when you cycle through the controls with the TAB key.

# How to Anchor a Control in Windows Forms

- **Anchoring**
  - Ensures that the edges of the control remain in the same position with respect to the parent container
- **To anchor a control to the form**
  - Set its **Anchor** property
  - Default value: **Top**, **Left**
  - Other Styles: **Bottom**, **Right**



Properties

**KeyClear** System.Windows.Forms.Button

| | |
|---|---|
| AccessibleName | |
| AccessibleRole | Default |
| AllowDrop | False |
| Anchor | Top, Left |
| BackColor | |
| BackgroundImage | |
| CausesValidation | |
| ContextMenu | |

**Anchor**

The anchor of the control. Anchors define to which edges of the container a certain control is bound. ...

### Introduction

If you are designing a form that the user can resize at run time, the controls on the form should resize and reposition properly. When a control is anchored to a form (or another container) and the form is resized, the control maintains the distance between the control and the anchor positions (which is the initial position). You use the **Anchor** property editor to anchor.

### Procedure: Anchoring a control on a form

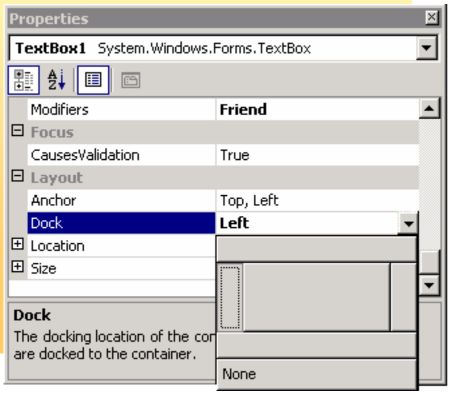
To anchor a control on a form:

1. Select the control that you want to anchor.
2. In the **Properties** window, click the **Anchor** property, and then click the **Anchor** arrow.

   The **Anchor** property editor is displayed; it contains a top bar, left bar, right bar, and bottom bar.
3. To set an anchor, click the top, left, right, or bottom bar in the **Anchor** property editor. Controls are anchored to the top and left by default. To clear a side of the control that has been anchored, click the bar on that side.

# How to Dock a Control in Windows Forms

- **Docking**
  - Enables you to glue the edges of a control to the edges of its parent control
- **To dock a control**
  - Set the Dock property



Properties

**TextBox1** System.Windows.Forms.TextBox

| | |
|---|---|
| Modifiers | **Friend** |
| ⊟ Focus | |
| CausesValidation | True |
| ⊟ Layout | |
| Anchor | Top, Left |
| Dock | **Left** |
| ⊞ Location | |
| ⊞ Size | |

**Dock**
The docking location of the cor
are docked to the container.

None

****************************ILLEGAL FOR NON-TRAINER USE******************************

## Introduction

You can dock controls to the edges of your form. For example, Windows Explorer docks the **TreeView** control to the left side of the window and the **ListView** control to the right side of the window. Use the **Dock** property for all visible Windows Forms controls to define the docking mode.

If you use the **Dock** property, a control is coupled with two edges of the container. Then, the control is horizontally or vertically resized when the container is resized. In real life, you do not indicate the edges, you indicate a border. If you dock a control to the left border, it is connected with the top left and bottom left edge. The value of the **Dock** property is one of the **DockStyle** values.

A special case is **DockStyle.Fill**. This value docks the control to all edges. The control fills in the complete client area of the container.

## Procedure: Docking a control on a form

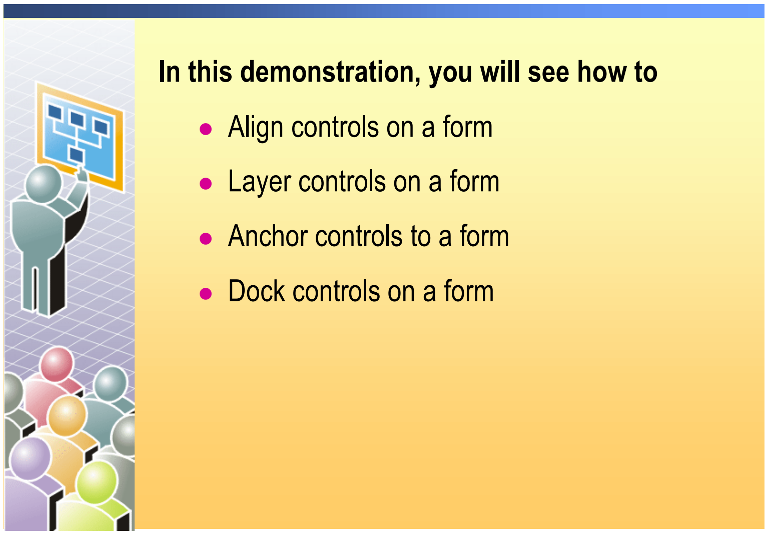
To dock a control on a form:

1. Select the control that you want to dock.
2. In the Properties window, click the arrow to the right of the **Dock** property.

   The **Dock** property editor is displayed; it contains a series of buttons that represent the edges and the center of the form.
3. Click the button that represents the edge of the form where you want to dock the control. To fill the contents of the control's form or container control, click the **Fill** (center) button. Click **None** to disable docking.

   The control is automatically resized to fit the boundaries of the docked edge.

# Demonstration: Organizing Controls on a Form



**In this demonstration, you will see how to**

- Align controls on a form
- Layer controls on a form
- Anchor controls to a form
- Dock controls on a form

**Introduction**

In this demonstration, you will see how to organize controls on a form.

**Instructions**

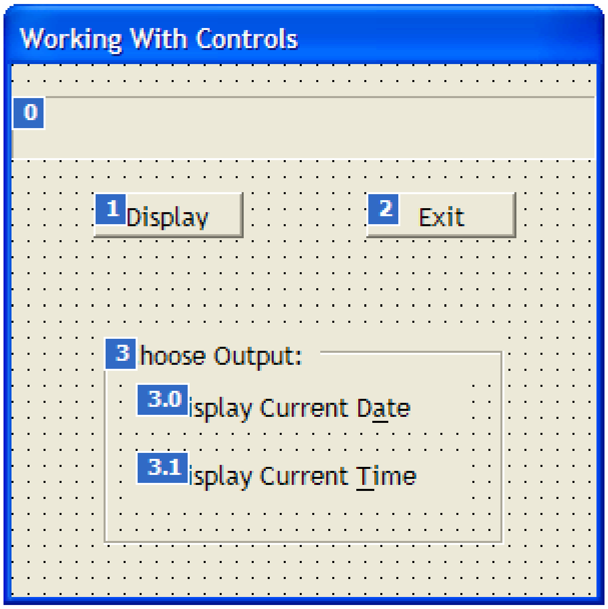
► **Organizing controls by using the Format menu**

1. Open the WorkingWithControls.sln solution in Visual Studio .NET from *install_folder*\Democode\ Mod01\Mod01_01\Starter.
2. If the WorkingWithControls form is not visible, display it in Design view. You will notice that the controls on the forms are not organized very well.
3. Organize the buttons on the form so that the **Display** button is positioned to the left of the **Exit** button.
4. Click the **Exit** button, hold down CTRL, and click the **Display** button. Both buttons are selected.
5. On the **Format** menu, click **Align**, and then click **Tops**. The **Exit** button aligns with the top of the **Display** button.
6. While the two buttons are selected, on the **Format** menu, click **Center in Form**, and then click **Horizontally**.
7. Select the **Choose Output** group box.
8. On the **Format** menu, click **Center in Form**, and then click **Horizontally**.
9. While the group box is selected, press the **up arrow** key twice to move the group box up.
10. Select the **Display Current Time** option button, hold down CTRL, and then click the **Display Current Date** option button.
11. On the **Format** menu, click **Align**, and then click **Lefts**.

12. While the two option buttons are selected, on the **Format** menu, click **Vertical** spacing, and then click **Increase**. Perform this step a second time further to increase the space between the two controls.
13. While the option buttons are selected, on the **Format** menu, click **Center in Form**, and then click **Horizontally**.
14. While the option buttons are selected, on the **Format** menu, click **Center in Form**, and then click **Vertically**.

### ► Set the tab order

1. On the **View** menu, click **Tab Order**.
2. Change the tab order by clicking each of the controls. Click the controls in the following order: **Label**, **Display** button, **Exit** button, **Choose Output** group box, **Display Current Date** option button, and **Display Current Time** option button. The resulting tab order will look like this.

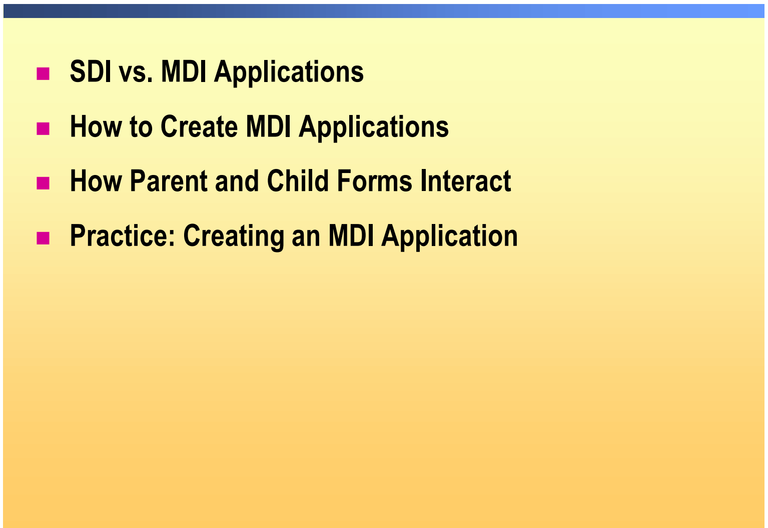


3. On the **View** menu, click **Tab Order**.

### ► Anchor and dock the controls

1. Click the **Label** control, and then set the **Dock** property to **Top**.
2. Click the **Exit** button, and then set the **Anchor** property to **Top, Right**.
3. Click the **Groupbox** control, and then set the **Anchor** property to **Bottom, Left, Right**.
4. Click the **Display Current Date** option button, and then set the **Anchor** property to **Top**.
5. Click the **Display Current Time** button, and then set the **Anchor** property to **Bottom**.
6. Build and run the application.
7. When the form appears, resize it to see how the controls behave with regard to their positioning.

# Lesson: Creating MDI Applications



- SDI vs. MDI Applications
- How to Create MDI Applications
- How Parent and Child Forms Interact
- Practice: Creating an MDI Application

**Introduction**

When creating Windows-based applications, you can use different styles for the user interface. Your application can have a single-document interface (SDI) or a multiple-document interface (MDI), or you can create an explorer-style interface. For more information about different types of application interfaces, see the .NET Framework software development kit (SDK).

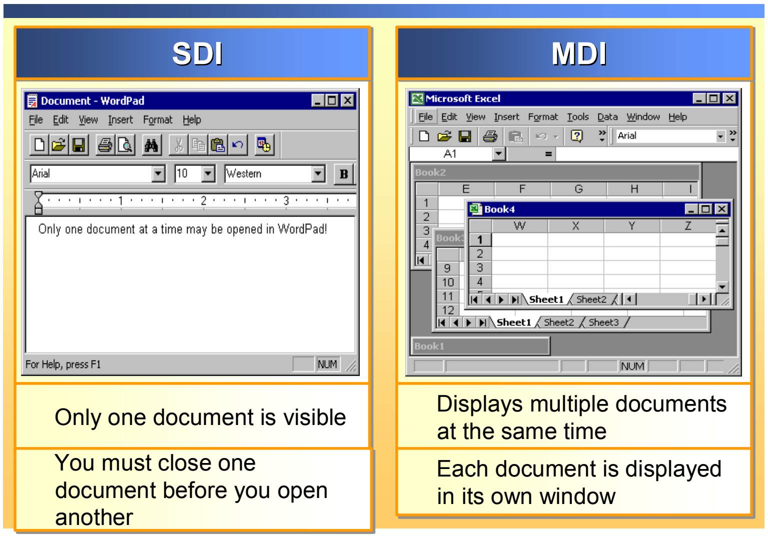
This lesson covers how to create and use MDI applications.

**Lesson objectives**

After completing this lesson, you will be able to:

- List the differences between SDI and MDI applications.
- Create MDI applications.
- Explain how parent and child forms interact.

# SDI vs. MDI Applications



SDI

Only one document at a time may be opened in WordPad!

Only one document is visible

You must close one document before you open another

MDI

Displays multiple documents at the same time

Each document is displayed in its own window

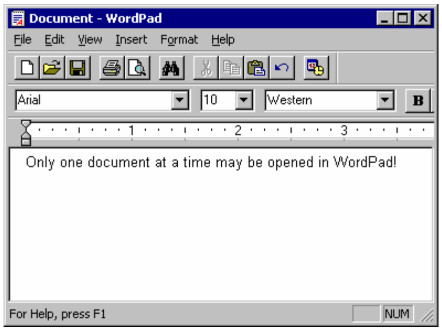
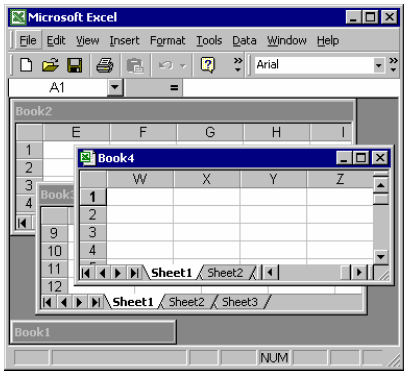
Document - WordPad
File Edit View Insert Format Help
For Help, press F1
Microsoft Excel
File Edit View Insert Format Tools Data Window Help
Book2
Book4
Book1
Sheet1
Sheet2
Sheet3

## Introduction

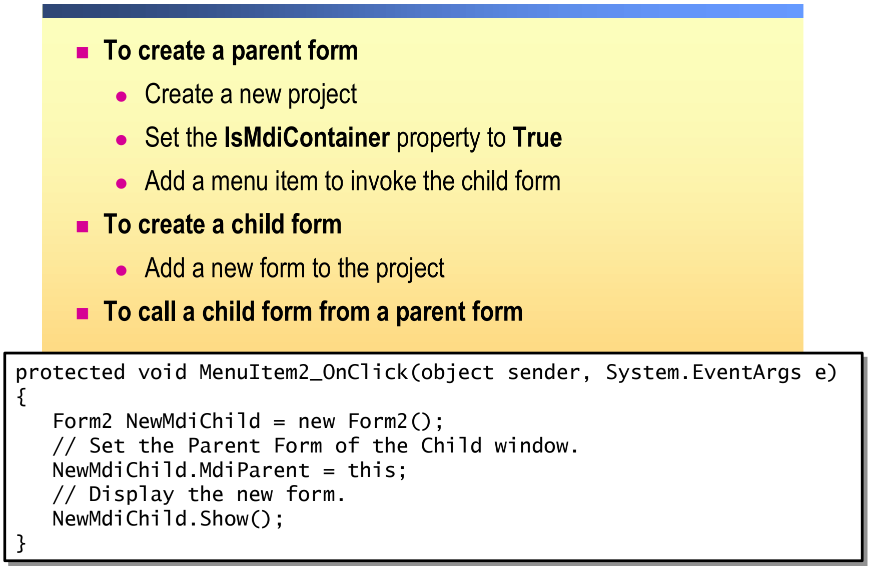
Before you create a Windows-based application, you must determine the style of user interface for the application.

## SDI vs. MDI

As the name suggests, single-document interface (SDI) applications can support only one document at a time, whereas a multiple-document interface (MDI) application can support several documents simultaneously. The following table lists the differences between SDI and MDI applications and also provides examples of which interface style to use for which scenario.

| SDI | MDI |
|---|---|
| Only one document is visible at a time. | Several documents are visible at the same time. |
| Must close one document before opening another. | Each document is displayed in its own window. |
| **Example**: Microsoft WordPad | **Example**: Microsoft Excel |
| <br>Only one document at a time may be opened in WordPad! |  |
| **Scenario**: A calendar application (because you may not need more than one instance of a calendar open at a time). | **Scenario**: An insurance application in which the user needs to work with multiple application forms. |

# How to Create MDI Applications



- To create a parent form
  - Create a new project
  - Set the **IsMdiContainer** property to **True**
  - Add a menu item to invoke the child form
- To create a child form
  - Add a new form to the project
- To call a child form from a parent form

```
protected void MenuItem2_OnClick(object sender, System.EventArgs e)
{
   Form2 NewMdiChild = new Form2();
   // Set the Parent Form of the Child window.
   NewMdiChild.MdiParent = this;
   // Display the new form.
   NewMdiChild.Show();
}
```

**Introduction**

There are three main steps involved in creating an MDI application: creating a parent form, creating a child form, and calling the child form from a parent form.

**Procedure: Creating MDI applications**

To create the parent form at design time:

The Parent form in an MDI application is the form that contains the MDI child windows. Child windows are used for interacting with users in an MDI application.

1. Create a new project.
2. In the Properties window, set the **IsMdiContainer** property to **True**.

   This designates the form as an MDI container for child windows.

   > **Note** When you are setting properties in the Properties window, you can also set the **WindowState** property to **Maximized**. This allows you to easily manipulate MDI child windows when the parent form is maximized.

3. From the Toolbox, drag a **MainMenu** component to the form.

   You need a menu to invoke the child forms from the parent form. As an example, create a top-level menu item with the **Text** property set to **&File** with submenu items called **&New** and **&Close**. Also, create a top-level menu item called **&Window**.

4. Set the **MdiList** property of the **Windows** menu item to **True**.

MDI child forms are critical for MDI applications because users interact with the application through child forms.

To create the child form at design time:

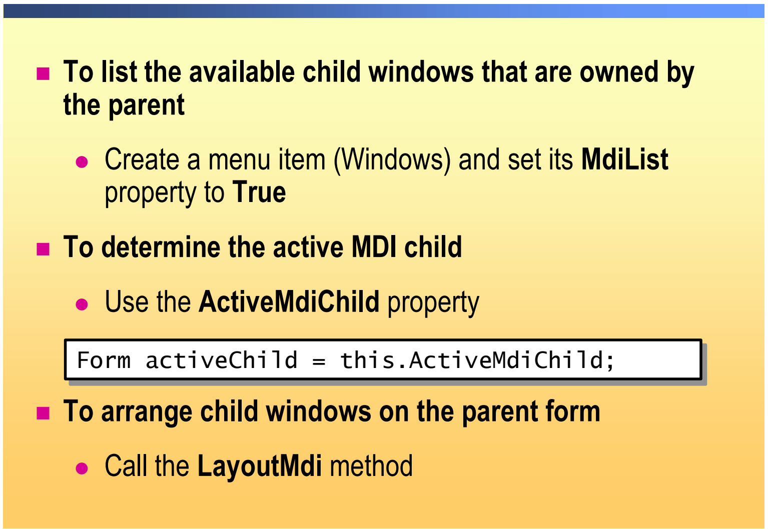
- In the same project that contains the parent form, create a new form.

To call the child form from the parent form:

1. Create a **Click** event handler for the **New** menu item on the parent form.
2. Insert code similar to the following code to create a new MDI child form when the user clicks the **New** menu item.

```
protected void MenuItem2_OnClick(object sender,
System.EventArgs e)
{
   Form2 NewMdiChild = new Form2();
   // Set the Parent Form of the Child window.
   NewMdiChild.MdiParent = this;
   // Display the new form.
   NewMdiChild.Show();
}
```

# How Parent and Child Forms Interact



- **To list the available child windows that are owned by the parent**
  - Create a menu item (Windows) and set its **MdiList** property to **True**
- **To determine the active MDI child**
  - Use the **ActiveMdiChild** property

```
Form activeChild = this.ActiveMdiChild;
```

- **To arrange child windows on the parent form**
  - Call the **LayoutMdi** method

### Introduction

In an MDI application, a parent form has several child forms, and each of the child forms interacts with the parent form. Visual Studio .NET includes several properties that allow parent and child forms in an MDI application to interact.

### Procedure: Listing the child windows of a parent form

An easy way to keep track of the different MDI child windows an application has open is to use a **Window** list. The functionality to keep track of all the open MDI child forms as well as which child form has focus is part of Visual Studio .NET and is set with the **MdiList** property of a menu item.

To list the child windows of a parent form by using the Window list:

1. Add a **MainMenu** component to the parent form.
2. Add the following top-level menu items to the **MainMenu** component by using the Menu Designer.

| Menu item | Text |
|---|---|
| MenuItem1 | &File |
| MenuItem2 | &Window |

3. Set the **MdiList** property of the **Windows** menu item to **True**.

### Procedure: Determining the active child form

When you are completing certain procedures in an application, it is important to determine the active form.

Because an MDI application can have many instances of the same child form, the procedure must know which form to use. To specify the correct form, use the **ActiveMdiChild** property, which returns the child form that has the focus or that was most recently active.

Use the following code to determine the active child form.

```
Form activeChild = this.ActiveMdiChild;
```

### Procedure: Arranging child windows on the parent form

To arrange child windows on a parent form, you can use the **LayoutMdi** method with the **MdiLayout** enumeration to rearrange the child forms in an MDI parent form.

There are four different **MdiLayout** enumeration values that can be used by the **LayoutMdi** method. These values help you display the form as cascading, horizontally or vertically tiled, or as child form icons arranged along the lower portion of the MDI form.

To arrange child forms, in an event, use the **LayoutMdi** method to set the **MdiLayout** enumeration for the MDI parent form.

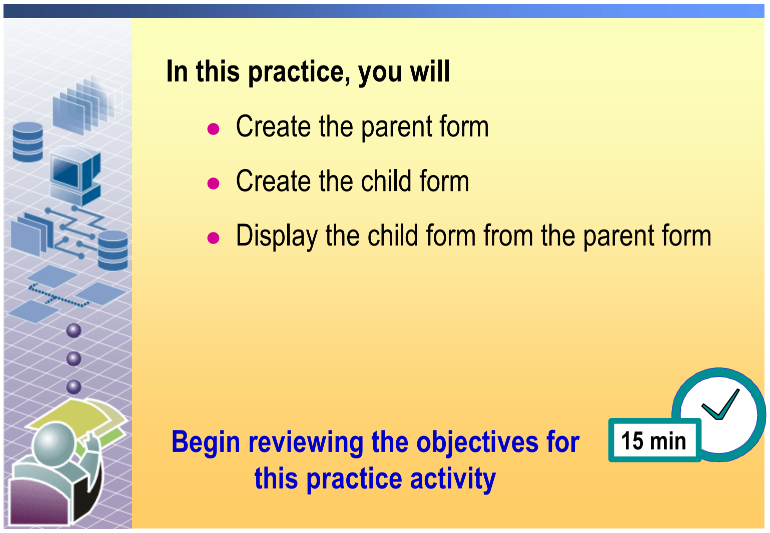
You can use the following members of the **MdiLayout** enumeration when calling the **LayoutMdi** method of the **Form** class.

| Member | Description |
|---|---|
| ArrangeIcons | All MDI child icons are arranged in the client region of the MDI parent form. |
| Cascade | All MDI child windows are cascaded in the client region of the MDI parent form. |
| TileHorizontal | All MDI child windows are tiled horizontally in the client region of the MDI parent form. |
| TileVertical | All MDI child windows are tiled vertically in the client region of the MDI parent form. |

The following example uses the Cascade setting of the **MdiLayout** enumeration for the child windows of the MDI parent form (Form1).

```
protected void CascadeWindows_Click(object sender,
  System.EventArgs e)
{
  Form1.LayoutMdi(System.Windows.Forms.MdiLayout.Cascade);
}
```

# Practice: Creating an MDI Application



In this practice, you will

- Create the parent form
- Create the child form
- Display the child form from the parent form

Begin reviewing the objectives for this practice activity

**Introduction**

In this practice, you will create an MDI application.

**Instructions**

### ► Open the practice project

1. Use Windows Explorer, and browse to *install_folder*\Practices\Mod01\Mod01_03\Starter.
2. Double-click the MdiApplication.sln solution file to open the project.

### ► Create the parent form

1. Open ParentForm.cs in Design view.
2. Set the **IsMdiContainer** property to **True**.

### ► Create the File menu

1. Open the Toolbox, add the **MainMenu** control to the form, and set its **Name** property to **MdiMenu**.
2. Click the menu at the top of the form and set the name of the first menu item to **&File**.
3. Set the **Name** property of the **File** menu to **FileMenuItem** and the **MergeOrder** property to **0**.
4. Open the **File** menu.
5. Click the menu item that appears under **File**, and set its **Text** property to **&New**.
6. Set the **Name** property of the **New** menu to **NewMenuItem**.
7. Click the menu item that appears under **New**, and set its **Text** property to **&Exit**.

8. Set the **Name** property of the **Exit** menu to **ExitMenuItem**.
9. Double-click the **Exit** menu item to create the **Click** event handler.
10. In the **exit** event handler, add the following code:

```
this.Close();
```

### ► Create the Window menu

1. Switch to Design view.
2. Click the second menu item to the right of **File**, and set its **Text** property to **&Window**.
3. Set the **Name** property of the **Window** menu to **WindowMenuItem** and the **MergeOrder** property to **2**.
4. Set the **MdiList** property of the **Window** menu item to **True**.
5. Open the **Window** menu.
6. Click the menu item that appears under **Window**, and set its **Text** property to **&Cascade**.
7. Set the **Name** property of the **Cascade** menu to **WindowCascadeMenuItem**.
8. Click the menu item that appears under **Cascade**, and set its **Text** property name to **&Tile**.
9. Set the **Name** property of the **Tile** menu to **WindowTileMenuItem**.
10. Double-click the **Cascade** menu, and add the following code to the **Click** event handler:

```
this.LayoutMdi (System.Windows.Forms.MdiLayout.Cascade);
```

11. Return to Design view and double-click the **Tile** menu item.
12. Add the following code to the **Click** event handler for the **Tile** menu item:

```
this.LayoutMdi
    (System.Windows.Forms.MdiLayout.TileHorizontal);
```

### ► Create the child form

1. Open the **Project** menu, and click **Add Windows Form**.
2. Set the name of the form to **ChildForm.cs**.
3. Set the **Text** property of the form to **Child Form**.
4. From the Toolbox, drag a **RichTextBox** control to the form, and set its **Name** property to **ChildTextBox**.
5. Set the **Dock** property of the **RichTextBox** to **Fill**.
6. Delete the existing value of the **Text** property of the **RichTextBox** and leave it blank.
7. From the Toolbox, drag a **MainMenu** control to the form.
8. Set the **Name** property of the **MainMenu** control to **ChildWindowMenu**.
9. Click the menu at the top of the form, and set the text to **F&ormat**.
10. Set the **Name** property of the **Format** menu to **FormatMenuItem**, and set the **MergeOrder** property to **1**.

11. Click the entry below the **Format** menu, and set the text to **&Toggle Foreground**.
12. Set the **Name** property of the Toggle Foreground menu item to **ToggleMenuItem**.
13. Double-click the **Toggle Foreground** menu, and add the following code to the **Click** event handler:

```
if (ToggleMenuItem.Checked)
{
   ToggleMenuItem.Checked = false;
   ChildTextBox.ForeColor = System.Drawing.Color.Black;
}
else
{
   ToggleMenuItem.Checked = true;
   ChildTextBox.ForeColor = System.Drawing.Color.Blue;
}
```

### ► Display the child form from the parent form

1. View ParentForm in Design view.
2. Double-click the **New** menu item on the **File** menu to create the **Click** event handler.
3. Add the following code to the **Click** event for the **New** menu item:

```
ChildForm newChild = new ChildForm();
newChild.MdiParent = this;
newChild.Show();
```

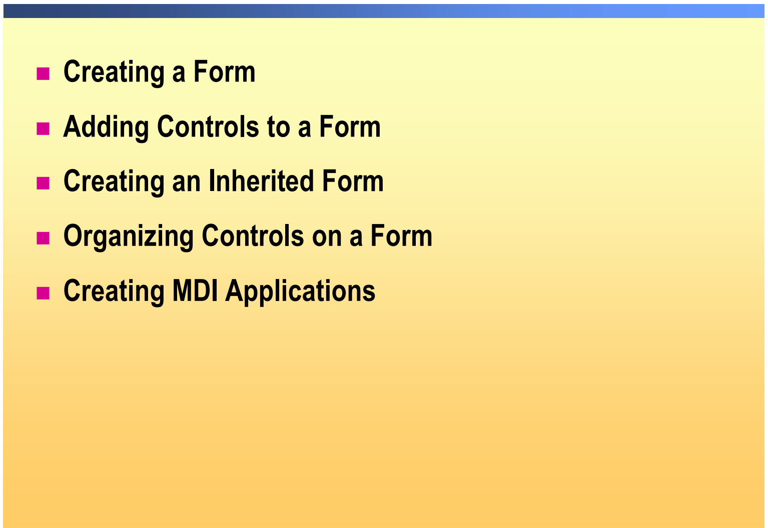
### ► Build and run the application

1. Build the application and run it.
2. When the parent form appears, on the **File** menu, click **New**.

   A new child window appears inside the parent window. Notice how the menu from the child window merges with the menu of the parent window and orders the menu according to the **MergeOrder** properties set in the **Display the child form from the parent form** procedure.
3. Type some text in the child form and use the Format menu to change the color of the text.
4. Open a few more child windows.
5. Click the **Window** menu, and select **Tile**. Notice how the child windows are reorganized in a tiled manner.
6. Close all the child windows.

   Notice that when the last child window is closed, the menu in the parent form changes and does not display the **Format** menu.
7. On the **File** menu, click **Exit** to end the application.

# Review



- Creating a Form
- Adding Controls to a Form
- Creating an Inherited Form
- Organizing Controls on a Form
- Creating MDI Applications

1. List some reasons why you would use Windows Forms as opposed to Web Forms.

   **Richer user interface**

   **Faster response time**

   **Better support for offline scenario**

2. What is Visual Inheritance?

   **The process of inheriting a form from an existing one is called Visual Inheritance.**

   **It means that if you already have designed a form for a different project that is similar to the one you need now, you can inherit from it. It also means that you can create a base form as a template for later use. It is a useful way of duplicating the main functionality of particular forms without having to recreate them from the beginning.**

3. What is the difference between anchoring and docking a control to a form?

   **When a control is anchored to a form (or another container) and the form is resized, the control maintains the distance between the control and the anchor positions (which is the initial position).**

   **In docking, a control is coupled with two edges of the container. The control is horizontally or vertically resized when the container is resized.**

4. What are the differences between SDI and MDI applications?

   **In an SDI application, only one document can be open at a time. You must close one document before opening another.**

   **In an MDI application, several documents can be open at the same time. Each document is displayed in its own window.**

5. When creating a form, what class must the form inherit from to make it a Windows Form?

   **System.Windows.Forms.Form**

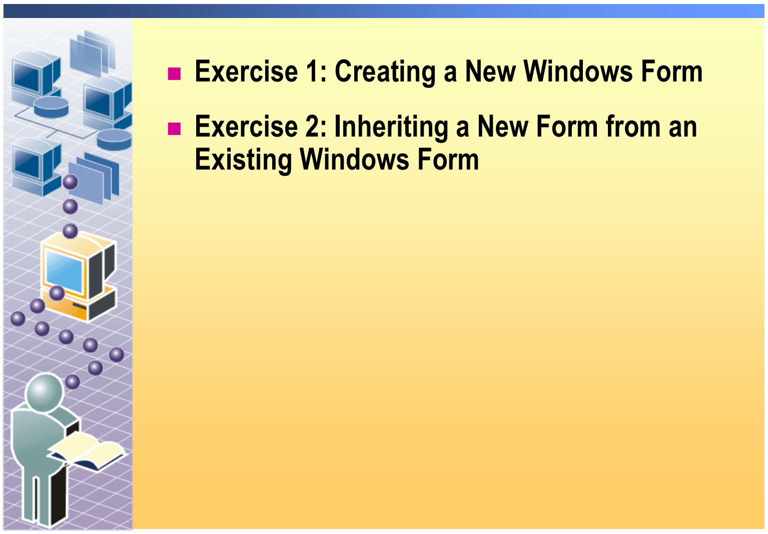
6. You want to perform some totaling of numbers as a form is being dismissed. Into which event handler should you add the code?

   **The code should be added to the Deactivated event handler.**

7. When creating a form that inherits from a base form, what must be available to override the base version of the methods of a control on the base form?

   **The Modifier property of the control on the base form must be set to either protected or public to override its functionality in the derived form.**

# Lab 1.1: Creating Windows Forms



- Exercise 1: Creating a New Windows Form
- Exercise 2: Inheriting a New Form from an Existing Windows Form

### Objectives

After completing this lab, you will have demonstrated your ability to:

- Create a new form.
- Inherit a new form from an existing form.
- Add controls to a form.
- Set form and control properties.

**Note** This lab focuses on the concepts in Module 1, “Introducing Windows Forms,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. As a result, this lab may not comply with Microsoft security recommendations.

### Prerequisites

Before working on this lab, you must have the knowledge and skills to develop a simple Windows Forms application by using a Visual Studio .NET–compatible programming language.

### Scenario

The Internal Business Application shell provides a common access point to various internal business applications. To ensure that the information provided by the application is viewed by the appropriate user, the application requires a logon form.

The logon form will prompt the user for his or her user name and password. The logon form will then attempt to authenticate the user’s credentials to determine if the user is permitted to access various internal applications.

In this lab, you will add a new form to the Internal Business Application shell and populate it with controls. You will also implement the **Click** event handler for the buttons on the logon form. In addition, you will create the **About** dialog box by inheriting a new form from an existing form.

### Estimated time to complete this lab: 30 minutes

# Exercise 1
# Creating a New Windows Form

In this exercise, you will update the Internal Business Application shell by adding a logon form and populating it with controls. You will also set form and control properties and implement the **Click** event handlers for the buttons on the logon form.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab01_1\Ex01\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab01_1\Ex01\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open the InternalBusinessApp project in Visual Studio .NET. Browse to *install_folder*\Labfiles\Lab01_1\Ex01\Starter to find this project.<br>**Note:** The project will not build until you complete this exercise. | **a.** For more information about opening a project file and starting an application, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. For additional information about starting an application from within Designer, in Index, search by using the phrase **Debugging Windows Applications**. |
| **2.** Add a new form to the project. Use the form name LoginForm, and use the file name LoginForm.cs. | **a.** For more information about Windows Forms, see the following resources:<br>• Practice: Creating a Form in Module 1, “Introducing Windows Forms,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. This practice contains information about how to add a new form to a project.<br>• Lesson: Creating a Form in Module 1, “Introducing Windows Forms,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. This lesson contains information about how to create a form.<br>• The Windows Forms section of the .NET Framework SDK documentation. |

<table>
<tr><th>Tasks</th><th>Additional information</th></tr>
<tr><td>

**3.** Set form properties. Use the following table to set the properties of the form.

<table>
<tr><th>Property</th><th>Value</th></tr>
<tr><td>(Name)</td><td>LoginForm</td></tr>
<tr><td>ControlBox</td><td>False</td></tr>
<tr><td>FormBorderStyle</td><td>Fixed3D</td></tr>
<tr><td>MaximizeBox</td><td>False</td></tr>
<tr><td>MinimizeBox</td><td>False</td></tr>
<tr><td>Size</td><td>322, 210</td></tr>
<tr><td>Text</td><td>Internal Business Application Logon</td></tr>
</table>

</td><td>

**a.** For more information about form properties and Windows Forms, see the following resources:

- Practice: Creating a Form in Module 1, “Introducing Windows Forms,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. This practice contains information about how to set form properties.
- Lesson: Creating a Form in Module 1, “Introducing Windows Forms,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- The Windows Forms section of the .NET Framework SDK documentation.

</td></tr>
<tr><td>

**4.** Add controls to the form. Add two labels, two text boxes, and two buttons to the form.

</td><td>

**a.** For more information about adding controls to a form and Windows Forms, see the following resources:

- Practice: Creating a Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- Lesson: Creating a Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- The Windows Forms section of the .NET Framework SDK.

</td></tr>
</table>

<table>
<tr><th>Tasks</th><th>Additional information</th></tr>
<tr><td>

**5.** Set the control properties. Use the following tables to set the properties of the controls.

<table>
<tr><th>Label1 Property</th><th>Value</th></tr>
<tr><td>(Name)</td><td>UserNameLabel</td></tr>
<tr><td>Location</td><td>64, 31</td></tr>
<tr><td>Size</td><td>63, 14</td></tr>
<tr><td>Text</td><td>User name:</td></tr>
</table>

<table>
<tr><th>Label2 Property</th><th>Value</th></tr>
<tr><td>(Name)</td><td>PasswordLabel</td></tr>
<tr><td>Location</td><td>64, 71</td></tr>
<tr><td>Size</td><td>64, 14</td></tr>
<tr><td>Text</td><td>Password:</td></tr>
</table>

<table>
<tr><th>Textbox1 Property</th><th>Value</th></tr>
<tr><td>(Name)</td><td>UserNameTextBox</td></tr>
<tr><td>Location</td><td>128, 29</td></tr>
<tr><td>Size</td><td>120, 20</td></tr>
<tr><td>Text</td><td><b>(Delete existing text and leave it blank)</b></td></tr>
</table>

<table>
<tr><th>Textbox2 Property</th><th>Value</th></tr>
<tr><td>(Name)</td><td>PasswordTextBox</td></tr>
<tr><td>Location</td><td>128, 64</td></tr>
<tr><td>PasswordChar</td><td>*</td></tr>
<tr><td>Size</td><td>120, 20</td></tr>
<tr><td>Text</td><td></td></tr>
</table>

<table>
<tr><th>Button1 Property</th><th>Value</th></tr>
<tr><td>(Name)</td><td>LogonButton</td></tr>
<tr><td>Location</td><td>67, 116</td></tr>
<tr><td>Size</td><td>75, 30</td></tr>
<tr><td>Text</td><td>&Log On</td></tr>
</table>

<table>
<tr><th>Button2 Property</th><th>Value</th></tr>
<tr><td>(Name)</td><td>CancelAppButton</td></tr>
<tr><td>Location</td><td>171, 116</td></tr>
<tr><td>Size</td><td>75, 30</td></tr>
<tr><td>Text</td><td>&Cancel</td></tr>
</table>

</td><td>

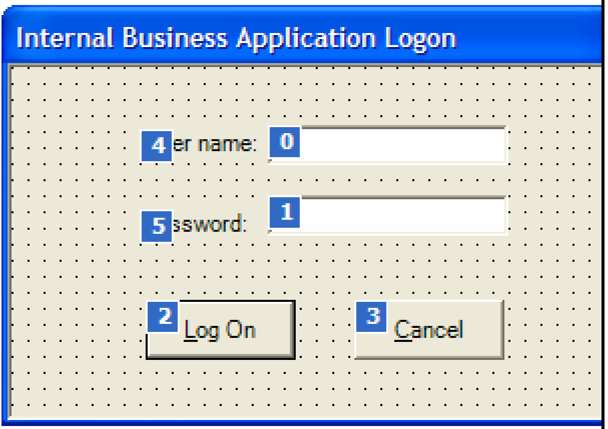
**a.** For more information about control properties and Windows Forms, see the following resources:

- Practice: Creating a Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. This practice contains information about how to set control properties.
- Lesson: Creating a Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- The Windows Forms section of the .NET Framework SDK.

</td></tr>
</table>

<table>
<tr><th>Tasks</th><th>Additional information</th></tr>
<tr><td>

**6.** Set the tab order for the controls on the form. The tab order should resemble the following diagram.



</td><td>

**a.** For more information about setting tab order on a form and Windows Forms, see the following resources:

- Lesson: Organizing Controls on a Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- The Windows Forms section in the .NET Framework SDK.

</td></tr>
<tr><td>

**7.** Complete the form properties. Use the following table to set the remaining properties of the form.

<table>
<tr><th>Property</th><th>Value</th></tr>
<tr><td>AcceptButton</td><td>LogonButton</td></tr>
<tr><td>CancelButton</td><td>CancelAppButton</td></tr>
</table>

</td><td>

**a.** For more information about setting form properties and Windows Forms, see the following resources:

- Practice: Creating a Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- Lesson: Creating a Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- The Windows Forms section of the .NET Framework SDK.

</td></tr>
<tr><td colspan="2">

*The following four steps (Steps 8, 9, 10, and 11) are not required to meet the objectives of the labs but are required to run the application and view results.*

</td></tr>
<tr><td>

**8.** Declare class member fields in the **LoginForm** class. Open the LoginFormCode.txt file, and copy the code under the heading *Declare these class members in the LoginForm class*.

</td><td>Additional information is not necessary for this task.</td></tr>
<tr><td>

**9.** Implement properties for the class. Open the LoginFormCode.txt file, and copy the code under the heading *Add these properties to the LoginForm class*.

</td><td>Additional information is not necessary for this task.</td></tr>
<tr><td>

**10.** Implement the **Click** event handler for the **Cancel** button. Open the LoginFormCode.txt file, and copy the required code under the heading *Create a Click event handler for the Cancel button and add the following code to the event handler.*

</td><td>Additional information is not necessary for this task.</td></tr>
</table>

| Tasks | Additional information |
|---|---|
| **11.** Implement the **Click** event handler for the **Log On** button. Open the LoginFormCode.txt file, and copy the required code under the heading *Create a Click event handler for the Log On button and add the following code to the event handler*. | Additional information is not necessary for this task. |
| **12.** Build and run the application. Specify **mario** for the user name and **P@ssw0rd** for the password. | **a.** For more information about working with forms and Windows Forms, see the following resources:<br>• Practice: Creating a Form, in Module 1, "Introducing Windows Forms," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Lesson: Creating a Form, in Module 1, "Introducing Windows Forms," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Windows Forms section of the .NET Framework SDK. |

# Exercise 2
# Inheriting a New Form from an Existing Windows Form

In this exercise, you will update the Internal Business Application shell by adding an About dialog by inheriting from a generic Windows Form.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab01_1\Ex02\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab01_1\Ex02\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

**Important** In the InternalBusinessApp.sln project in \install_folder\Labfiles\Lab01_1\Ex02\Solution, because the About Internal Business Application dialog box for the project is an inherited form, you must build the solution before you can view the AppControlAboutForm form in Windows Forms Designer.

| Tasks | Additional information |
|---|---|
| **1.** Open the InternalBusinessApp project in Visual Studio .NET. Browse to *install_folder*\Labfiles\Lab01_1\Ex02\Starter to find this project. | **a.** For more information about opening a project file and starting an application, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. For additional information about starting an application from within Designer, in Index, search by using the phrase **Debugging Windows Applications**. |
| **2.** Open the BaseAboutForm form in Design view. | **a.** For more information about adding new forms to a project and Windows Forms, see the following resources:<br>• Practice: Creating an Inherited Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Lesson: Creating an Inherited Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Windows Forms section of the .NET Framework SDK. |

<table>
<tr><th>Tasks</th><th>Additional information</th></tr>
<tr><td>
<b>3.</b> Use the following table to set the <b>Modifier</b> property of each control. To set a property for multiple controls simultaneously, use the CTRL key to select the controls.
<table>
<tr><th>Control</th><th>Modifier Property Value</th></tr>
<tr><td>ProductNameLabel</td><td>protected</td></tr>
<tr><td>VersionNumber</td><td>protected</td></tr>
<tr><td>CopyrightLabel</td><td>protected</td></tr>
<tr><td>AllRightsReservedLabel</td><td>protected</td></tr>
<tr><td>AboutOkButton</td><td>protected</td></tr>
</table>
</td><td>
<b>a.</b> For more information about adding new forms to a project and Windows Forms, see the following resources:<br>
• Practice: Creating an Inherited Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, <i>Developing Microsoft .NET Applications for Windows (Visual C# .NET)</i>.<br>
• Lesson: Creating an Inherited Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, <i>Developing Microsoft .NET Applications for Windows (Visual C# .NET)</i>.<br>
• The Windows Forms section of the .NET Framework SDK.
</td></tr>
<tr><td><b>4.</b> Save the BaseAboutForm, and build the project.</td><td>Additional information is not necessary for this task.</td></tr>
<tr><td>
<b>5.</b> Add a new form to the project by using the <b>Inheritance Picker</b> dialog box. Use the form name AppControlAboutForm. Inherit the form from the BaseAboutForm form. Save the new form, and build the project.
</td><td>
<b>a.</b> For more information about adding new forms to a project and Windows Forms, see the following resources:<br>
• Practice: Creating an Inherited Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, <i>Developing Microsoft .NET Applications for Windows (Visual C# .NET)</i>.<br>
• Lesson: Creating an Inherited Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, <i>Developing Microsoft .NET Applications for Windows (Visual C# .NET)</i>.<br>
• The Windows Forms section of the .NET Framework SDK.
</td></tr>
<tr><td>
<b>6.</b> Complete the properties on the AppControlAboutForm form. Use the following table to set the properties of the form.
<table>
<tr><th>Form</th><th>Value</th></tr>
<tr><td>BackColor</td><td>Control</td></tr>
<tr><td>Size</td><td>500, 212</td></tr>
<tr><td>Text</td><td>About Internal Business Application</td></tr>
</table>
Use the following table to set the properties of the controls.
<table>
<tr><th>Control Property</th><th>Value</th></tr>
<tr><td>ProductNameLabel.Text</td><td>Internal Business Application</td></tr>
<tr><td>VersionNumber.Text</td><td>Version 1.0.3153</td></tr>
<tr><td>CopyrightLabel.Text</td><td>Copyright © 2002 Contoso, Ltd.</td></tr>
</table>
</td><td>
<b>a.</b> For more information about working with inherited forms and Windows Forms, see the following resources:<br>
• Practice: Creating an Inherited Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, <i>Developing Microsoft .NET Applications for Windows (Visual C# .NET)</i>. This practice contains information about how to set inherited form and control properties.<br>
• Lesson: Creating and Inherited Form, in Module 1, “Introducing Windows Forms,” in Course 2555A, <i>Developing Microsoft .NET Applications for Windows (Visual C# .NET)</i>. This lesson contains information about how to work with inherited forms.<br>
• The Windows Forms section of the .NET Framework SDK.
</td></tr>
</table>

| Tasks | Additional information |
|---|---|
| **7.** In AppControlForm, implement click event handler for the **About** menu item. | **a.** For more detailed information about the tasks that you must perform, see the TODO comments in the code.<br>**b.** For more information about Windows Forms, see the following resource:<br>• The Windows Forms section of the .NET Framework SDK. |
| **8.** Run the application to test the inherited About dialog box. | **a.** For more information about Windows Forms, see the following resources:<br>• The Windows Forms section of the .NET Framework SDK. |

msdn® training

# Module 2: Working with Controls

**Contents**

Microsoft®

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

# Instructor Notes

**Presentation:**
**120 minutes**

**Lab:**
**30 minutes**

This module provides students with an overview of using Windows Controls to create Microsoft® .NET Framework Windows Forms applications. In the module, students will create event handlers for controls and use some of the controls in a Windows Forms application. They will also learn how to use dialog boxes and menus in a Windows Forms application. Students will then create controls at run time and validate user input in an application.

After completing this module, students will be able to:

- Create an event handler for a control.
- Select and use the appropriate controls in a Windows Forms application.
- Use dialog boxes in a Windows Forms application.
- Add controls to a form at run time.
- Create and use menus in a Windows Forms application.
- Validate user input in a Windows Forms application.

**Required materials**

To teach this module, you need Microsoft® PowerPoint® file 2555A_02.ppt.

**Preparation tasks**

To prepare for this module:

- Read all of the materials for this module.
- Complete the practices, demonstrations, and lab.

# How to Teach This Module

This section contains information that will help you to teach this module. The following are some tips on how to teach this module:

- To compress the time spent on this module, find out if you have a class consisting entirely of advanced students (students that are comfortable programming in an object-oriented programming environment and who are familiar with the Microsoft .NET Framework and event handling in Windows Forms). If yes, then you can try moving quickly through the slides and get the students started on practice activities as soon as possible. The practices include step-by-step instructions, so an advanced class should be able to do the practices very easily.
- Lab 2.1, "Working with Controls" is based on the Purchase Order application in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual Visual C#™ .NET)* and is intended to simulate a real-world environment in which students will demonstrate what they learned during the lecture and practice portions of the module. The lab does not provide step-by-step detailed instructions; instead, the students are given tasks to complete in the left column and a list of resources that they can use (if they need help) in the right column. Students get the hands-on experience that they need by completing the practice activities at the end of each lesson.
- In addition to the demonstration project that covers using a drag-and-drop operation to move data between controls, there are three demonstration projects that can be used to help you teach the content of this module. The following list identifies the design goal for each of the four demonstration projects. Only the demonstration that covers implementing simple drag-and-drop appears in the module content. The following are some demonstrations that do not appear in the module but will help you demonstrate some concepts:
  - Mod02_01\DelegateSample\DelegateSample.sln. This project demonstrates how to use a class to define a delegate and two methods that can be used to format a variable of type float. Two radio buttons are used to simulate different situations, each requiring a different method of handling the same event. When you click a button on the form, a text box is filled by using the currently assigned method.

    The DelegateSample project uses **RadioButton**, **Button**, and **TextBox** controls.
  - Mod02_02\EventHandlers\EventHandlers.sln. This project provides a general overview of event handlers. It covers adding an event to a handler at design time, using the *event arguments (e)* parameter, and adding and removing handlers at run time.

    The EventHandlers project uses **MainMenu**, **RadioButton**, and **Button** controls.
  - Mod02_03\ControlCollection\ControlCollection.sln. This project demonstrates how the **Controls** property of a container control can be used to affect the properties of the controls which it contains. As a side note, the **SelectedIndexChanged** event and **SelectedItem** property of a **ComboBox** are also demonstrated.

    The ControlCollection project uses **ComboBox** and **GroupBox** controls.

## Lesson: Creating an Event Handler for a Control

This section describes the instructional methods for teaching this lesson. The following are some tips on how to teach this lesson:

- Make sure that your students understand delegates in the generic sense before you introduce the **Delegate** class. Be sure that students understand when the **Delegate** keyword is used (and when it is not used).
- If students have any confusion regarding delegates, try running the DelegateSample demonstration to help explain how and when to create and use delegates.
- There are two additional demonstrations that can help you teach the content in this lesson. To run the first demonstration, open DelegateSample.sln from *install_folder*\Democode\Mod02\Mod02_01\Starter. To run the second demonstration, open EventHandlers.sln from *install_folder*\Democode\Mod02\Mod02_02\Starter.

## Lesson: Using Windows Forms Controls

This section describes the instructional methods for teaching this lesson. The following are some tips on how to teach this lesson:

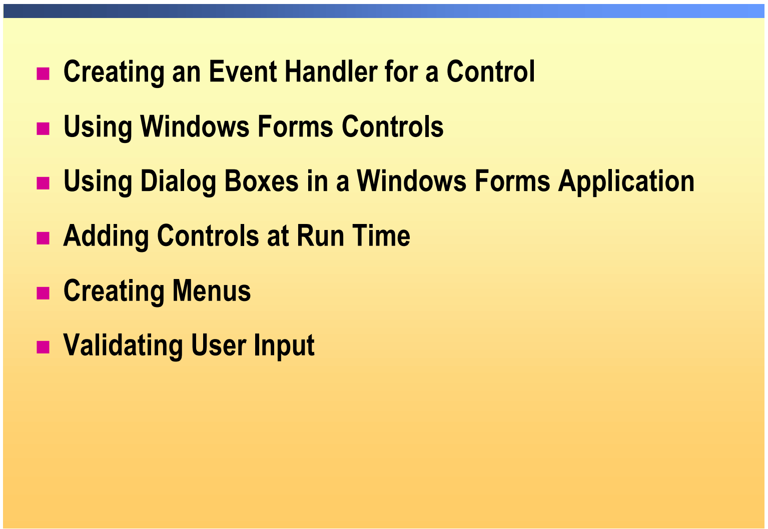
- There is a practice exercise on Toolbars at the end of this lesson. However, to show how other controls are used, show simple demonstrations of using **StatusBar**, **ListBox**, **GroupBoxes**, and **Panels** in a Windows Forms application.
- This lesson includes a demonstration on how to use a drag-and-drop operation to move and copy between controls on a form. The first example shows how the contents of a **TextBox** control can be copied to another TextBox, and covers the **AllowDrop** property, **DragEnter** event, **DragDrop** event, and **DragEventArgs** class. The second example uses two **TreeView** controls to show that some controls include events specific to drag-and-drop operations (in this case, the **DragItem** event). While observing how to use a drag-and-drop operation to copy data from one **TreeView** control to another, the concept of adding and removing nodes (**TreeNode** objects) from a **TreeView** control is also covered in this demonstration.

  The SimpleDragDrop project uses **TextBox** and **TreeView** controls.

## Lab 2.1: Working with Controls

- Make sure that you have demonstrated the two lab applications—the Expense Report application and the Purchase Order application—in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual Visual C# .NET)* before students begin the lab. To see how to demonstrate lab scenarios, see the Introduction module in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual Visual C# .NET).*
- Practice exercises will enable students to successfully complete the lab exercises. Therefore, make sure that students have completed all practice exercises before they begin the lab.

# Overview



- Creating an Event Handler for a Control
- Using Windows Forms Controls
- Using Dialog Boxes in a Windows Forms Application
- Adding Controls at Run Time
- Creating Menus
- Validating User Input

**Introduction**

When you design the user interface (UI) and develop the code that operates behind the UI of an application, you will need to work with controls and their events, properties, and methods to meet the design requirements that you have been given.

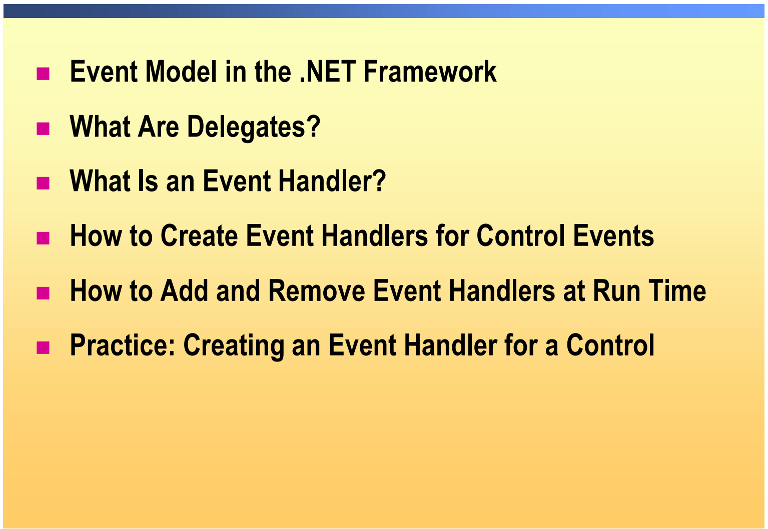
This module covers how to create event *procedures* (or handlers) in your application that will run in response to user actions. You will learn how to add programming logic to the event procedures of a control, how to use the Microsoft® .NET Framework Windows Forms intrinsic controls, dialog boxes, and menus, and how to validate the data entered by users of your application.

**Objectives**

After completing this module, you will be able to:

- Create an event handler for a control.
- Select and use the appropriate controls in a Windows Forms application.
- Use dialog boxes in a Windows Forms application.
- Add controls to a form at run time.
- Create and use menus in a Windows Forms application.
- Validate user input in a Windows Forms application.

# Lesson: Creating an Event Handler for a Control



- Event Model in the .NET Framework
- What Are Delegates?
- What Is an Event Handler?
- How to Create Event Handlers for Control Events
- How to Add and Remove Event Handlers at Run Time
- Practice: Creating an Event Handler for a Control

### Introduction

In the Microsoft® .NET Framework, an event is a message sent by an object to signal the occurrence of an action that is either invoked by a user or programmatically. Each event has a sender that raises the event and a receiver that handles the event.

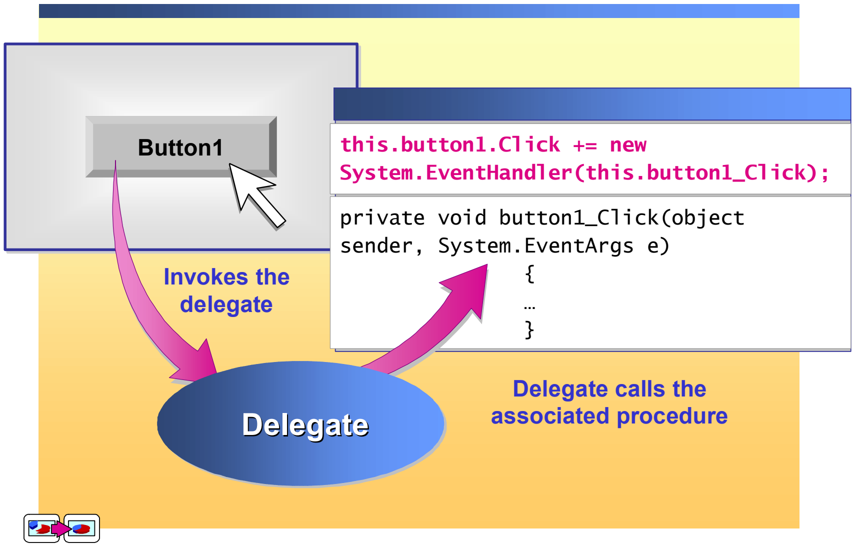
In this lesson, you will learn about events and the ways in which events can be handled in your application. You will then learn how to create procedures that handle events and how to add and remove event handlers at run time.

### Lesson objectives

After completing this lesson, you will be able to:

- Describe the event model in the .NET Framework.
- Create and use event handlers.
- Create event procedures.
- Add and remove event handlers from event procedures at run time.

# Event Model in the .NET Framework



```
this.button1.Click += new
System.EventHandler(this.button1_Click);
```

```
private void button1_Click(object
sender, System.EventArgs e)
                {
                …
                }
```

### Introduction

In the .NET Framework, an event is used to signal the occurrence of an action. For example, this action could be user invoked, such as the **Click** event of a **Button** control, or the event could be raised programmatically to signal the end of a long computation.

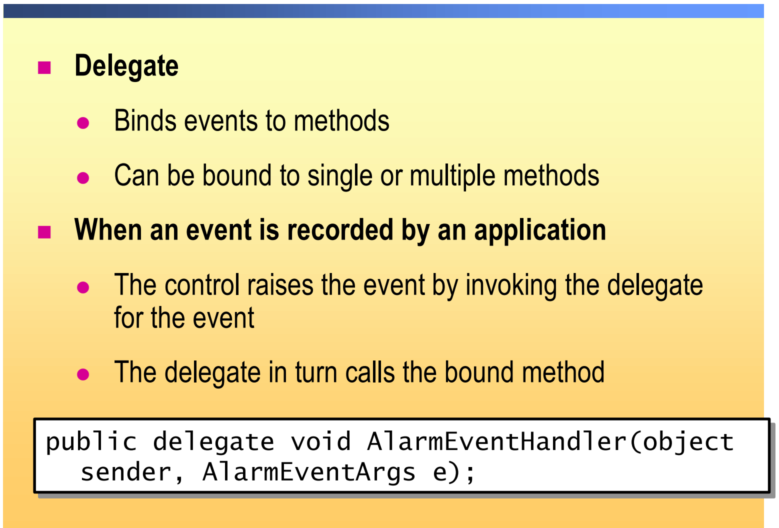
### Event and delegates

The object that raises (triggers) the event is referred to as the *event sender.* The procedure that handles the event is referred to as the *event receiver.* In either case, the sender does not know which object or method will respond to the events that it raises. Therefore, it is necessary to have a component that links the event sender with the event receiver. The .NET Framework uses a delegate type to work as a function pointer between the sender and the event receiver. In most cases, the .NET Framework creates the delegate and takes care of the details for you. However, you can create your own delegates for cases when you want an event to use different event handlers under different circumstances.

### Creating Delegates

Delegates are objects that you can use to call the methods of other objects. You can use the **Delegate** keyword in a declaration statement to create your own delegate that derives from the **MulticastDelegate** class. Creating your own delegates can be useful in situations where you need an intermediary between a calling procedure and the procedure being called. For more information about creating and using delegates, search by using the phrase **Delegate Class** in the Microsoft Visual Studio® .NET Help documentation.

# What Are Delegates?



- **Delegate**
  - Binds events to methods
  - Can be bound to single or multiple methods
- **When an event is recorded by an application**
  - The control raises the event by invoking the delegate for the event
  - The delegate in turn calls the bound method

```
public delegate void AlarmEventHandler(object
  sender, AlarmEventArgs e);
```

**Introduction**

In the .NET Framework, delegates are used to hold a reference to the method that will handle an event. For example, an event occurs when a user clicks a button. The button raises a click event but does not know what behavior you, the programmer, want to occur when the button is clicked, so the button has a delegate member to which you assign your own method for handling the event. You can use the same infrastructure that is used by the .NET Framework to create your own delegates.

**Definition**

A *delegate* is a data structure, derived from the Delegate Class, which refers to a static method or to a class instance and an instance method of that class. Delegates are useful when your application must perform an action by calling a method but you do not know what that action will be.

Delegates allow you to specify at run time the method to be invoked. Delegates are object-oriented, type-safe, and secure.

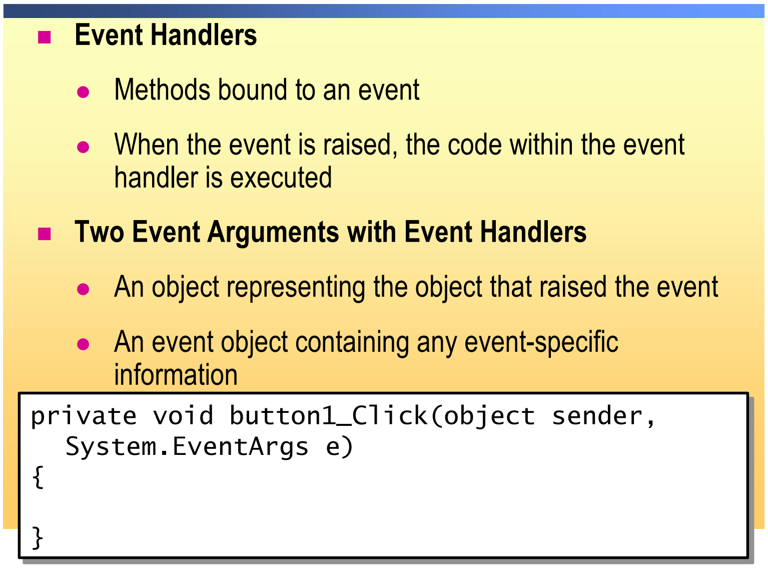
**Event delegate declaration**

By convention, event delegates in the .NET Framework have two parameters, the source that raised the event and the data for the event. The following example shows an event delegate declaration:

```
public delegate void AlarmEventHandler(object sender,
AlarmEventArgs e);
```

Event delegates are multicast, which means that they can hold references to more than one event handling method. Delegates allow for flexibility and fine-grain control in event handling. A delegate acts as an event dispatcher for the class that raises the event by maintaining a list of registered event handlers for the event. For more information on Delegates, see “Delegate Class” in the Visual Studio .NET Help documentation.

# What Is an Event Handler?



- **Event Handlers**
  - Methods bound to an event
  - When the event is raised, the code within the event handler is executed
- **Two Event Arguments with Event Handlers**
  - An object representing the object that raised the event
  - An event object containing any event-specific information

```
private void button1_Click(object sender,
  System.EventArgs e)
{

}
```

**Introduction**

Functionality is added to controls by raising and consuming events. Before your application can respond to an event, you must create an event handler. The event handler (event procedure) contains the program logic that runs when the event is raised.

**Definition**

An event handler is a method that is bound to an event. When the event is raised, the code in the event handler runs. You can use the same event handler to handle more than one event. For example, you can create a single event handler to handle events of a button and a menu item that are used for the same purpose. Similarly, if you have a group of **RadioButton** controls on a form, you could create a single event handler and have each control's **Click** event bound to the single event handler.

**Example of event handler**

The following code example is an event handler for the **Click** event of a button.

```
private void button1_Click(object sender, System.EventArgs e)
{

}
```

The following code example shows how you can use a single event handler to handle events for multiple controls.

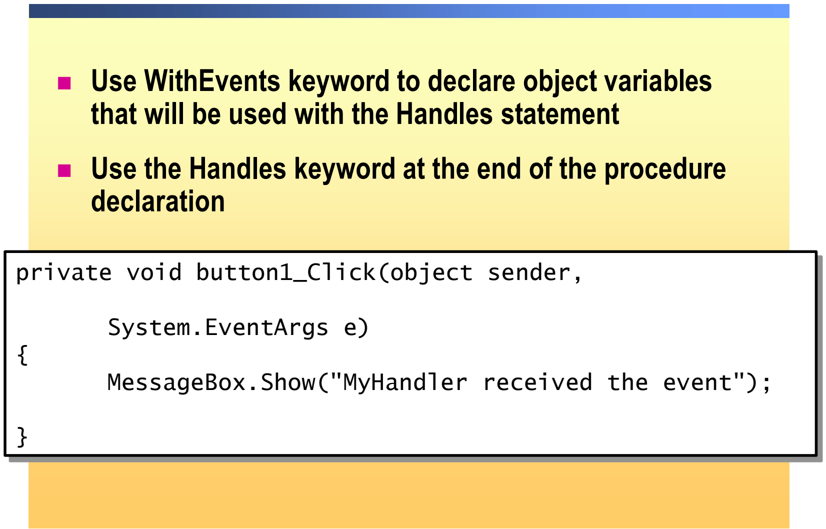
```
// inside the Windows Form Designer generated code region
…
this.button1.Click += new
             System.EventHandler(this.button1_Click);

// add the button2.click event to button1_click handler
this.button2.Click += new
             System.EventHandler(this.button1_Click);

private void button1_Click(object sender, System.EventArgs e)
{

}
```

### Event handler parameters

Each event handler provides two parameters that allow you to handle the event properly.

- The first parameter (*Sender* in the previous code example) provides a reference to the object that raised the event. It specifies the source that raised the event.
- The second parameter (*e* in the previous code example) passes an object specific to the event being handled. This parameter contains all of the data that is required to handle the event.

# How to Create Event Handlers for Control Events



- Use WithEvents keyword to declare object variables that will be used with the Handles statement
- Use the Handles keyword at the end of the procedure declaration

```
private void button1_Click(object sender,

       System.EventArgs e)
{
       MessageBox.Show("MyHandler received the event");

}
```

*****************************ILLEGAL FOR NON-TRAINER USE******************************

**Introduction**

Microsoft Visual C#™ .NET uses the 'System.EventHandler += new …' syntax to define an event handler. The standard way to create an event handler in Visual C# .NET is through the Properties window of the control.

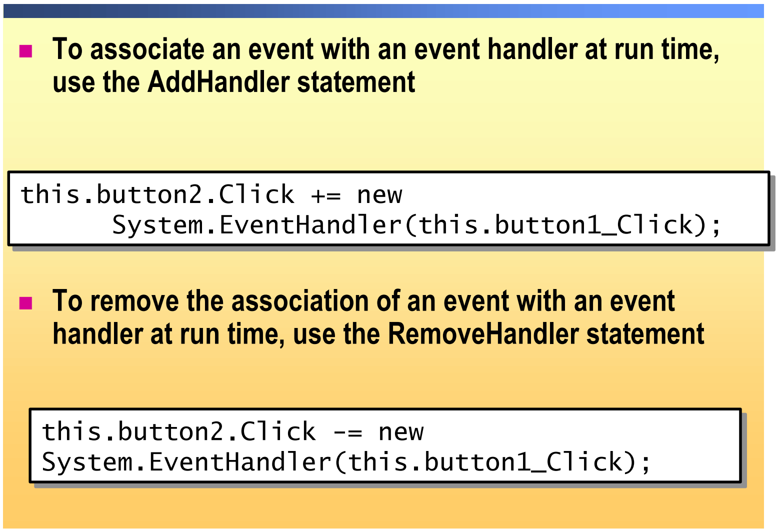
**Procedure**

To create an event procedure for a control:

1. In Design view, click the control that you want to create an event handler for.
2. In the Properties window, click **Event** (the button displaying the lightning bolt).
3. Double-click the event that you want to handle. Code will be added to the Code Editor.
4. Add program logic to the event handler procedure by using the supplied arguments. The following code provides an example:

```
private void button1_Click(object sender,
                           System.EventArgs e)
{
      MessageBox.Show("MyHandler received the event");

}
```

# How to Add and Remove Event Handlers at Run Time



- To associate an event with an event handler at run time, use the AddHandler statement

```
this.button2.Click += new
      System.EventHandler(this.button1_Click);
```

- To remove the association of an event with an event handler at run time, use the RemoveHandler statement

```
this.button2.Click -= new
System.EventHandler(this.button1_Click);
```

### Introduction

In Visual C# .NET, you can add and remove event handlers at run time by using the **System.EventHandler** += **new …** and **System.EventHandler -= new …**syntax.

### Procedure: Adding event handlers

To add event handlers by using **System.EventHandler** += **new …** syntax:

- Use the **System.EventHandler** += **new …** syntax to specify the name of the event sender and receiver, as shown in the following code example:

```
// inside the Windows Form Designer generated code region
…
this.button1.Click += new
      System.EventHandler(this.button1_Click);

// add the button2.click event to button1_click handler
this.button2.Click += new
      System.EventHandler(this.button1_Click);

private void button1_Click(object sender, System.EventArgs
e)
{

}
```

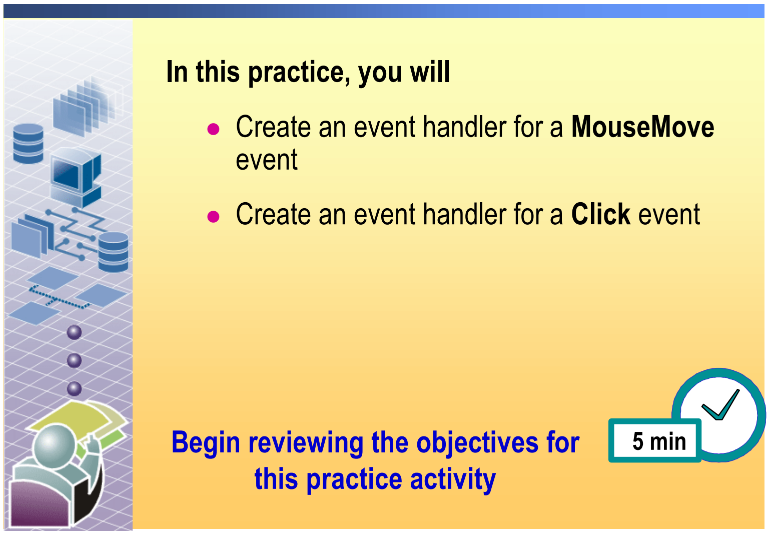
### Procedure: Removing event handlers

To remove event handlers by using **System.EventHandler -= new …**syntax:

- Use the **System.EventHandler -= new …**syntax to specify the name of the event sender and receiver.

```
// remove the button2.click event from button1_click
handler
this.button2.Click -= new
System.EventHandler(this.button1_Click);
```

# Practice: Creating an Event Handler for a Control



**In this practice, you will**

- Create an event handler for a **MouseMove** event
- Create an event handler for a **Click** event

**Begin reviewing the objectives for this practice activity**

**Introduction**

In this practice, you will create an event handler for a control.

**Instructions**

► **Open the practice project**

1. Using Windows Explorer, navigate to *install_folder*\Practices\Mod02\Mod02_01\Starter.

> **Note** If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

2. Double-click the EventHandlers.sln solution file to open the project.

► **Create an event handler for a MouseMove event**

1. Open Form1.cs in Design view, and then click the **VB** button.
2. In the Properties window, click the **Events** button (lightning bolt icon).
3. Scroll down until you see the **MouseMove** event, and then double-click **MouseMove**.
4. Add the following code statements to the **vbButton_MouseMove** event handler:

```
vbButton.Top -= e.Y;
vbButton.Left += e.X;

if (vbButton.Top < -16 || vbButton.Top > 160)
   vbButton.Top = 73;

if (vbButton.Left < -64 || vbButton.Left > 384)
   vbButton.Left = 160;
```

5. What is the purpose of the second parameter (e) that is passed to this event handler?

   **The e parameter contains event data. It is either an EventArgs object (the base class which actually contains no event data) or it is an instance of a derived class like MouseEventArgs. To see a complete list of the derived classes, search by using the phrase EventArgs Class in Visual Studio .NET Help documentation and following the link to 'Derived classes'.**

   ____________________________________________________________

   ____________________________________________________________

   ____________________________________________________________

6. Run the application and click each button on the form.
7. Close the application.

► **Create an event handler for a Click event**

1. Open Form1.cs in Design view.
2. Double-click the **Close** button control.
3. Why is a **Click** event handler created for you?

   **The IDE automatically creates a handler for the default event when you double click a control while in Design view.**

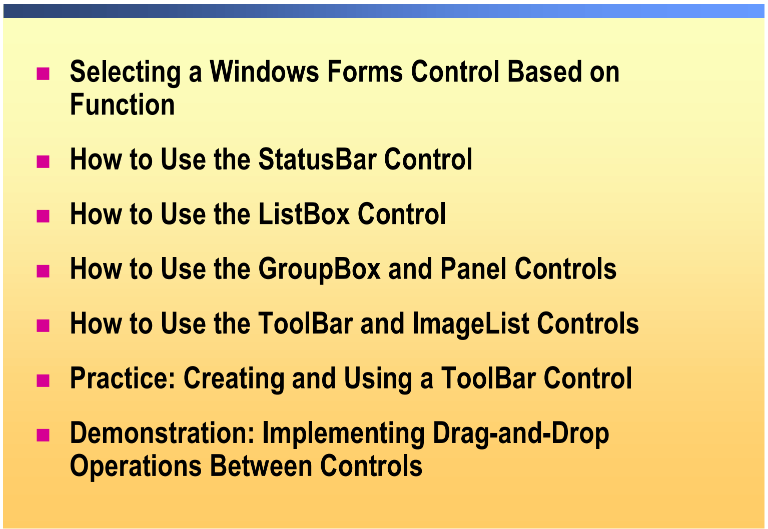
   ____________________________________________________________

   ____________________________________________________________

4. Add the following code statement to the **closeButton_Click** subroutine:

   ```
   this.Close();
   ```
5. Run the application, and then click the **Close** button.
6. Save your project, and then close Visual Studio.

# Lesson: Using Windows Forms Controls



- Selecting a Windows Forms Control Based on Function
- How to Use the StatusBar Control
- How to Use the ListBox Control
- How to Use the GroupBox and Panel Controls
- How to Use the ToolBar and ImageList Controls
- Practice: Creating and Using a ToolBar Control
- Demonstration: Implementing Drag-and-Drop Operations Between Controls

### Introduction

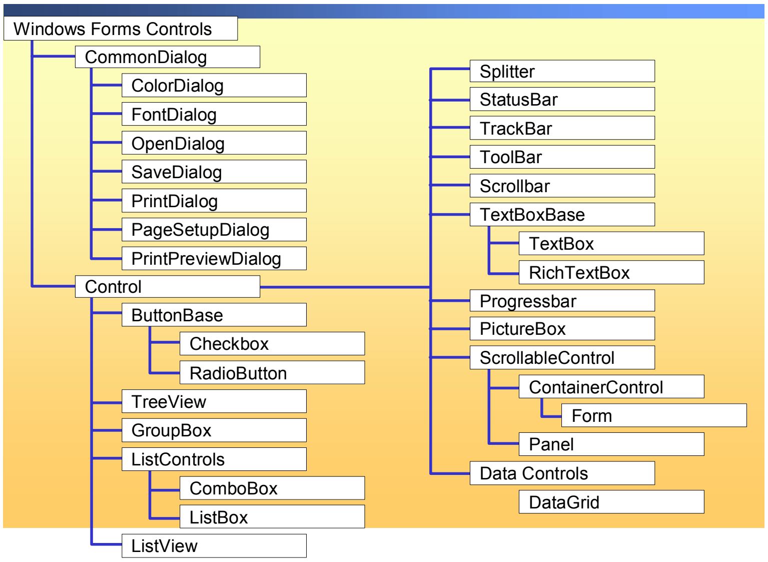
The **Windows Forms** tab in the Visual Studio .NET Toolbox offers a variety of very useful controls, such as the **Button**, **StatusBar**, **ListBox**, **GroupBox**, **ImageList**, **MainMenu**, and **OpenFileDialog** controls. The controls listed in the Toolbox can be categorized based on the way they are used in your applications. In this lesson, you will learn about the different categories of Windows Forms controls and you will then learn how to use a number of controls from each category. This lesson focuses on the controls that are used in the sample applications for Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. Some of the controls will be covered in greater detail in the Using Dialog Boxes in a Windows Forms Application and the Creating Menus lessons in this module.

### Lesson objectives

After completing this lesson, you will be able to:

- Select appropriate Windows Forms controls for adding the desired functionality to a user interface.
- Use the **StatusBar** control in a Windows Forms application to display text information.
- Use the **ListBox** control in a Windows Forms application to provide the user with a predefined list of items.
- Use **GroupBox** and **Panel** controls in a Windows Forms application as containers for other controls.
- Use the **ToolBar** and **ImageList** controls in a Windows Forms application to display command buttons as an array of graphics images.
- Implement drag-and-drop operations between controls.

# Selecting a Windows Forms Control Based on Function



### Introduction

Windows Forms controls are reusable components that encapsulate user interface functionality and are used in a Windows Forms application. Not only does the .NET Framework Class Library provide many ready-to-use controls, it also provides the infrastructure for developing your own controls.

The controls that appear on the Windows Forms tab of the Toolbox can be categorized based on their functions. Depending on the functionality that you want to provide in the user interface of your application, you will select a control from one of the following categories: Commands, Text, Options, Containers, Graphics, Menus, or Dialog Boxes. Notice that the class hierarchy and the functional category for a control do not always match one another.

### Commands category controls

The following are the categories of command controls:

- **Button**

  Used to start, stop, or interrupt a process.

- **LinkLabel**

  Displays text as a Web-style link and triggers an event when the user clicks the special text. Usually the text is a link to another window or a Web site.

- **NotifyIcon**

  Displays an icon in the status notification area of the taskbar that represents an application running in the background.

- **ToolBar**

  Contains a collection of button controls.

### Text category controls

The following text controls are used to enable users to enter text and edit the text contained in these controls at run time:

- **Textbox**

  Displays text entered at design time that can be edited by users at run time, or changed programmatically.

- **RichTextBox**

  Enables text to be displayed with formatting in plain text or rich-text format (RTF).

The following additional text controls can be used to display text but do not allow application users to directly edit the text content that they display:

- **Label**

  Displays text that users cannot directly edit.

- **StatusBar**

  Displays information about the application's current state by using a framed window. A status bar is usually located at the bottom of a parent form.

### Options category controls

The following selection controls allow users to select a value from a list:

- **CheckedListBox**

  Displays a scrollable list of items, each accompanied by a check box.

- **ComboBox**

  Displays a drop-down list of items.

- **DomainUpDown**

  Displays a list of text items that users can scroll through by using up and down buttons.

- **ListBox**

  Displays a list of text and graphical items (icons).

- **ListView**

  Displays items in one of four different views. Views include text only, text with small icons, text with large icons, and a report view.

- **NumericUpDown**

  Displays a list of numerals that users can scroll through by using up and down buttons.

- **TreeView**

  Displays a hierarchical collection of node objects, which can consist of text with optional check boxes or icons.

### Containers category controls

Container controls can be used to group other controls on a form. Some examples of container controls are:

- **Panel**

  Groups a set of controls on an unlabeled, scrollable frame.

- **GroupBox**

  Groups a set of controls (such as radio buttons) on a labeled, nonscrollable frame.

- **TabControl**

  Provides a tabbed page for organizing and accessing grouped objects efficiently.

### Graphics category controls

The following are the categories of graphic controls:

- **ImageList**

  Serves as a repository for images. **ImageList** controls and the images they contain can be reused from one application to the next.

- **PictureBox**

  Displays graphical files, such as bitmaps and icons, in a frame.

### Dialog boxes category controls

Visual Studio .NET provides a set of common dialog boxes. These include **ColorDialog**, **FontDialog**, **PageSetupDialog**, **PrintDialog**, **OpenFileDialog**, and so on. You will learn more about dialog boxes in the lesson Using Dialog Boxes in a Windows Forms Application in this module.

### Menus category controls

The following are the categories of menu controls:

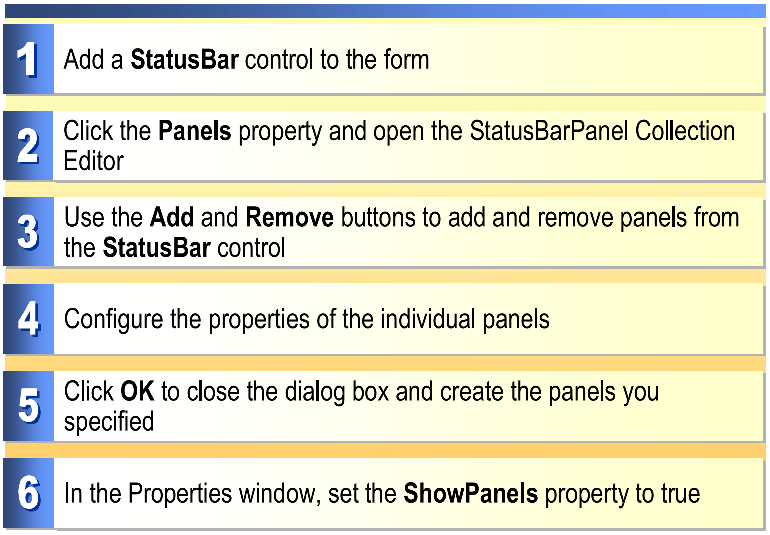
- **MainMenu**

  Provides a design-time interface for creating menus.

- **ContextMenu**

  Implements a menu that appears when the user right-clicks an object.

# How to Use the StatusBar Control



1 Add a **StatusBar** control to the form

2 Click the **Panels** property and open the StatusBarPanel Collection Editor

3 Use the **Add** and **Remove** buttons to add and remove panels from the **StatusBar** control

4 Configure the properties of the individual panels

5 Click **OK** to close the dialog box and create the panels you specified

6 In the Properties window, set the **ShowPanels** property to true

### Introduction

The **StatusBar** control is an interesting example of a control taken from the text category of controls. A status bar is a horizontal window at the bottom of a parent window in which an application can display various kinds of status information. The status bar can be divided into parts to display more than one type of information. **StatusBar** controls can have status bar panels on them that display text or icons to indicate state, or a series of icons in an animation that indicate a process is working, such as the status bar in Microsoft Word indicating that the document is being saved.

### Procedure: Using the StatusBar control

The .NET Framework offers the **StatusBar** control for the status bar. You can create panels in the status bar by using the **Add** method of the **Panels** collection. To display the panels, you must set the **ShowPanels** property to **True**. You can indicate the size and alignment of each panel by setting additional properties.

To create a status bar with panels:

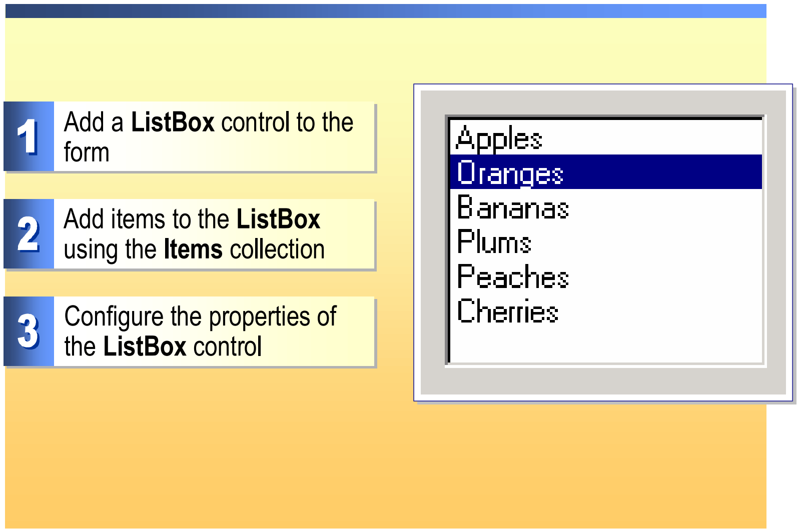
1. Add a **StatusBar** control to the form.
2. In the Properties window, click the **Panels** property to select it. Then click the **ellipsis** (…) button to open the StatusBarPanel Collection Editor.
3. Use the **Add** and **Remove** buttons to add and remove panels from the **StatusBar** control at design time. You can use the **Add** and **Remove** methods of the **StatusBarPanels** object to add and remove panels at run time.

4. Configure the properties of the individual panels in the Properties window. The following table lists the important properties and their descriptions.

| Property | Description |
|---|---|
| **AutoSize** | Determines the resize behavior of the panel. |
| **Alignment** | Sets the alignment of the panel in the **StatusBar** control. |
| **BorderStyle** | The type of border displayed at the edges of the panel. |
| **Icon** | The icon (.ico file) displayed in the panel. |
| **Style** | Sets the style of the panel. It must be one of the values of the **StatusBarPanelStyle** enumeration. |
| **Text** | The text string displayed in the panel. |
| **MinWidth** | The minimum width of the panel in the status bar. |

5. Click **OK** to close the dialog box and create the panels you specified.
6. In the Properties window, set the **ShowPanels** property to **True**.

# How to Use the ListBox Control



1. Add a **ListBox** control to the form
2. Add items to the **ListBox** using the **Items** collection
3. Configure the properties of the **ListBox** control

****************************ILLEGAL FOR NON-TRAINER USE******************************

## Introduction

The **ListBox** control is a good example of a control from the options category of controls. A Windows Forms **ListBox** control displays a list of items from which the user can select one or more. If the total number of items exceeds the number that can be displayed, a scroll bar is automatically added to the **ListBox** control.

## ListBox properties

The following table lists the important properties of the **ListBox** control.

| Property | Description |
|---|---|
| MultiColumn | When set to **True**, the list box displays items in multiple columns and a horizontal scroll bar appears. When set to **False**, the list box displays items in a single column and a vertical scroll bar appears. |
| ScrollAlwaysVisible | When set to **True**, the scroll bar appears regardless of the number of items. |
| SelectionMode | Determines how many list items can be selected at a time. |
| SelectedIndex | Returns an integer value that corresponds to the first selected item in the list box. If no item is selected, the **SelectedIndex** value is **-1**. If the first item in the list is selected, then the **SelectedIndex** value is **0**. |
| Items.Count | Reflects the number of items in the list. |
| Items.Add/Items.Insert | Adds items to the **ListBox** control. |
| Items.Remove/Items.Clear | Removes items from the **ListBox** control. |
| DataSource | Binds the **ListBox** to a data source. |
| DisplayMember | Binds the **ListBox** to a name of the column in the Data source. |

### Procedure: Using the ListBox control

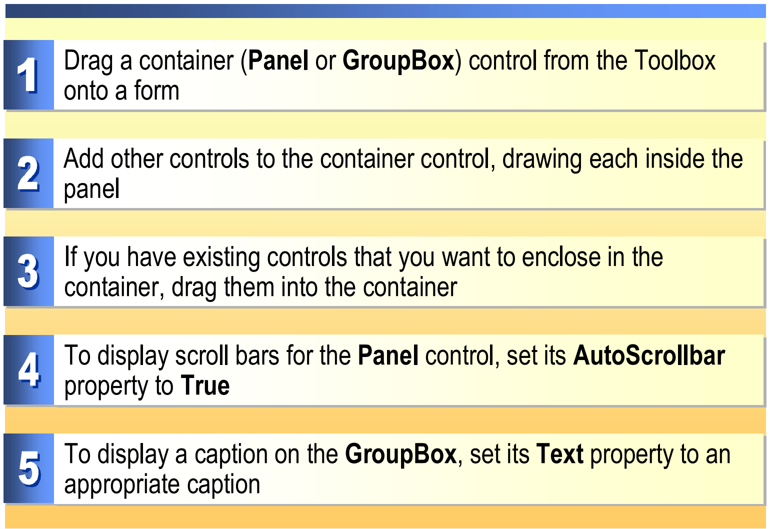
To use a **ListBox** control:

1. Add a **ListBox** control to the form.
2. Add items to the **ListBox** by using the **Items** collection.

   You can add multiple items to the **ListBox** at the same time by using the **AddRange** method.

   ```
   ListBox1.Items.Addrange(NewObject() {“Apples”, “Oranges”,
   “Bananas”});
   ```
3. Configure the properties of the **ListBox** control.

# How to Use the GroupBox and Panel Controls



1 Drag a container (**Panel** or **GroupBox**) control from the Toolbox onto a form

2 Add other controls to the container control, drawing each inside the panel

3 If you have existing controls that you want to enclose in the container, drag them into the container

4 To display scroll bars for the **Panel** control, set its **AutoScrollbar** property to **True**

5 To display a caption on the **GroupBox**, set its **Text** property to an appropriate caption

**Introduction**

When you want the user to select one or more options from a group of available options, you typically use check boxes (more than one selection) or radio buttons (single selection) grouped inside a container control. All controls in a container control function as a group. The following are three main reasons to group controls:

- Visual grouping of related form elements for a clear user interface
- Moving the controls as a unit at design time
- Programmatic grouping of controls

Visual Studio .NET includes container controls such as **GroupBox** and **Panel** that allow you to group radio buttons, check boxes, or any other controls that you want to treat as part of a control collection. The **Panel** control is similar to the **GroupBox** control; however, the **Panel** control can have scroll bars, and only the **GroupBox** control displays a caption.

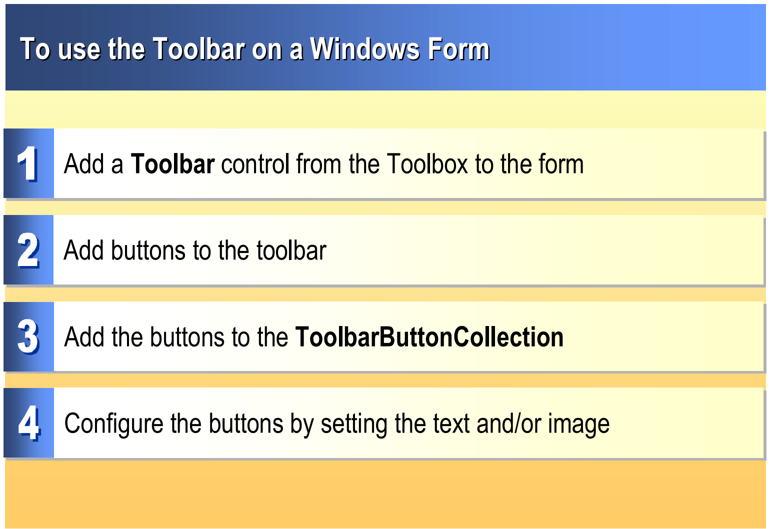
**Procedure: Creating and populating container controls**

To create and populate a container control:

1. Drag a container (**Panel** or **GroupBox**) control from the Windows Forms tab of the Toolbox onto a form.
2. Add other controls to the container control, drawing each inside the panel.
3. If you have existing controls that you want to enclose in the container, drag them into the container.
4. To display scroll bars for the **Panel** control, set its **AutoScrollbar** property to **True**.
5. To display a caption on the **GroupBox**, set its **Text** property to an appropriate caption.

The controls grouped in a container control can be accessed by using the **Controls** property. Each control that is grouped inside a **Panel** or a **GroupBox** is a member of the **Control.ControlCollection** object, which is assigned to the **Control** property of the container. You will learn more about the ControlCollection in the lesson, Adding Controls at Run Time, in this module.

# How to Use the ToolBar and ImageList Controls



To use the Toolbar on a Windows Form

1. Add a **Toolbar** control from the Toolbox to the form
2. Add buttons to the toolbar
3. Add the buttons to the **ToolbarButtonCollection**
4. Configure the buttons by setting the text and/or image

### Introduction

The **ToolBar** control is a good example of a control from the commands category of controls. Toolbars are an alternative graphical user interface (GUI) element to menus. A toolbar contains a set of buttons, which are represented by the **ToolBarButton** class in the .NET Framework.

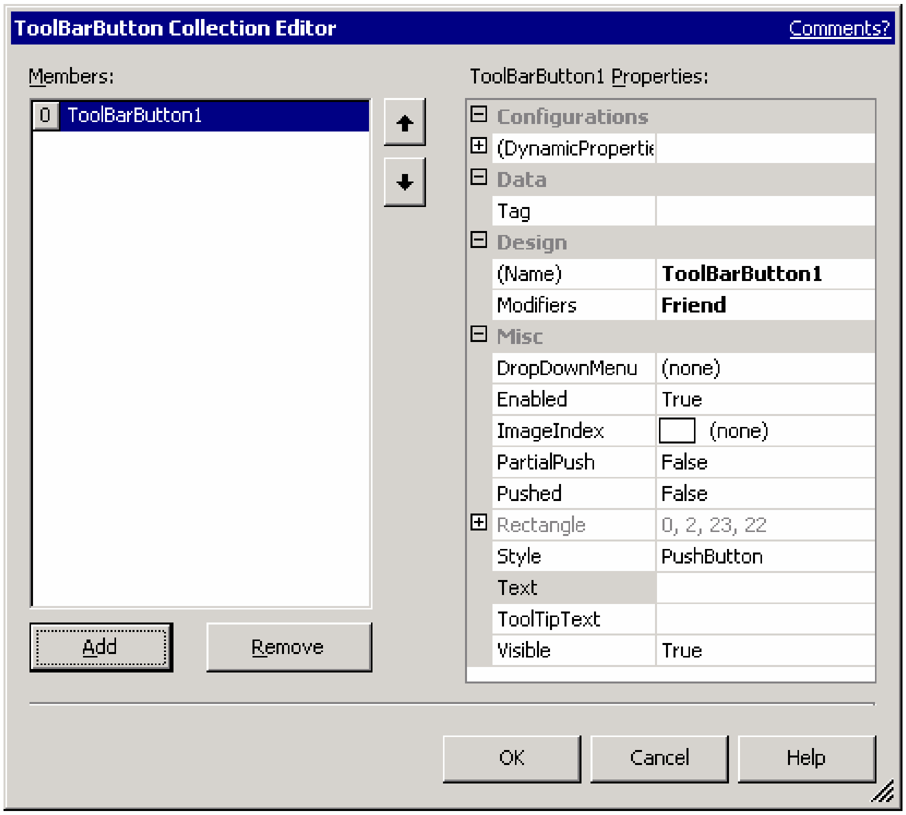
### Procedure: Using the ToolBar

To use a **ToolBar** in a Windows Forms application:

1. Add a **Toolbar** from the Toolbox to the form.
2. Add buttons to the **Toolbar**.

   You can use the **Buttons** property to add buttons to the **Toolbar**.

3. Add the buttons to the **ToolbarButtonCollection** by using the **Buttons** property.



| | |
|---|---|
| ⊟ Configurations | |
| ⊞ (DynamicProperti | |
| ⊟ Data | |
| Tag | |
| ⊟ Design | |
| (Name) | **ToolBarButton1** |
| Modifiers | **Friend** |
| ⊟ Misc | |
| DropDownMenu | (none) |
| Enabled | True |
| ImageIndex | □ (none) |
| PartialPush | False |
| Pushed | False |
| ⊞ Rectangle | 0, 2, 23, 22 |
| Style | PushButton |
| Text | |
| ToolTipText | |
| Visible | True |

4. Configure the buttons by setting the text, image, and so on.

   Each button on the **Toolbar** can have its own look. The caption and the image are optional. Similar to the **ListView** and **TreeView** control, toolbar images are stored in an image list. The **ToolBarButton** class has an **Index** property that refers to the image in this list.

   Toolbar buttons can appear in different types. The **Style** property can be set to one of the **ToolBarButtonStyle** values shown in the following table.

| Value | Description |
|---|---|
| DropDownButton | A drop-down control that displays a menu or other window when clicked. |
| PushButton | A standard, three-dimensional button. |
| Separator | A space or line between toolbar buttons. The appearance depends on the value of the **Appearance** property. |
| ToggleButton | A toggle button that appears sunken when clicked and retains the sunken appearance until clicked again. |

**Note** To display a tooltip for a button, set the **ShowToolTips** property to **True**. You can define the content of a tooltip by setting the **ToolTipText** property of the button.

### Procedure: Using the ImageList

The Windows Forms **ImageList** control is used to store images, which can then be displayed by controls, and is a good example of a control from the graphics category of controls. For example, you can enable the button to display different images by changing the **ImageIndex** property. You can also associate the same image list with multiple controls. You can use an image list with any control that has an **ImageList** property—or, in the case of the **ListView** control, the **SmallImageList** and **LargeImageList** properties. The controls that can be associated with an image list include the **ListView**, **TreeView**, **ToolBar**, **TabControl**, **Button**, **CheckBox**, **RadioButton**, and **Label** controls.

To associate the image list with a control, set the control's **ImageList** property to the name of the **ImageList** control. The key property of the **ImageList** control is **Images**, which contains the pictures to be used by the associated control. Each individual image can be accessed by its index value. The **ColorDepth** property determines the number of colors that the images are rendered with. The images are all displayed at the same size; this is determined by the **ImageSize** property. Images that are larger will be scaled to fit.

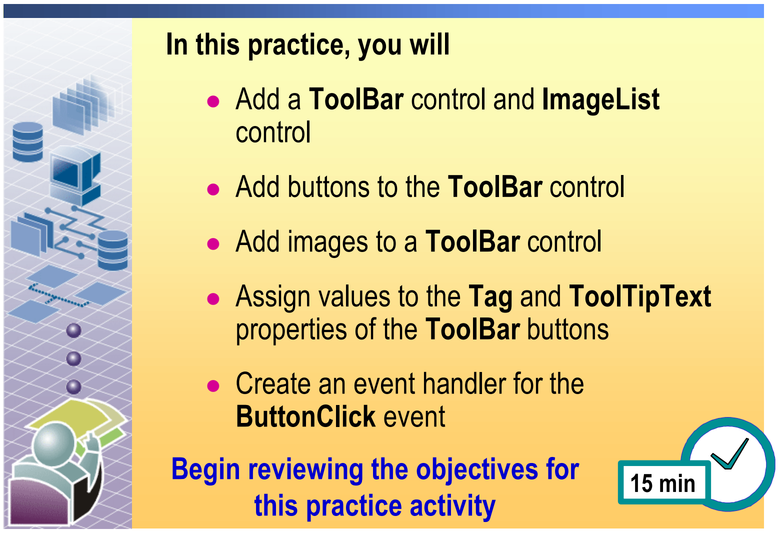
### Procedure: Triggering events for the toolbar buttons

If your Windows Forms application features a **ToolBar** control with buttons, you will want to know which button the user clicks. To determine which button is clicked, add an event handler to the **ButtonClick** event of **ToolBar** control. Use a **Select Case** statement and the **ToolBarButtonClickEventArgs** class to determine the toolbar button that is clicked. The following example shows how to use the **Button** property of the **ToolBarButtonClickEventArgs** object to determine which button is clicked.

**Note** The following code example uses the **Tag** property to determine the control that is clicked, but you can also do this by using the index value of a control. However, using the index value of controls makes it difficult to keep track of controls and their corresponding index values. For example, if you have a separator in your form, the separator will also use an index value, and you need to take the separator into account when referencing the index value.

```
private void ToolBar1_ButtonClick(object sender,
System.Windows.Forms.ToolBarButtonClickEventArgs e)
{
  switch(e.Button.Tag.ToString())
  {
      case "Cut":
          this.cutRadioButton.Checked = true;
          panelText = this.cutRadioButton.Text +
              " Radio Button is checked";
          break;
      case "Copy":
          this.copyRadioButton.Checked = true;
          panelText = this.copyRadioButton.Text +
              " Radio Button is checked";
          break;
  }

  MessageBox.Show("The " + e.Button.Tag
      + " button is index number "
      + ToolBar1.Buttons.IndexOf(e.Button));

}
```

# Practice: Creating and Using a ToolBar Control



In this practice, you will

- Add a **ToolBar** control and **ImageList** control
- Add buttons to the **ToolBar** control
- Add images to a **ToolBar** control
- Assign values to the **Tag** and **ToolTipText** properties of the **ToolBar** buttons
- Create an event handler for the **ButtonClick** event

Begin reviewing the objectives for this practice activity

15 min

**Introduction**

In this practice, you will create and use a **ToolBar** control.

**Instructions**

### ► Open the practice project

1. Using Windows Explorer, navigate to *install_folder*\Practices\Mod02\Mod02_02\Starter.
2. Double-click the ToolBar.sln solution file to open the project.

### ► Add a ToolBar control and ImageList control to your project

1. Open Form1.cs in Design view.
2. In the Toolbox, double-click **ImageList**.
3. In the Toolbox, double-click **ToolBar**.

### ► Add buttons to the ToolBar control

1. On the form, click **toolBar1**.
2. In the **toolBar1** Properties window, click **Buttons**.
3. In the value column for the **Buttons** property, click the **ellipsis** (…) button.
4. On the ToolBarButton Collection Editor window, click **Add**.
5. Use the **Add** button to add six more buttons to the **ToolBar** control.
6. Click **toolBarButton5**, and then, in the properties table, click **Style**.
7. In the **Style** property value list, click **Separator**.

8. Why can't you add images to the **ToolBar** buttons yet?

   **A ToolBar control uses an ImageList control as a source of images. You specify which ImageList control the Toolbar will get images from by using the ImageList property of the ToolBar control.**

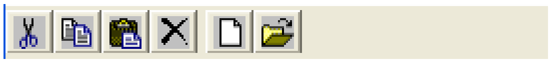
   ________________________________________________________________

   ________________________________________________________________

9. Click **OK**.

► **Add images to a ToolBar control by using an ImageList control**

1. In the **toolBar1** Properties window, scroll down until you see the **ImageList** property, and then click **ImageList**.
2. Open the list for the **ImageList** property, and then click **imageList1**.
3. In the component tray at the bottom of Design view, click **imageList1**.
4. In the **imageList1** Properties window, click **Images**.
5. In the value column for the **Images** property, click the **ellipsis** (…) button.
6. In the **Image Collection Editor** dialog box, click **Add**.
7. In the **Look in** list, navigate to *install_folder*\Practices\Mod02\Mod02_02\Starter\bin, click **CUT.BMP**, and then click **Open**.
8. Repeat steps 6 and 7 to add the COPY.BMP, PASTE.BMP, DELETE.BMP, NEW.BMP, and OPEN.BMP images to **imageList1**, and then click **OK**.
9. Open the ToolBarButton Collection Editor, and then click **toolBarButton1**.
10. In the ToolBarButton1 Properties window, click **ImageIndex**, and then, in the **ImageIndex property** value list, click **0** (image index 0 is the scissors icon).
11. Use the properties table for ToolBar buttons 2-4 and 6-7 to assign a value to the **ImageIndex** property. When you have finished adding images to **toolBar1**, it should appear as follows:



12. Click **OK**, and then save the changes to your application.
13. Run the application. Does anything happen when you click a ToolBar button? What event must be handled to respond to **ToolBar** button clicks?

    **The ButtonClick event of the ToolBar is used to handle ToolBar button clicks. ToolBarButtonClickEventArgs is used to determine which button was clicked.**

    ________________________________________________________________

    ________________________________________________________________

14. Close the application.

### ► Assign values to the Tag and ToolTipText properties of the ToolBar buttons

1. Open the ToolBarButton Collection Editor, and then click **toolBarButton1**.
2. In the **toolBarButton1** Properties table, double-click **Tag**, type **Cut** and then press ENTER.
3. Use the properties table for **toolBarButton2**, **toolBarButton3**, **toolBarButton4**, **toolBarButton6**, and **toolBarButton7** to assign **Tag** property values to the buttons in the following order: **Copy**, **Paste**, **Delete**, **New**, and **Open**.
4. In the toolBarButton1 Properties table, double-click **ToolTipText**, type **Cut item** and then press ENTER.
5. Optional: Use the Properties table for **toolBarButton2**, **toolBarButton3**, **toolBarButton4**, **toolBarButton6**, and **toolBarButton7** to assign **ToolTipText** property values to the buttons in the following order: **Copy item**, **Paste item**, **Delete an existing item**, **Create a New item**, and **Open an existing item**.
6. Click **OK**, and then save the changes to your application.

### ► Create an event handler for the ButtonClick event

1. Open Form1.cs in Design view, and then click **toolBar1**.
2. In the **toolBar1** Properties window, click **Events**, and then double-click **ButtonClick**.
3. Add the following code statements to the **toolBar1_ButtonClick** procedure:

```
switch(e.Button.Tag.ToString())
{
   case "Cut";
       this.cutRadioButton.Checked = true;
       panelText = this.cutRadioButton.Text +
           " Radio Button is checked";
       break;
}

this.StatusBar1.Panels[0].Text = panelText;

MessageBox.Show("The " + e.Button.Tag
   + " button is index number "
   + toolBar1.Buttons.IndexOf(e.Button));
```

4. On the **View** menu, point to **Show Tasks**, and then click **Comment**.
5. In the Task List, double-click **TODO: paste inside switch statement**.
6. Use a cut-and-paste operation to move the commented code inside the switch statement structure that you just created, and then uncomment the code statements.

7. What are some of the reasons for using the **Tag** property to determine which **ToolBar** button was clicked?

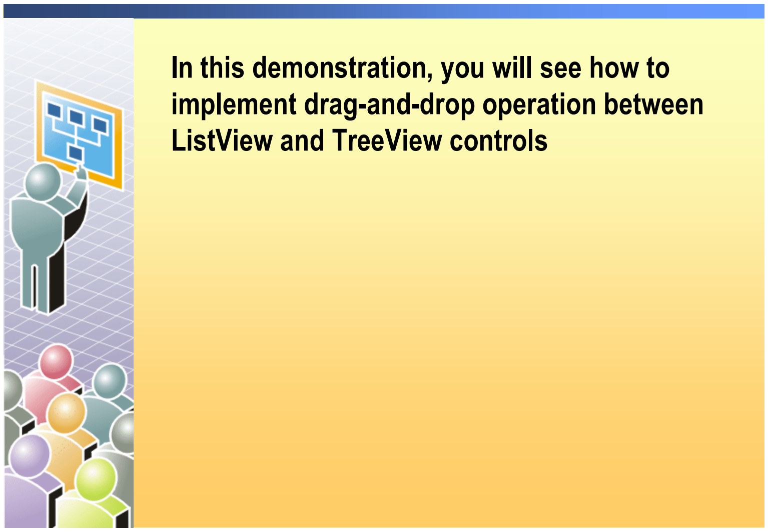
   **You do not need to update code when the button order changes and you do not need to consider button separators.**

   ____________________________________________________________

   ____________________________________________________________

8. Run the application. Test the ToolTips for the **ToolBar** buttons by positioning the mouse pointer over a button. Test the **ButtonClick** event handler by clicking the **ToolBar** buttons.
9. If time permits, examine the code used to construct the **StatusBar** control.
10. Save your project and then close Visual Studio.

# Demonstration: Implementing Drag-and-Drop Operations Between Controls



In this demonstration, you will see how to implement drag-and-drop operation between ListView and TreeView controls

### Introduction

Another way in which you can enhance the usefulness of the controls you add to your application is to enable drag-and-drop operations. The concept of drag-and-drop functionality is familiar to users, and in some cases, it is even expected. The manner in which you implement drag-and-drop functionality depends on the controls being used. The following demonstration walks you through the code that must be added to an application to use drag-and-drop operations to move data between two **TextBox** controls, and between two **TreeView** controls.

### Instructions

► **Open the SimpleDragDrop solution file**

- Open the SimpleDragDrop.sln solution in Visual Studio .NET from *install_folder*\Democode\ Mod02\Mod02_04\SimpleDragDrop.

► **Run the application and demonstrate a drag-and-drop operation between the TextBox controls**

1. Run the application.
2. Position the mouse over the **TextBox** control on the left side of the form.
3. Use a left-click and drag operation to drag the contents of the left **TextBox** control and position it over the upper-right TextBox.
4. Notice that the mouse pointer changes to indicate when a control has been configured to accept dropped data. This control will not accept dropped data.
5. With the mouse positioned over the upper-right TextBox, release the left mouse button.
6. Use a left-click and drag operation to drag the contents of the left **TextBox** control to a position over the lower-right **TextBox** control.

7. Notice that the mouse pointer changes to indicate that this control will accept dropped data.
8. With the mouse positioned over the lower-right **TextBox** control, release the left mouse button.
9. Close the application.

► **Walk through the code that is used to support drag-and-drop operations between two TextBox controls**

1. Open Form1.cs in Design view.
2. Click the upper-right **TextBox** control. Notice that the **AllowDrop** property is set to **False**.
3. Click the lower-right **TextBox** control. Notice that the **AllowDrop** property is set to **True**.
4. Open Form1.cs in the Code Editor.
5. Examine the TextBox1_MouseDown handler that is used to initiate this drag-and-drop operation.

   Notice that the **DoDragDrop** method is used to specify the data that will be used in the drag-and-drop operation. The **DoDragDrop** method is also used to specify the type, or types, of operations allowed. The information associated with this operation is stored in the event arguments variable.
6. Examine the TextBox2_DragEnter handler.

   Notice that the data type of the data being dragged is checked to ensure that the data can be used as intended. The *DragEventArgs* parameter is used to access the data, and, if the data is of an inappropriate type, the **DragEventArgs.Effect** property is set to **DragDropEffects.None**.
7. Examine the TextBox2_DragDrop handler.

   Notice that data contained in the **GetData** method of the **DragEventArgs.Data** property is used to extract the data that will be placed in the control.

► **Run the application and demonstrate a drag-and-drop operation between TreeView controls**

1. Run the application.
2. Position the mouse over the Mustard node displayed on the left **TreeView** control.
3. Use a left-click and drag operation to drag the Mustard node to a position over the Garnishes node of the right **TreeView** control.
4. With the mouse positioned over the Garnishes node, release the left mouse button.
5. Close the application.

► **Walk through the code that is used to support drag-and-drop operations between two TreeView controls**

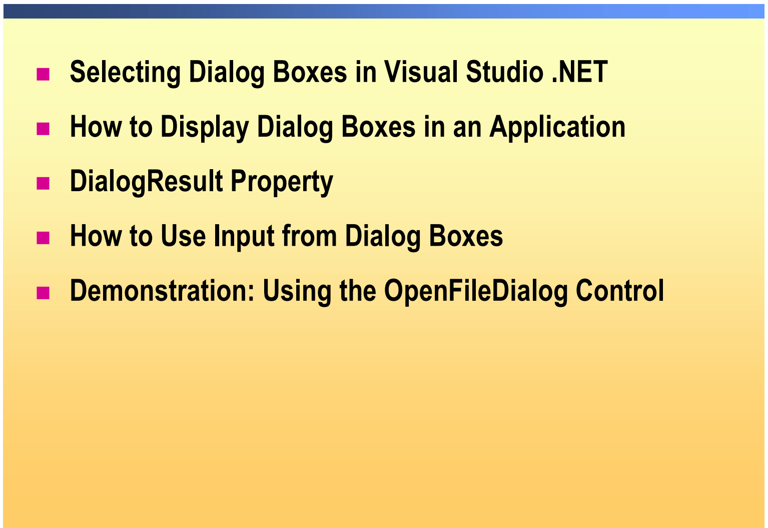
1. Open Form1.cs in the Code Editor.
2. Examine the **TreeView_ItemDrag** handler that is used to initiate this drag-and-drop operation.

   Notice the **TreeView** control includes a special event designed for initiating a drag-and-drop operation. The **DoDragDrop** method is used again to specify the data (in this case the currently selected item) that will be used in the drag-and-drop operation if the type of operation is allowed.
3. Examine the **TreeView_DragEnter** handler.

   Notice that the **DragEnter** event handler is used in the same way it was used when performing a drag-and-drop operation between **TextBox** controls.
4. Examine the **TreeView_DragDrop** handler.

   Notice how **TreeNodes** are handled in this procedure.
5. Close Visual Studio .NET.

# Lesson: Using Dialog Boxes in a Windows Forms Application



- Selecting Dialog Boxes in Visual Studio .NET
- How to Display Dialog Boxes in an Application
- DialogResult Property
- How to Use Input from Dialog Boxes
- Demonstration: Using the OpenFileDialog Control

**Introduction**

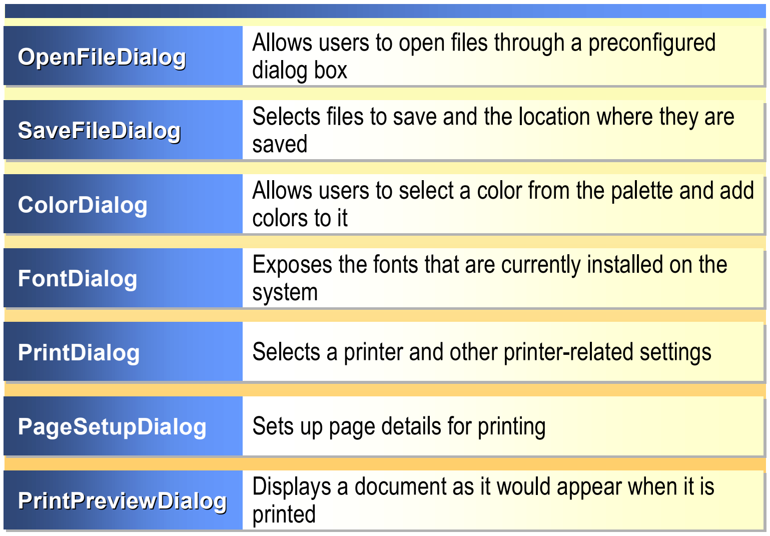
Dialog boxes are used to interact with the user and retrieve user input. Visual Studio .NET provides some preconfigured dialog boxes that can be used in Windows Forms applications to interact with users. This lesson introduces the standard dialog boxes provided by Visual Studio .NET and also covers how to use these dialog boxes to retrieve user input.

**Lesson objectives**

After completing this lesson, you will be able to:

- Select appropriate Visual Studio .NET dialog boxes for a Windows Forms application.
- Use the dialog boxes available in Visual Studio .NET in a Windows Forms application.
- Retrieve user input by using the **DialogResult** property.

# Selecting Dialog Boxes in Visual Studio .NET



| | |
|---|---|
| OpenFileDialog | Allows users to open files through a preconfigured dialog box |
| SaveFileDialog | Selects files to save and the location where they are saved |
| ColorDialog | Allows users to select a color from the palette and add colors to it |
| FontDialog | Exposes the fonts that are currently installed on the system |
| PrintDialog | Selects a printer and other printer-related settings |
| PageSetupDialog | Sets up page details for printing |
| PrintPreviewDialog | Displays a document as it would appear when it is printed |

**Introduction**

Visual Studio .NET includes a set of preconfigured dialogs boxes, which you can adapt for your own applications. Depending on the requirement of the application, you can select an appropriate dialog box from any of the preconfigured dialog boxes.

**OpenFileDialog**

Use the **OpenFileDialog** control in a Windows Forms application as a simple solution for file selection instead of configuring your own dialog box. When using the **OpenFileDialog** control, you must write your own file-opening logic. **OpenFileDialog** is the same **File Open** dialog box used in the Microsoft Windows® operating system. When it is added to a form, the **OpenFileDialog** control appears in the tray at the bottom of the Windows Forms Designer. It inherits from the **CommonDialog** class.

**SaveFileDialog**

The **SaveFileDialog** control enables users to save files in an application. As with the other dialog boxes, when using the **SaveFileDialog** control, you must write your own file-saving logic. **SaveFileDialog** is the same as the standard **Save File** dialog box used by Windows. It inherits from the **CommonDialog** class.

**ColorDialog**

The Windows Forms **ColorDialog** control is a preconfigured dialog box that allows the user to select a color from a palette and to add custom colors to that palette. It is the same dialog box that you see in other Windows applications to select colors.

The color selected in the dialog box is returned in the **Color** property. If the **AllowFullOpen** property is set to **False**, the **Define Custom Colors** button is disabled and the user is restricted to the predefined colors in the palette. If the **SolidColorOnly** property is set to **True**, the user cannot select dithered colors.

**FontDialog**

The Windows Forms **FontDialog** control is the standard Windows **Font** dialog box used to expose the fonts that are currently installed on the system. By default, the dialog box includes options for Font, Font style, and Size. It also includes check boxes for effects like Strikeout and Underline, and a drop-down list for Script.

**PrintDialog**

The **PrintDialog** control is used to select a printer, choose the pages to print, and determine other print-related settings in Windows applications. You can provide users with options such as print all, print a specified page range, or print a selection.

**PageSetupDialog**

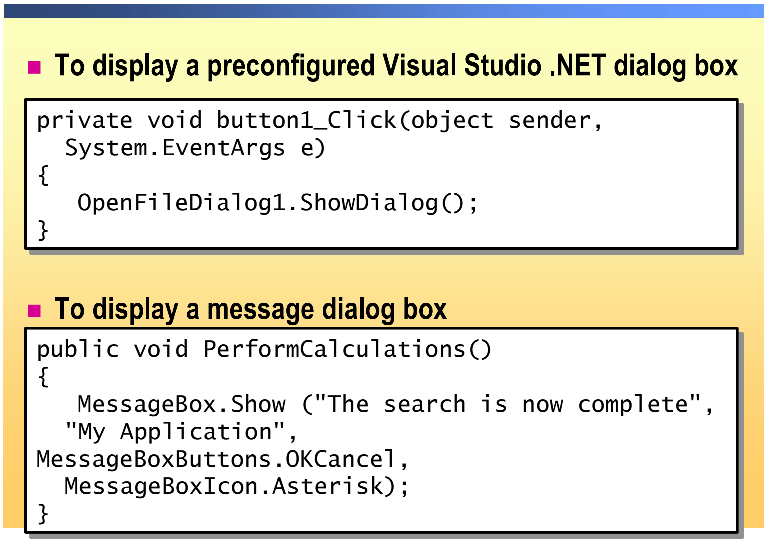
The **PageSetupDialog** control is used to set page details for printing in Windows applications. You can enable users to set border and margin adjustments, headers and footers, and portrait or landscape orientation.

The **PageSetupDialog** allows users to set properties that relate to either a single page (**PrintDocument** class) or any document (**PageSettings** class). Additionally, the **PageSetupDialog** control can be used to determine specific printer settings, which are stored in the **PrinterSettings** class.

**PrintPreviewDialog**

The **PrintPreviewDialog** control is used to display how a document will appear when printed. The control contains buttons for printing, zooming in, displaying one or multiple pages, and closing the dialog box.

# How to Display Dialog Boxes in an Application



- To display a preconfigured Visual Studio .NET dialog box

```
private void button1_Click(object sender,
  System.EventArgs e)
{
   OpenFileDialog1.ShowDialog();
}
```

- To display a message dialog box

```
public void PerformCalculations()
{
   MessageBox.Show ("The search is now complete",
  "My Application",
MessageBoxButtons.OKCancel,
  MessageBoxIcon.Asterisk);
}
```

### Introduction

You display dialog boxes in an application the same way you display a form. To display any of the Visual Studio .NET dialog boxes in an application, you need to write the code to load and display it, just the way you would display a second form in the main form.

### Procedure: Displaying a preconfigured dialog box

To display a preconfigured dialog box:

1. In the Code Editor, navigate to the event handler with which you want to open the dialog box.

   The first step is to locate the event handler that will be used to display the dialog box. In an application, a dialog box is typically opened in response to a button click or a menu command, but any event can be used.

2. Add the code to show the dialog box.

   You use the **ShowDialog** method to display a dialog box in Windows Forms applications.

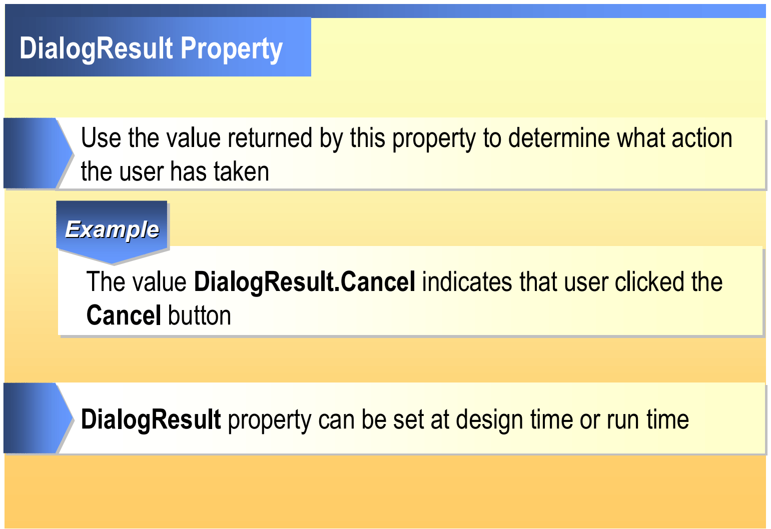
```
private void button1_Click(object sender, System.EventArgs
e)
{
   openFileDialog1.ShowDialog();
}
```

### Procedure: Displaying a message box

You can display a message box by using the **Show** method of the **MessageBox** class. The **Show** method requires that you supply the text of the message and you can optionally specify the following: the dialog box caption, the buttons, the icon, the default button, and options relating to how the message box and the text that it contains will display.

```
public void PerformCalculations()
{
   MessageBox.Show ("The search is now complete",
                    "My Application",
                    MessageBoxButtons.OKCancel,
                    MessageBoxIcon.Asterisk);
}
```

# DialogResult Property



## DialogResult Property

- Use the value returned by this property to determine what action the user has taken

*Example*

The value **DialogResult.Cancel** indicates that user clicked the **Cancel** button

- **DialogResult** property can be set at design time or run time

### Introduction

When you display a dialog box in an application, it is very important to know the result input. For example, if you display a dialog box that prompts the user to **OK** or **Cancel** an action, you need to know whether the user clicked the **OK** or the **Cancel** button.

### DialogResult property

The user input in any dialog box is processed by the form that displays the dialog box. You can use the **DialogResult** property of the form to determine the result of the user input. Based on the value returned by the **DialogResult** property, you can decide whether to discard or use the information returned by the dialog box.

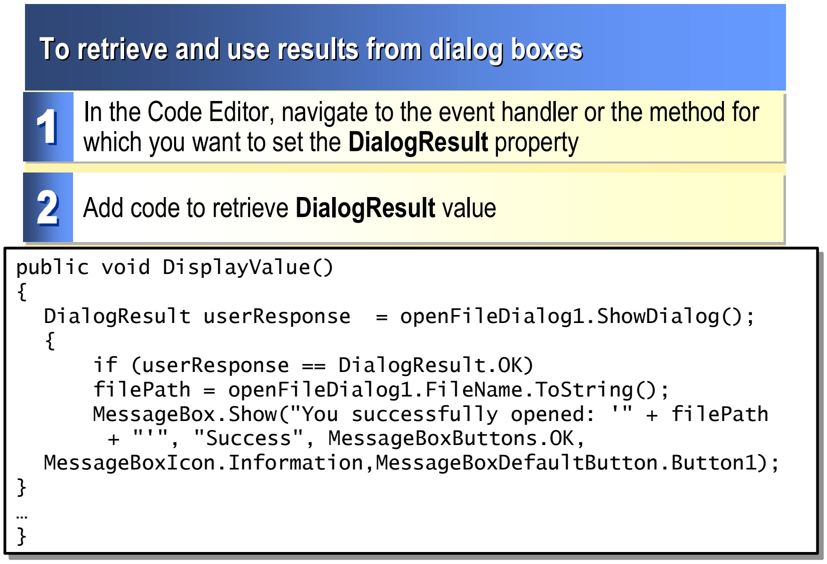
### Example

If a user clicks the **Cancel** button in a dialog box, the value of the **DialogResult** property is set to **DialogResult.Cancel**. The parent form then retrieves this value and discards the information in the dialog box.

### Set DialogResult property at design time or run time

You can set the **DialogResult** property at design time or run time. At design time, you can set the **DialogResult** property for all the button controls in the dialog box. Setting the **DialogResult** property at run time allows you to dynamically handle user responses.

# How to Use Input from Dialog Boxes



To retrieve and use results from dialog boxes

1 In the Code Editor, navigate to the event handler or the method for which you want to set the **DialogResult** property

2 Add code to retrieve **DialogResult** value

```
public void DisplayValue()
{
  DialogResult userResponse  = openFileDialog1.ShowDialog();
  {
      if (userResponse == DialogResult.OK)
      filePath = openFileDialog1.FileName.ToString();
      MessageBox.Show("You successfully opened: '" + filePath
       + "'", "Success", MessageBoxButtons.OK,
  MessageBoxIcon.Information,MessageBoxDefaultButton.Button1);
}
…
}
```

**Introduction**

After the dialog box is closed, the form that displayed the dialog box can reference the values from the **DialogResult** property.

## Procedure

To retrieve and use input from a dialog box:

1. In the Code Editor, navigate to the event handler or the method for which you want to set the **DialogResult** property.
2. Add code to retrieve the **DialogResult** value.

   The following code shows how user input is retrieved from an open file dialog box.

```
public void DisplayValue()
{
   DialogResult userResponse  =
                     openFileDialog1.ShowDialog();
   if (userResponse == DialogResult.OK)
   {
       filePath = openFileDialog1.FileName.ToString();
       MessageBox.Show("You successfully opened: '" +
           filePath + "'",
           "Success",
           MessageBoxButtons.OK,
           MessageBoxIcon.Information,
           MessageBoxDefaultButton.Button1);
   }
   else
   {
       MessageBox.Show("You canceled " +
           "the open file operation.",
           "Warning",
           MessageBoxButtons.OK,
           MessageBoxIcon.Warning,
           MessageBoxDefaultButton.Button1,
           MessageBoxOptions.RightAlign);
   }
}
```

# Demonstration: Using the OpenFileDialog Control

In this demonstration, you will see how to

- Add an **OpenFileDialog** control to your project
- Create the code to display the **OpenFileDialog**
- Set the **OpenFileDialog** properties

### Introduction

In this demonstration, you will see how to use the **OpenFileDialog** control in a Windows Forms application.

### Instructions

► **Open the OpenFileDialog.sln file**

- Open the OpenFileDialog.sln solution in Visual Studio .NET from *install_folder*\Democode\Mod02\Mod02_05\Starter.

► **Add an OpenFileDialog control to your project**

1. Open Form1.cs in Design view.
2. In the Toolbox, double-click **OpenFileDialog**.

► **Create the code to display the OpenFileDialog**

1. Open Form1.cs in the Code Editor.
2. On the **View** menu, point to **Show Tasks**, and then click **Comment**.
3. In the Task List, double-click **TODO: show the OpenFileDialog and check DialogResult**.
4. Add the following code statement below the TODO line:

```
openFileDialog1.ShowDialog();
```

5. Run the application, and then click **Use the OpenFileDialog Control**.
6. Click **Cancel**, and then close the application.
7. What can you do to determine if the dialog box was closed by clicking **Open** or by clicking **Cancel**?

   **You can use the DialogResult property of the parent form.**

8. Replace the 'openFileDialog1.ShowDialog();' code statement with the following code statements:

   ```
   if (openFileDialog1.ShowDialog() == DialogResult.OK)
   {
      filePath = openFileDialog1.FileName.ToString();
      MessageBox.Show("You successfully opened: '"
          + filePath + "'",
          "Success",
          MessageBoxButtons.OK,
          MessageBoxIcon.Information,
          MessageBoxDefaultButton.Button1);
   }
   else
   {
      MessageBox.Show("You canceled the operation.",
          "Warning",
          MessageBoxButtons.OK,
          MessageBoxIcon.Warning,
          MessageBoxDefaultButton.Button1,
          MessageBoxOptions.RightAlign);
   }
   ```

9. Run the application, and then click **Use the OpenFileDialog Control**.
10. Click **Cancel**, and then close the application.

► **Set the OpenFileDialog properties**

1. In the Task List, double-click **TODO: set the initial directory and filter properties**.
2. Add the following code statement below the TODO line:

   ```
   openFileDialog1.InitialDirectory = Application.StartupPath;
   openFileDialog1.Filter = "Text Files (*.txt)|*.txt";
   ```

3. Run the application, and then click **Use the OpenFileDialog Control**.
4. Use both **Open** and **Cancel** to close the dialog box, and then close the application.
5. What happens when you select a file and click **Open**?

   **Properties of the OpenFileDialog are updated to reflect the file that you selected. You can use this information to perform actions on the selected file, such as displaying the contents of the file by using a PrintPreviewDialog.**

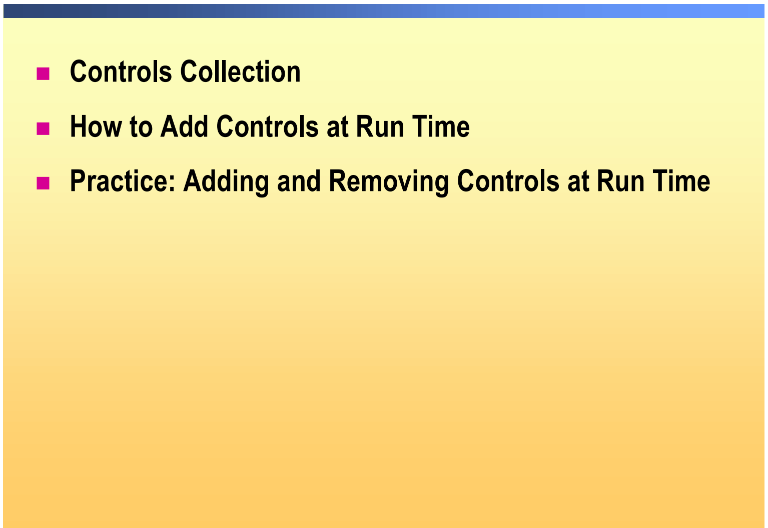
   ____________________________________________________________

   ____________________________________________________________

   ____________________________________________________________

6. If time permits, experiment with additional **OpenFileDialog** properties, such as **Multiselect** and **CheckPathExists**, and modify the appearance of the message dialog boxes.
7. Save your project, and then close Visual Studio .NET.

# Lesson: Adding Controls at Run Time



- Controls Collection
- How to Add Controls at Run Time
- Practice: Adding and Removing Controls at Run Time

**Introduction**

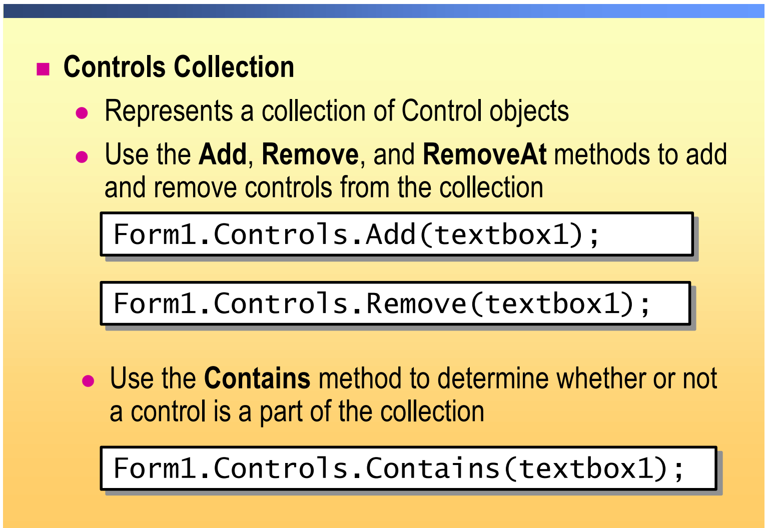
Visual Studio .NET provides you with the flexibility of adding controls at run time. This lesson introduces the controls collection and how controls can be added at run time.

**Lesson objectives**

After completing this lesson, you will be able to:

- Use the **Controls** property to access the **ControlCollection** object of a container.
- Add and remove controls from a container at run time.

# Controls Collection



- Controls Collection
  - Represents a collection of Control objects
  - Use the **Add**, **Remove**, and **RemoveAt** methods to add and remove controls from the collection

```
Form1.Controls.Add(textbox1);
```

```
Form1.Controls.Remove(textbox1);
```

  - Use the **Contains** method to determine whether or not a control is a part of the collection

```
Form1.Controls.Contains(textbox1);
```

### Introduction

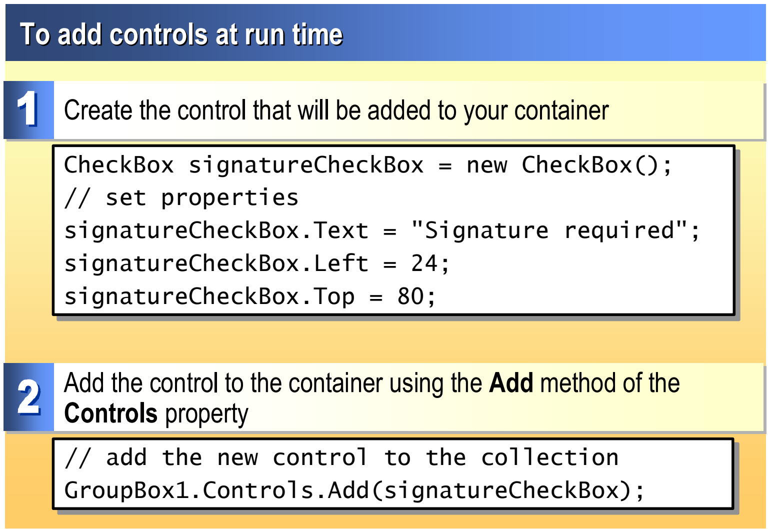
Visual Studio .NET provides a collection object that contains all the controls on a form or on a container control. This object is known as the **ControlCollection** object and is accessed by using the **Controls** property of the form or control. The controls collection represents a collection of Control objects. You can add and remove controls from a container at run time by using the **Controls** property. The **ControlCollection Count** property returns the number of controls on the container and can be used to loop through the controls collection.

### ControlCollection Methods

The following table lists some of the methods of **ControlCollection**.

| **Method** | **Description** |
|---|---|
| Add | Adds the specified control to the control collection. |
| AddRange | Adds an array of control objects to the collection. |
| Clear | Removes all controls from the collection. |
| Contains | Determines whether the specified control is a member of the collection. |
| Remove | Removes the specified control from the control collection. |
| RemoveAt | Removes a control from the control collection at the specified indexed location. |
| ToString (inherited from Object) | Returns a String that represents the current Object. |
| IndexOf | Retrieves the index of the specified control in the control collection. |
| GetEnumerator | Returns an enumerator that can be used to iterate through the control collection. |

# How to Add Controls at Run Time



To add controls at run time

1 Create the control that will be added to your container

```
CheckBox signatureCheckBox = new CheckBox();
// set properties
signatureCheckBox.Text = "Signature required";
signatureCheckBox.Left = 24;
signatureCheckBox.Top = 80;
```

2 Add the control to the container using the **Add** method of the **Controls** property

```
// add the new control to the collection
GroupBox1.Controls.Add(signatureCheckBox);
```

### Introduction

You can add and remove controls at run time by using the **Controls** property. The ability to add controls at run time allows you to customize your application based on user input. For example, you may want the application to display additional menu items depending on the user input.

### Procedure

To add controls at run time:

1. Create a control that will be added to the form at run time.

   The following code shows an example of how to create a **CheckBox** control.

   ```
   CheckBox signatureCheckBox = new CheckBox();
   ```
2. Add the control to the form or the container control by using the **Add** method of the **Controls** property.

   The following code adds the button control at a specific location in the form.

   ```
   signatureCheckBox.Name = "myCheckBox";
   signatureCheckBox.Text = "Signature required";
   signatureCheckBox.Width = 224;
   signatureCheckBox.Left = 24;
   signatureCheckBox.Top = 80;

   GroupBox1.Controls.Add(signatureCheckBox);
   ```

**Note** When adding several controls to a parent control, it is recommended that you call the **SuspendLayout** method before initializing the controls to be added. After adding the controls to the parent control, call the **ResumeLayout** method. This will increase the performance of applications with many controls.

# Practice: Adding and Removing Controls at Run Time

In this practice, you will

- Remove unwanted controls
- Add a new control
- Specify properties of the new control

Begin reviewing the objectives for this practice activity

**Introduction**

In this practice, you will add and remove controls at run time.

**Instructions**

### ► Open the practice project

1. Using Windows Explorer, navigate to *install_folder*\Practices\Mod02\Mod02_03\Starter.
2. Double-click the AddingAndRemovingControls.sln solution file to open the project.

### ► Remove unwanted controls

1. Open Form1.cs in the Code Editor.
2. On the **View** menu, point to **Show Tasks**, and then click **Comment**.
3. In the Task List, double-click **TODO: remove controls from GroupBox1**.
4. Add the following code statements below the TODO line:

```
lcv = GroupBox1.Controls.Count;
while (lcv > 2)
{
   GroupBox1.Controls.Remove(GroupBox1.Controls[lcv - 1]);
   lcv -= 1;
}
```

5. Run the application, click **Use International Policies**, and then click **Use Domestic Policies**.
6. Close the application.

### ► Add a new control

1. In the Task List, double-click **TODO: create an instance of a CheckBox**.
2. Add the following code statement below the TODO line:

```
CheckBox decimalCheckBox = new CheckBox();
```

3. In the Task List, double-click **TODO: add a control to GroupBox1**.
4. Add the following code statement below the TODO line:

```
GroupBox1.Controls.Add(decimalCheckBox);
```

5. Run the application, click **Use International Policies**, and then click **Use Domestic Policies**.
6. Why does the **decimalCheckBox** appear in the upper-left corner of **GroupBox1**?

   **The default values for the location property of a control are zero for both the X and the Y coordinates.**

   ____________________________________________________________

   ____________________________________________________________

7. Close the application.

### ► Specify properties of the new control

1. In the Task List, double-click **TODO: specify the control properties**.
2. Add the following code statement below the TODO line:

```
decimalCheckBox.Left = 24;
decimalCheckBox.Top =
GroupBox1.Controls[GroupBox1.Controls.Count - 1].Top
   + 32;
decimalCheckBox.Text = "Use comma seperated decimals";
decimalCheckBox.Name = "commaSeperatedDecimalsCheckBox";
```

3. Run the application, click **Use International Policies**, and then click **Use Domestic Policies**.
4. Close the application.
5. In the Task List, double-click **TODO: specify the control properties**.
6. Add the following code statement below the TODO line:

```
decimalCheckBox.Width = 224;
```

7. Run the application, click **Use International Policies**, and then click **Use Domestic Policies**.
8. Close the application.

9. If time permits, open Form1.cs in Design view and perform the following steps:

   a. Drag the **Use local dataset when available CheckBox** from **GroupBox1** to **GroupBox2** and then drag it back to its original position on **GroupBox1**.

      This is now the last control added to **GroupBox1**.

   b. Run the application and test your code by clicking the radio buttons.

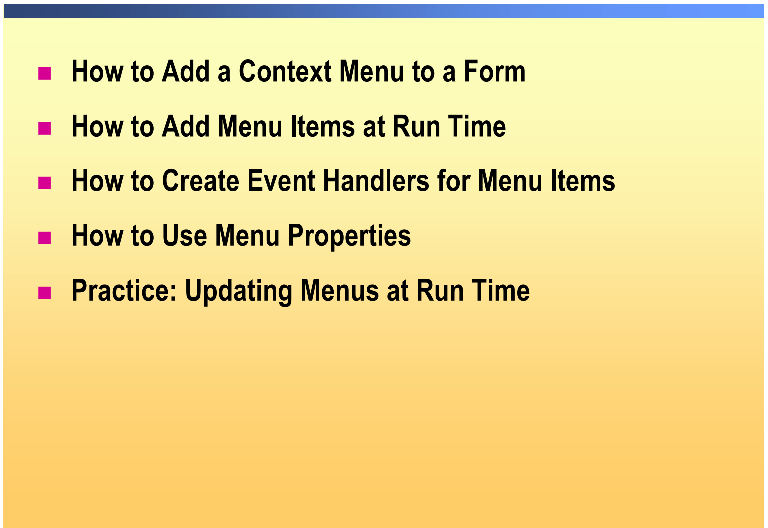
   c. Remove **Print report automatically** from **GroupBox1** and then replace it.

      **Print report automatically** is now the last control in the **ControlCollection** of **GroupBox1**.

   d. Test your application again to see that your code now adds controls to **GroupBox1** in the desired locations.

10. Save your project, and then close Visual Studio .NET.

# Lesson: Creating Menus



- How to Add a Context Menu to a Form
- How to Add Menu Items at Run Time
- How to Create Event Handlers for Menu Items
- How to Use Menu Properties
- Practice: Updating Menus at Run Time

**Introduction**

Menus and context menus are ways of exposing functionality to your users or alerting them to important information in your application. Menus hold commands, grouped by a common theme. Context menus pop up in response to a right-click of the mouse and hold commonly used commands for a particular area of an application.

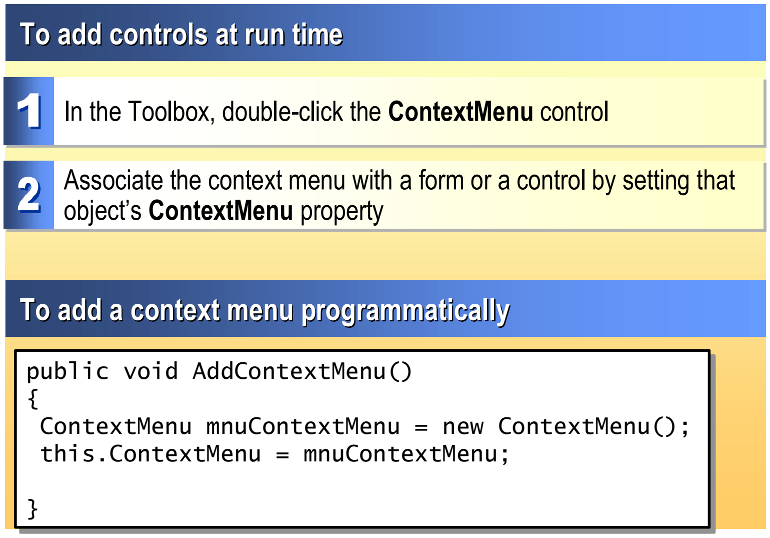
**Note** You can have multiple main menus for an application. If your application is large, then you could use different menus for different parts of an application.

**Lesson objectives**

After completing this lesson, you will be able to:

- Create context menus.
- Add menu items at run time.
- Create shortcuts for menus.
- Enable the checked property for menu items.

# How to Add a Context Menu to a Form



**To add controls at run time**

1 In the Toolbox, double-click the **ContextMenu** control

2 Associate the context menu with a form or a control by setting that object's **ContextMenu** property

**To add a context menu programmatically**

```
public void AddContextMenu()
{
 ContextMenu mnuContextMenu = new ContextMenu();
 this.ContextMenu = mnuContextMenu;

}
```

**Introduction**

The Windows Forms **ContextMenu** control is used to provide users with an easily accessible menu of frequently used commands that are associated with the selected object. The items in a context menu are frequently a subset of the items from main menus that appear elsewhere in the application. Context menus are usually available by right-clicking the mouse. On Windows Forms they are associated with controls.

**Procedure: Adding a context menu at design time**

To add a context menu at design time:

1. Add the context menu from the Toolbar to the form.

   In the Toolbox, double-click the **ContextMenu** control.

2. Associate the context menu with a form or a control by setting that object's **ContextMenu** property.

   You can associate a context menu with a control by setting the control's **ContextMenu** property to the **ContextMenu** control. You can associate a single context menu with multiple controls, but each control can have only one context menu.

**Procedure: Adding a context menu at design time**

To add a context menu at design time:

1. Create a new method that will include the code required to create a menu item.

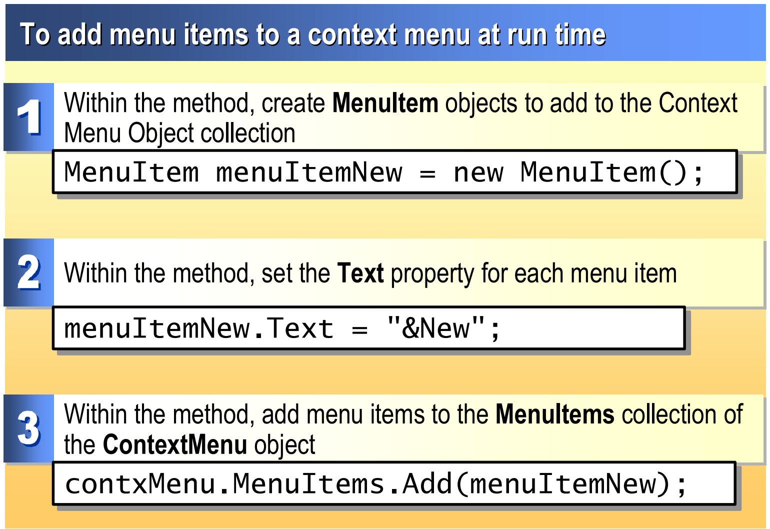
```
public void AddContextMenu()
{

}
```

2. Create an instance of the context menu.

   Add the code as shown to create an instance of the context menu.

```
ContextMenu mnuContextMenu = new ContextMenu();
this.ContextMenu = mnuContextMenu;
```

# How to Add Menu Items at Run Time



To add menu items to a context menu at run time

1 Within the method, create **MenuItem** objects to add to the Context Menu Object collection

```
MenuItem menuItemNew = new MenuItem();
```

2 Within the method, set the **Text** property for each menu item

```
menuItemNew.Text = "&New";
```

3 Within the method, add menu items to the **MenuItems** collection of the **ContextMenu** object

```
contxMenu.MenuItems.Add(menuItemNew);
```

### Introduction

After you add a **ContextMenu** control to your form, you add menu items to it. Menu contents are stored in a collection. You can add menu items to a context menu at run time by adding **MenuItem** objects to this collection.

### Procedure

The key property of the **ContextMenu** control is the **MenuItems** property. You can add menu items by programmatically creating **MenuItem** objects and adding them to the **MenuItems** collection of the context menu. Because the items in a context menu are usually drawn from other menus, you will most frequently add items to a context menu by copying them. You can also disable, hide, or delete menu items.

To add items to a context menu at run time:

1. In the method to create a context menu, create **MenuItem** objects which are added to the **Controls** collection.

   ```
   MenuItem menuItemNew = new MenuItem();
   ```

2. Set the **Text** property for each menu item.

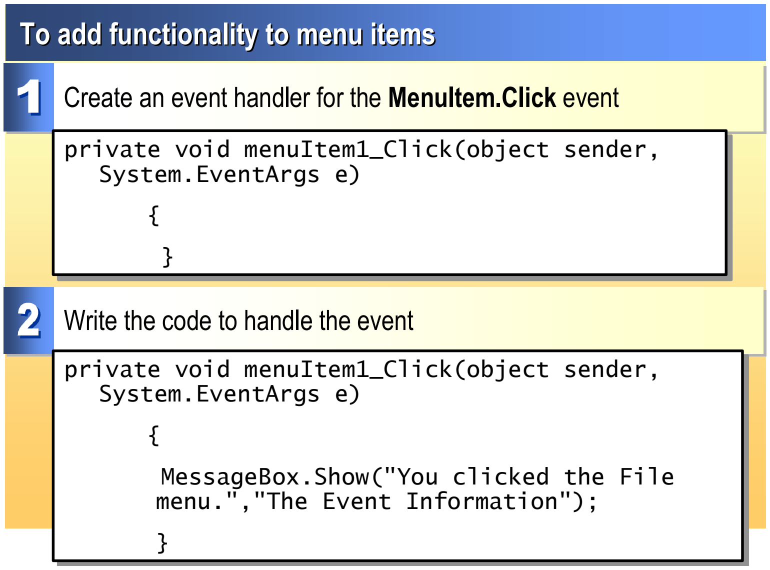
   ```
   menuItemNew.Text = "&New";
   ```

3. Add menu items to the **MenuItems** collection of the **ContextMenu** object.

   ```
   contxMenu.MenuItems.Add(menuItemNew);
   ```

**Note** You can add images to menu items. To add images to menu items, create an instance of the **MenuItem** class, override the **OnPaint()** method, and draw an image to the left of the menu item text.

# How to Create Event Handlers for Menu Items



To add functionality to menu items

1 Create an event handler for the **MenuItem.Click** event

```
private void menuItem1_Click(object sender,
   System.EventArgs e)
      {
       }
```

2 Write the code to handle the event

```
private void menuItem1_Click(object sender,
   System.EventArgs e)
      {
       MessageBox.Show("You clicked the File
      menu.","The Event Information");
       }
```

**Introduction**

After a menu structure is established, you provide functionality to it. You provide functionality to a menu by creating an event handler for the **MenuItem.Click** event and writing code to handle the event.

**Procedure**

To add functionality to menu items:

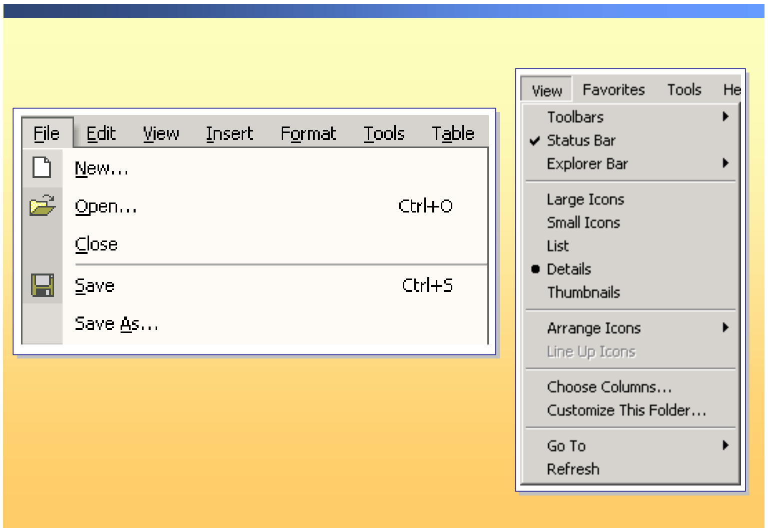
1. Create an event handler for **the MenuItem.Click** event.

```
private void menuItem1_Click(object sender,
                             System.EventArgs e)
{

}
```

2. Write the code to handle the **MenuItem.Click** event.

```
private void menuItem1_Click(object sender,
                             System.EventArgs e)
{
   MessageBox.Show("You clicked menuItem1",
                        "Menu Event");
}
```

# How to Use Menu Properties



### Introduction

Menu items include several properties that enable you to enhance the user experience. These properties include properties for creating shortcut keys and enabling and disabling menu items by using check marks, radio buttons, and so on.

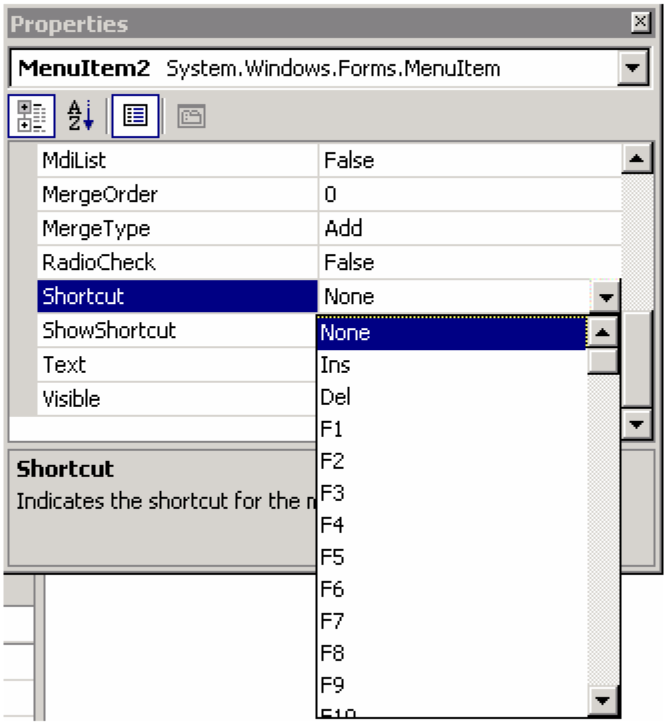
### Procedure: Creating shortcut keys

Shortcut keys provide a method for users to activate frequently used menu items in the menu system and to provide keyboard access to the application.

For example, in Microsoft Word, you can open a file by using the Ctrl+O shortcut or save a file by pressing Ctrl+S.

To create a shortcut for menu items:

1. Select the menu item for which you need to create a shortcut.
2. In the **Properties** dialog box, select the **Shortcut** property.



Properties

MenuItem2 System.Windows.Forms.MenuItem

| | |
|---|---|
| MdiList | False |
| MergeOrder | 0 |
| MergeType | Add |
| RadioCheck | False |
| Shortcut | None |
| ShowShortcut | |
| Text | |
| Visible | |

**Shortcut**
Indicates the shortcut for the m

3. Select the required shortcut in the shortcut list.
4. In the **Properties** dialog box, set the **ShowShortcut** property to **True**.

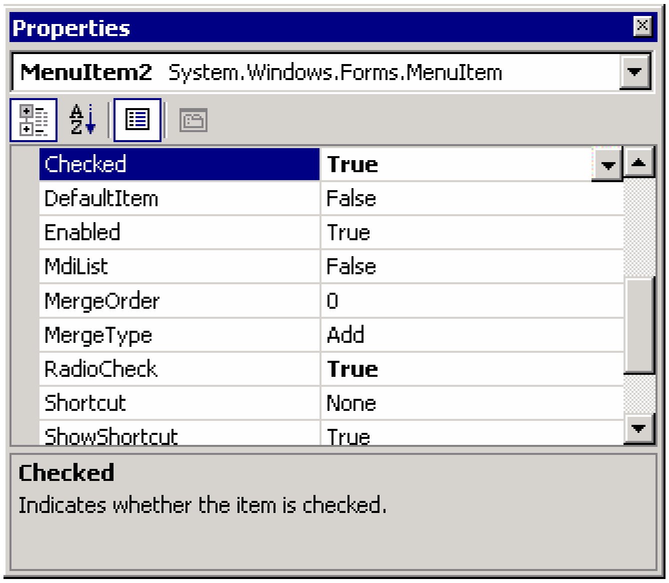
### Procedure: Displaying checked menu items

You can use the **Checked** and **RadioCheck** property to identify the selected menu item in a group of mutually exclusive menu items. You can also place a check mark on a menu item in a group of items to identify the size of the font to be displayed for the text in an application.

For example, mutually exclusive items on the **View** menu in Windows Explorer use a check or a radio button to show the selected option.

To display checked menu items:

1. Select the menu item.
2. In the **Properties** dialog box, set the **Checked** property to **True**.



**MenuItem2** System.Windows.Forms.MenuItem

| | |
|---|---|
| Checked | **True** |
| DefaultItem | False |
| Enabled | True |
| MdiList | False |
| MergeOrder | 0 |
| MergeType | Add |
| RadioCheck | **True** |
| Shortcut | None |
| ShowShortcut | True |

**Checked**
Indicates whether the item is checked.

To display the radio check for menu items:

1. Select the menu item.
2. In the **Properties** dialog box, set the **Checked** property to **True**.
3. Set the **RadioCheck** property to **True**.

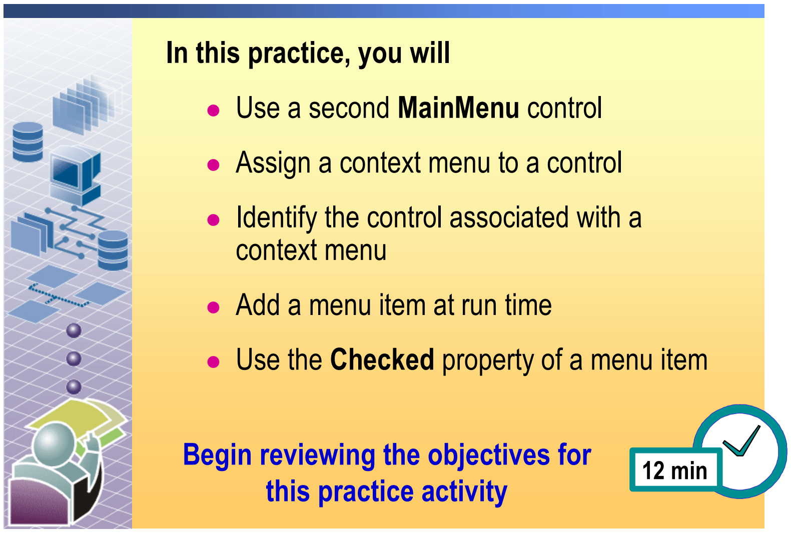
### Procedure: Enabling and disabling menu items

You may want to disable certain menu items for users depending on their roles, permission, or their input. You can use the **Enable** property to enable or disable menu item. If the value of the **Enable** property is set to **True**, the menu item is enabled. However, if the value is set to **False**, the menu item is disabled.

To enable or disable menu items:

1. Select the menu item.
2. Set the **Enabled** property to **True** or **False**.

# Practice: Updating Menus at Run Time



In this practice, you will

- Use a second **MainMenu** control
- Assign a context menu to a control
- Identify the control associated with a context menu
- Add a menu item at run time
- Use the **Checked** property of a menu item

Begin reviewing the objectives for this practice activity

## Introduction

In this practice, you will update the menus in an application at run time.

## Instructions

### ► Open the practice project

1. Using Windows Explorer, navigate to *install_folder*\Practices\Mod02\Mod02_04\Starter.
2. Double-click the RuntimeMenus.sln solution file to open the project.

### ► Use a second MainMenu

1. Run the application, open the **File** menu, and then open the **View** menu.
2. On the **Reports** tab, open the **File** menu, open the **View** menu, and then close the application.

   Notice that the menu items do not really match the requirements of the Reports tab.
3. In Solution Explorer, click **Form1**, and then click **View Code**.
4. On the **View** menu, point to **Show Tasks**, and then click **Comment**.
5. In the Task List, double-click **TODO: assign a different MainMenu to a form**.
6. Add the following code statement below the TODO line:

```
this.Menu = MainMenu2;
```

7. Run the application, open the **File** menu, and then open the **View** menu.

8. Click the **Reports** tab, open the **File** menu, and then open the **View** menu again.
9. When would you use more than one **MainMenu** control in an application?

   **It can be helpful to use (display) more than one MainMenu object when the context of your application changes, or when your application has multiple states.**

10. Close the **View** menu, and then close the application.

► **Assign a ContextMenu to a control**

1. In the Task List, double-click **TODO: create a control and assign a context menu to it**.
2. Add the following code statements below the TODO line:

```
Label fileLabel1 = new Label();
fileLabel1.Dock = DockStyle.Top;
fileLabel1.Text = Application.StartupPath & "\Chai.txt";
fileLabel1.ContextMenu = ContextMenu1;
this.Panel1.Controls.Add(fileLabel1);
```

3. Run the application, right-click the **Label** control displaying the Chai.txt file path, and then click **Open**.
4. Click **OK**, and then close the application.
5. How can you identify which control is associated with a **ContextMenu**?

   **The ContextMenu.SourceControl property gets the control that displayed the shortcut menu.**

► **Identify the control associated with a ContextMenu**

1. In the Task List, double-click **TODO: use the SourceControl property**.
2. Add the following code statements below the TODO line:

```
Panel1.Controls.Remove(ContextMenu1.SourceControl);
```

3. Run the application, right-click the **Label** control displaying the Chai.txt file path, and then click **Remove from list**.
4. On the **View** menu, click **Show Previously Opened Files**, and then open the **File** menu.
5. Close the **File** menu, and then close the application.
6. What method of a **MenuItem** object is used to add a menu item at run time?

   **The Add method.**

### ► Add a menu item at run time

1. In the Task List, double-click **TODO: add a menu item to a menu**.
2. Add the following code statements below the TODO line:

```
fileMenuItem.MenuItems.Add(previousFile1);
```

3. Run the application, and then, on the **View** menu, click **Show Previously Opened Files**.
4. Open the **File** menu, and then open the **View** menu.

   Notice that the item has been added to the **File** menu.
5. Close the application.
6. Is there any way to show the user that you are currently displaying the previously opened file on the **File** menu?

   **You can use the Checked property of a menu item to indicate when a menu item has been selected. By switching this property on and off each time it is clicked and adding the appropriate code to your application, you can make the menu item perform like a radio button.**

   ____________________________________________________________

   ____________________________________________________________

   ____________________________________________________________

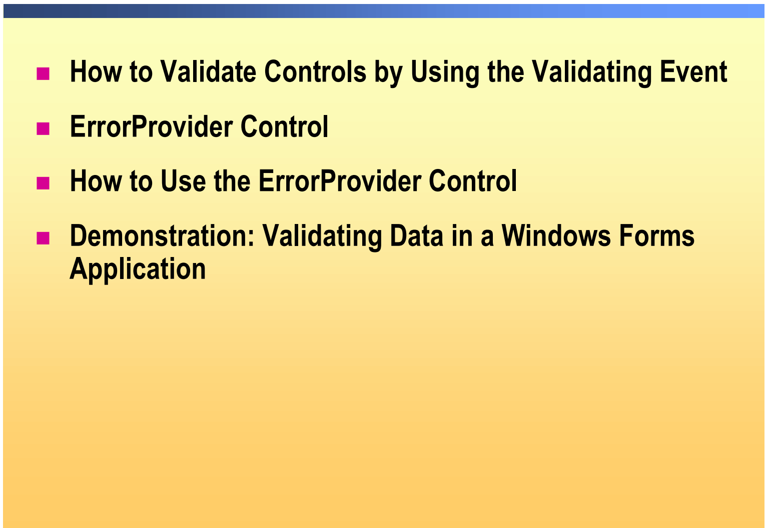
### ► Use the Checked property of a menu item to signal the user

1. In the Task List, double-click **TODO: display a menu item as checked**.
2. Add the following code statements below the TODO line:

```
viewShowPreviousMenuItem.Checked = true;
```

3. Run the application, and then, on the **View** menu, click **Show Previously Opened Files**.
4. Open the **File** menu, and then open the **View** menu.
5. On the **View** menu, click **Show Previously Opened Files**.
6. Open the **File** menu, and then open the **View** menu.
7. If time permits, examine the code used to remove a menu item from the **File** menu and the code used to respond to the click events of the second context menu.
8. Close the application, save your project files, and then close Visual Studio .NET.

# Lesson: Validating User Input



- How to Validate Controls by Using the Validating Event
- ErrorProvider Control
- How to Use the ErrorProvider Control
- Demonstration: Validating Data in a Windows Forms Application

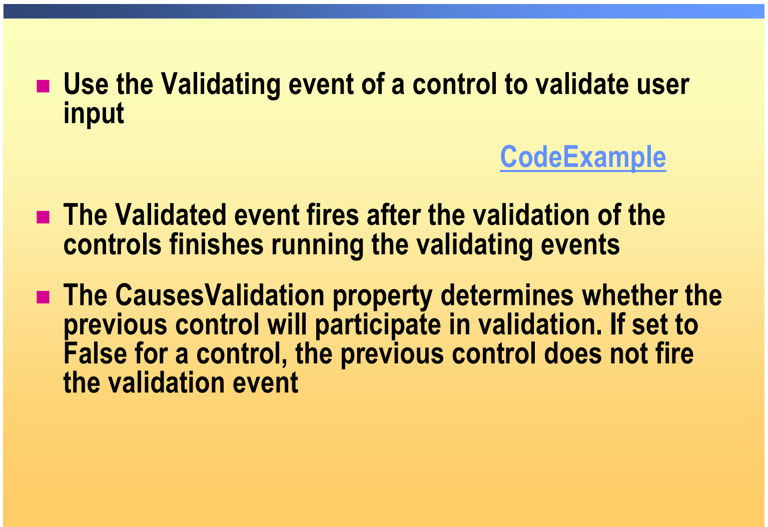
Whether you are writing a simple calculator application or an interface for a complex database, you will need to validate information entered by a user. This lesson describes how to validate user input in a Windows Forms application. It also describes how to display an error message if the user input is invalid.

**Lesson objectives**

After completing this lesson, you will be able to:

- Validate user input by using the **Validating** event of a control.
- Use the **ErrorProvider** control to notify the user when an entered value does not meet acceptance criteria.
- Display appropriate error messages when users enter invalid data.

# How to Validate Controls by Using the Validating Event



- Use the Validating event of a control to validate user input

CodeExample

- The Validated event fires after the validation of the controls finishes running the validating events
- The CausesValidation property determines whether the previous control will participate in validation. If set to False for a control, the previous control does not fire the validation event

### Introduction

In most applications, a user enters information for the application to process. Data validation ensures that every data value entered into your application is accurate and of a valid data type.

Visual Studio .NET includes the **Validating** event for controls, which occurs before a control loses focus. This event occurs only when the **CausesValidation** property of the control that is about to receive the focus is set to **True** (which is the default). Use both the **Validating** event and **CausesValidation** property for a control to evaluate input before allowing the user to move focus away from that control.

### Validating event

The simplest way to validate data in a Windows Forms application is to use the **Validating** event. The **Validating** event also allows you to control when focus can be moved to other controls. By using the **Validating** event, you can prevent the focus from shifting to another control until all validation rules have been met. Possible uses for the **Validating** event include:

- A data entry application must perform more sophisticated validation than is provided by the **Masked Edit** control.
- A form must prevent users from moving off a control, by pressing TAB or a shortcut key, until data has been entered in a field.

### CausesValidation property

The **CausesValidation** property works with the **Validating** event to limit when a control can lose focus. You can set the **CausesValidation** property to determine whether the **Validating** event will occur on a second control from which the focus is being shifted. If the **Validating** event is being handled for a **TextBox**, and the user clicks a button that has its **CausesValidation** property set to **True**, the **Validating** event for the text box will fire. However, if the **CausesValidation** property for the button is set to **False**, the **Validating** event will not fire. By default, the **CausesValidation** property is set to **True** for all controls.

### Procedure: Validating a control

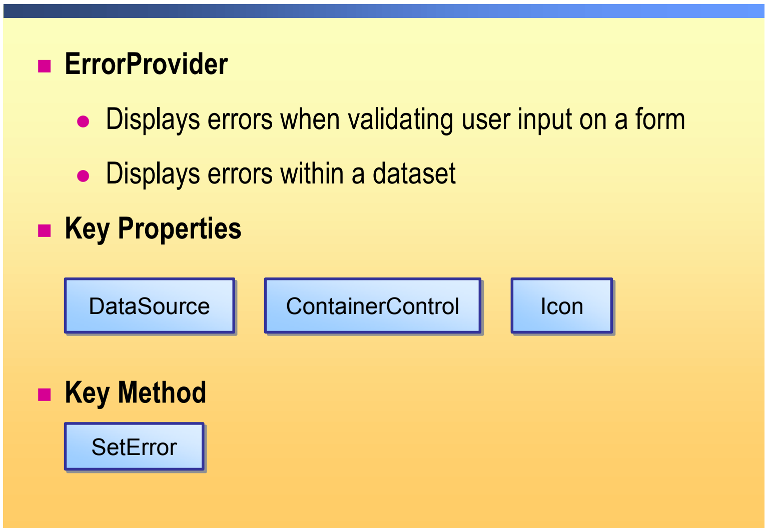
To use the **Validating** event of a **TextBox** control:

1. Add a **TextBox** control to a form.
2. In the Code Editor, in the procedure list, click the **Validating** event.
3. Type a validation code in the **Validating** event for the **TextBox**.

```
private void minValueTextBox_Validating(object sender,
                  System.ComponentModel.CancelEventArgs e)
{
   if (Convert.ToInt32(minValueTextBox.Text) >=
                  Convert.ToInt32(maxValueTextBox.Text))
   {
       e.Cancel = true;
       MessageBox.Show("You must enter a minimum value " +
               "that is less than the maximum value");
   }
}
```

4. Set the **CausesValidation** property to **False** for any controls for which you do not want the **Validating** event to fire, such as a **Help** button.

# ErrorProvider Control



- **ErrorProvider**
  - Displays errors when validating user input on a form
  - Displays errors within a dataset
- **Key Properties**

DataSource ContainerControl Icon

- **Key Method**

SetError

****************************ILLEGAL FOR NON-TRAINER USE******************************

**Introduction**

The Windows Forms **ErrorProvider** control allows you to display an error message if the data entered by the user is invalid. It is used with validating user input on a form or displaying errors in a dataset.

**Advantage of using ErrorProvider**

An error provider is a better alternative than displaying an error message in a message box. If you display the error message in a message box, the error message is no longer visible if the user dismisses the message box. In the case of the **ErrorProvider** control, it displays an error icon next to the relevant control. When the user positions the mouse pointer over the error icon, a ToolTip appears, showing the error message string.

**ErrorProvider properties**

The key properties of the **ErrorProvider** control are:

- **ContainerControl**

  You must set the **ContainerControl** property to the appropriate container for the **ErrorProvider** control to display an error icon on the form. When you add the control in the designer, the **ContainerControl** property is automatically set to the containing form. If you add the control in code, you must set it manually.

- **DataSource**

  When the **DataSource** property is set, the **ErrorProvider** control displays error messages for a dataset.

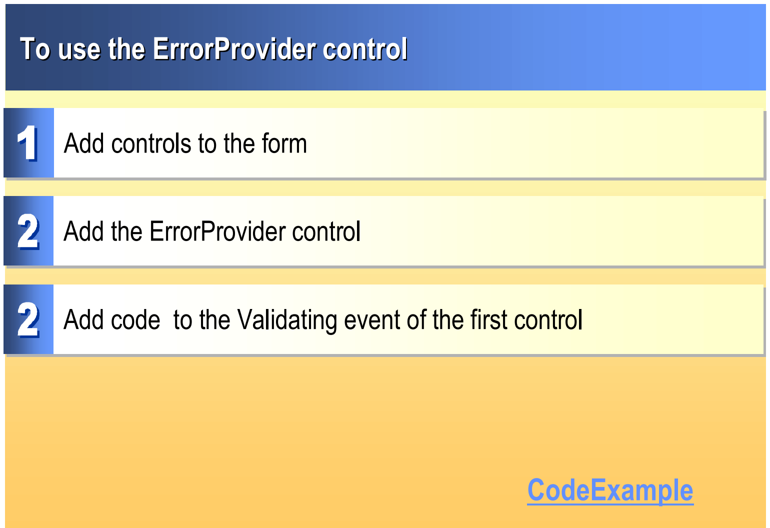
- **Icon**

  The **Icon** property can be set to a custom error icon instead of the default.

**ErrorProvider methods**

The key method of the **ErrorProvider** control is the **SetError** method, which specifies the control that the error icon should appear next to, and the value of the error message string.

# How to Use the ErrorProvider Control



To use the ErrorProvider control

1 Add controls to the form

2 Add the ErrorProvider control

2 Add code  to the Validating event of the first control

CodeExample

## Introduction

You can use a Windows Forms **ErrorProvider** control to display an error icon when the user enters invalid data.

## Procedure

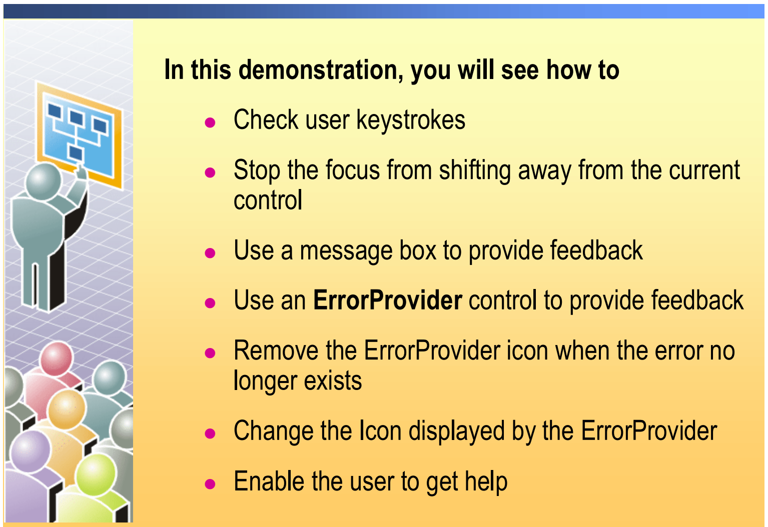
To display an error icon by using the **ErrorProvider** control:

1. Add the controls to be validated to the form.
2. Add the **ErrorProvider** control.
3. Select the first control and add code to its **Validating** event handler.

   The following code tests for integer values in a text box. If the data is invalid, an error icon displays next to the text box.

```
protected void textBox1_Validating (object sender,
                                    CancelEventArgs e)
{
   try
   {
      int x = Int32.Parse(textBox1.Text);
      errorProvider1.SetError(textBox1, "");
   }
   catch
   {
      errorProvider1.SetError(textBox1,
                  "Not an integer value.");
   }
}
```

# Demonstration: Validating Data in a Windows Forms Application



**In this demonstration, you will see how to**

- Check user keystrokes
- Stop the focus from shifting away from the current control
- Use a message box to provide feedback
- Use an **ErrorProvider** control to provide feedback
- Remove the ErrorProvider icon when the error no longer exists
- Change the Icon displayed by the ErrorProvider
- Enable the user to get help

## Introduction

In this demonstration you will see how to validate user input in a Windows Forms application.

## Instructions

### ► Open the ValidatingInput.sln file

1. Open the ValidatingInput.sln solution in Visual Studio .NET from *install_folder*\Democode\Mod02\Mod02_06\Starter.
2. Open Form1.cs in Design view.
3. In the Toolbox, double-click **ErrorProvider**.

### ► Check user keystrokes

1. Open the Form1.cs in the Code Editor.
2. On the **View** menu, point to **Show Tasks**, and then click **Comment**.
3. In the Task List, double-click **TODO: check for valid characters**.
4. Add the following code statements below the TODO line:

```
if (!char.IsDigit(e.KeyChar) )
{
   e.Handled = true;
   MessageBox.Show("Integer numbers only");
}
```

5. Run the application, press a letter key on the keyboard, and then click **OK**.
6. Close the application.
7. What happens if you don't include the code statement that sets the value of the **Handled** property to **true**?

   **The character representing the key that was pressed will be added to the text box.**

### ► Stop the focus from shifting away from the current control

1. In the Task List, double-click **TODO: don't let the focus shift**.
2. Add the following code statement below the TODO line:

```
e.Cancel = true;
```

3. Run the application, type **15** and then click **Submit Data and Exit**.
4. Type **5** and then click **Submit Data and Exit**.
5. Close the application.

### ► Use a message box to provide feedback

1. In the Task List, double-click **TODO: give the user feedback using a messagebox**.
2. Add the following code statement below the TODO line:

```
MessageBox.Show("You must enter a minimum value that " +
   "is less than the maximum value");
```

3. Run the application, type **15** and click **Submit Data and Exit**.
4. Type **5** and then click **Submit Data and Exit**.
5. Close the application.

### ► Use an ErrorProvider control to provide feedback

1. Open Form1.cs in the Code Editor.
2. In the Task List, double-click **TODO: use the error provider to provide a message**.
3. Add the following code statement below the TODO line:

```
errorProvider1.SetError(epMinTextBox,
   "The minimum value must be smaller " +
   "than the maximum value");
```

4. Run the application, press TAB, press TAB again, type **15** and then click **Submit Data and Exit**.
5. Position the mouse over the **ErrorProvider** icon that is displayed to the right of the text box.
6. Type **5** and then click inside one of the other text boxes on the form.
7. What can you do to remove the ErrorProvider message?

   **Set the error description value of the SetError property to a zero length string.**

   ____________________________________________________________

8. Close the application.

### ► Remove the ErrorProvider icon when the error no longer exists

1. In the Task List, double-click **TODO: reset the error provider**.
2. Add the following code statement below the TODO line:

```
errorProvider1.SetError(epMinTextBox, "");
```

3. Run the application, press TAB, press TAB again, type **15** and then click **Submit Data and Exit**.
4. Type **5** and then click inside one of the other text boxes on the form.
5. Close the application.

### ► Change the icon displayed by the ErrorProvider

1. In the Task List, double-click **TODO: change the icon displayed by the error provider**.
2. Add the following code statements below the TODO line:

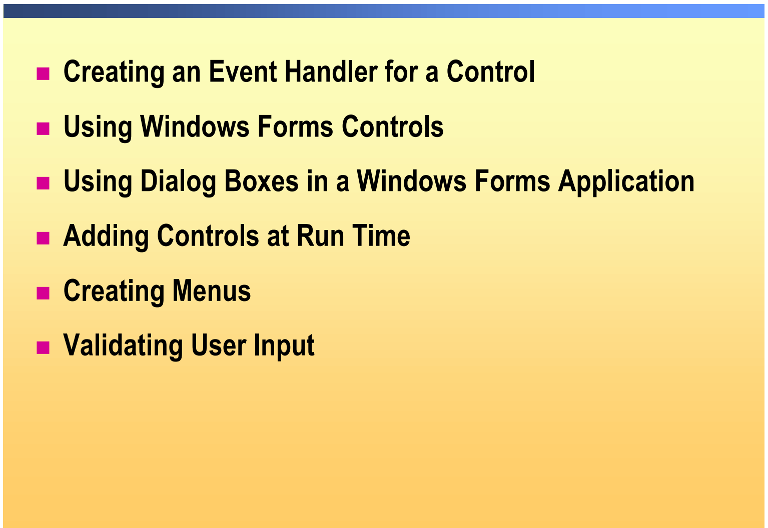
```
Icon ico = new Icon(Application.StartupPath +
   @"\msgbox01.ico");
errorProvider1.Icon = ico;
```

3. Run the application, press TAB, press TAB again, type **6** and then press TAB.
4. Type **4** and then click **Submit Data and Exit**.
5. Type **8** and then click **Submit Data and Exit**.

### ► Enable the user to get help

1. Run the application, press TAB, press TAB again, type **15** and then click **Help**.
2. Type **4** and then click **Submit Data and Exit**.
3. Open Form1.cs in Design view.
4. Click **Help**, and then, in the Properties window, set the value of **CausesValidation** to False.
5. Run the application, press TAB, press TAB again, type **15** and then click **Help**.
6. Close the application.
7. Save your project, and then close Visual Studio .NET.

# Review



- Creating an Event Handler for a Control
- Using Windows Forms Controls
- Using Dialog Boxes in a Windows Forms Application
- Adding Controls at Run Time
- Creating Menus
- Validating User Input

1. How are events raised and handled in the .NET Framework?

   **Events are raised by an event sender in response to either a user action or an application-generated occurrence. A delegate is used to connect the event sender with the event receiver that is used to handle the event.**

2. In the user interface of a hotel room reservation application, you want to let users choose a mode of payment (credit, cash, or check). What controls could you use to create this feature in your application?

   **There are several ways to provide the user with several options and restrict them to a single selection. Since there are only three options in this example, one of the simplest solutions would be to provide three RadioButton controls on the form. For situations involving a larger number of options, one of the list controls would be a better choice (for example, the ListBox control).**

3. How do you display tooltips for the Toolbar buttons?

   **You can create tooltips for the buttons on a ToolBar control by opening the ToolBarButton Collection Editor and specifying a value for the ToolTipText property of a ToolBarButton.**

4. How can you determine the action taken by a user when the user closes a dialog box?

   **The DialogResult property of the parent form is used to capture the action taken to close a dialog box. For example, DialogResult can be used to determine whether the OpenFileDialog was closed by clicking Open or by clicking Cancel.**

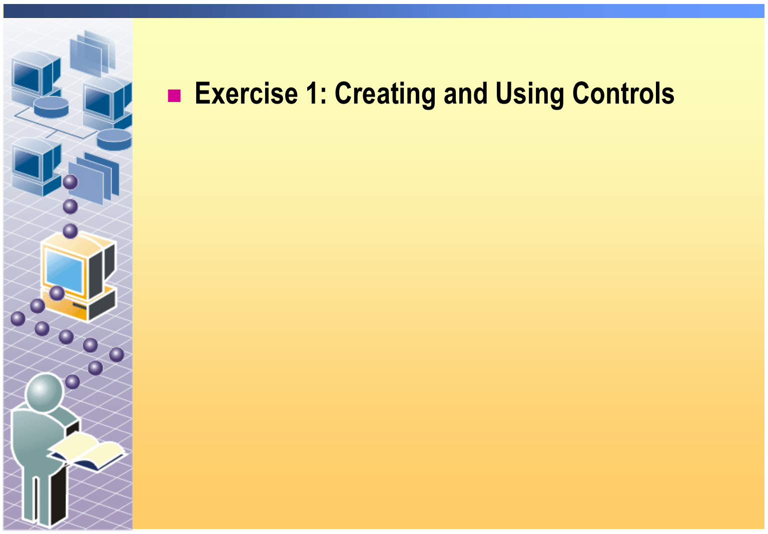
5. How is an **ErrorProvider** control used?

   **The ErrorProvider control is used to associate an invalid-data error message with the controls on a form. To use the ErrorProvider control, open the Validating event handler for a control and use the SetError method of the ErrorProvider control to specify the control and the error message. At run time, an icon will appear next to the specified control.**

6. What property of a container control is used to access the **ControlCollection** and what are the primary properties and methods of the **ControlCollection** class that can be accessed using this property?

   **The Controls property of a container is used to access the ControlCollection object. The primary properties and methods of the ControlCollection that can be accessed through the Controls property are Count, Add, and Remove.**

# Lab 2.1: Working with Controls



- Exercise 1: Creating and Using Controls

**Objectives**

After completing this lab, you will have demonstrated your ability to:

- Create an event handler for a control.
- Create and use a **ToolBar** control.
- Add and remove a control from **ControlCollection** at run time.
- Assign a context menu to a control at run time.

**Note** This lab focuses on the concepts in Module 2, “Working with Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).* As a result, this lab may not comply with Microsoft security recommendations.

**Prerequisites**

Before working on this lab, you must have:

- The knowledge and skills to develop a simple Windows Forms application by using a Visual Studio .NET–compatible programming language.
- The knowledge and skills to create and use a **ToolBar** control.
- The knowledge and skills to add and remove controls from **ControlCollection** at run time.
- The knowledge and skills to use a context menu as a shortcut menu for a control.

## Scenario

You are an application developer at a trading company called Northwind Traders. The department that you work in is developing a Purchase Order application that will be used by the Northwind Traders sales force. You have been assigned the task of completing the user interface and generating some of the underlying code for the Purchase Order application.

To complete this task, you must add a toolbar to the project and assign properties to the buttons of the toolbar, create an event handler that specifies which toolbar button was clicked by a user, and call the appropriate procedure for each button. You must also develop code that creates an instance of a composite control at run time, assigns a context menu to the new control, and adds the new control to the control collection of a **Panel** control that already exists on the main form of the application. To finish this task, you will develop code that handles the click event for a **ContextMenu** control, determines the control that raised the event, and removes that control from a control collection.

**Estimated time to complete this lab: 30 minutes**

# Exercise 1
# Creating and Using Controls

In this exercise, you will begin by opening an existing Windows Forms application. You will add a **ToolBar** control to the main form of the application, add the appropriate number of buttons to the **ToolBar** control, and set values for each of the buttons. After the design for the toolbar is complete, you will develop the code that handles the **ButtonClick** event of the toolbar and responds appropriately for each of the different buttons. You will then develop the code statements required to create an instance of a composite control, associate a context menu with the control, and then add the control to the control collection of a **Panel** control that already exists on the form. You will finish this exercise by creating the event handler for a **ContextMenu** control, developing the code that determines which control raised the event, and removing that control from the controls collection of the **Panel** control to which it belongs. This exercise assesses your knowledge of events and event handlers as well as your ability to use Windows Forms controls and **ContextMenus** at run time.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab02_1\Ex01\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab02_1\Ex01\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

## Scenario

You have been given the task of adding a toolbar to the Purchase Order application and completing some code sections that are used to display purchase item data. You have been given the following table that contains design specification data for the toolbar. You will use this design information to construct the toolbar and the buttons that it contains.

| Item | Item property | Property value |
|---|---|---|
| ToolBar | **Name** | POToolBar |
| POToolBar | **ImageList** | POImageList |
| POToolBar Member 0 | **Name** | RefreshToolBarButton |
| POToolBar Member 1 | **Name** | AddOrderItemToolBarButton |
| POToolBar Member 2 | **Name** | SaveOrderToolBarButton |
| POToolBar Member 3 | **Name** | PrintPreviewToolBarButton |
| POToolBar Member 4 | **Name** | SubmitToolBarButton |
| POToolBar Member 5 | **Name** | ViewUnsubmittedToolBarButton |
| POToolBar Member 6 | **Name** | ViewSubmittedToolBarButton |
| RefreshToolBarButton | **Tag** | Refresh |
| RefreshToolBarButton | **ImageIndex** | 0 |
| RefreshToolBarButton | **ToolTipText** | Refresh product and customer data |
| AddOrderItemToolBarButton | **Tag** | Add |
| AddOrderItemToolBarButton | **ImageIndex** | 1 |
| AddOrderItemToolBarButton | **ToolTipText** | Add a new order item to the purchase order |
| SaveOrderToolBarButton | **Tag** | Save |
| SaveOrderToolBarButton | **ImageIndex** | 2 |
| SaveOrderToolBarButton | **ToolTipText** | Save the current purchase order |

***(continued)***

| Item | Item property | Property value |
|---|---|---|
| PrintPreviewToolBarButton | **Tag** | PrintPreview |
| PrintPreviewToolBarButton | **ImageIndex** | 3 |
| PrintPreviewToolBarButton | **ToolTipText** | Print preview |
| SubmitToolBarButton | **Tag** | Submit |
| SubmitToolBarButton | **ImageIndex** | 4 |
| SubmitToolBarButton | **ToolTipText** | Submit purchase order data to the NWTraders database |
| ViewUnsubmittedToolBarButton | **Tag** | Unsubmitted |
| ViewUnsubmittedToolBarButton | **ImageIndex** | 5 |
| ViewUnsubmittedToolBarButton | **ToolTipText** | View the unsubmitted orders |
| ViewSubmittedToolBarButton | **Tag** | Submitted |
| ViewSubmittedToolBarButton | **ImageIndex** | 6 |
| ViewSubmittedToolBarButton | **ToolTipText** | View report of submitted orders |

After you have created the toolbar, you will develop code for the **ToolBar.ButtonClick** event. The following table identifies the actions that should be taken when each button is clicked.

| Toolbar button name | Action taken when this button is clicked |
|---|---|
| RefreshToolBarButton | DataRefreshMenuItem.PerformClick() |
| AddOrderItemToolBarButton | NewOrderItemButton.PerformClick() |
| SaveOrderToolBarButton | SaveOrderButton.PerformClick() |
| PrintPreviewToolBarButton | PrintPreview() |
| SubmitToolBarButton | DataSubmitMenuItem.PerformClick() |
| ViewUnsubmittedToolBarButton | ViewUnsubmittedOrdersMenuItem.PerformClick() |
| ViewSubmittedToolBarButton | ViewSubmittedOrdersMenuItem.PerformClick() |

After the toolbar is complete, you will develop the code to add and remove an instance of a composite control from the controls collection of a container control (the **Panel** control, **ProductOrderPanel**, has already been added to the main form of the Purchase Order application). Each composite control represents a single purchase item; together, these purchase items make up a purchase order. Although users can add a new purchase item by using either **NewOrderItemButton** or **AddOrderItemToolBarButton**, you will add your code to the event handler for the **NewOrderItemButton.Click** event. You will add code to this event handler that creates an instance of the composite control (**PurchaseOrder.OrderItemControl**), sets the **ContextMenu** property of the new control (the context menu is used by the application user to remove a purchase item from the **Panel** control), and adds the control to the controls collection. To finish up this task, you will create an event handler that responds to a context menu click event, and you will develop the code that removes a purchase item (the purchase item that raised the context menu click event) from the controls collection of **ProductOrderPanel**.

| Tasks | Additional information |
|---|---|
| **1.** Open the Lab02Application.sln file in Visual Studio .NET. The solution file is located in *install_folder*\Labfiles\Lab02_1\Ex01\Starter\Lab02Application. | **a.** For more information about opening a project file and starting an application, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. For additional information about starting an application from Designer, search the Index by using the phrase **Debugging Windows Applications**. |
| **2.** In the Design view, add a **ToolBar** control to MainForm.cs. Configure the toolbar as specified in the scenario. | **a.** For more information about the **ToolBar** control, see the following resources:<br>• Lesson, Using Windows Forms Controls, in Module 2, “Working with Controls,” in Course 2555A*, Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice, Creating and Using a ToolBar Control, in Module 2, “Working with Controls,” in Course 2555A*, Developing Microsoft .NET Applications for Windows (Visual C# .NET).*<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Introduction to the Windows Forms ToolBar Control**. |
| **3.** Use the Task List to locate the code section ‘TODO: create an event handler for the POToolBar.ButtonClick event’, and then create an event handler for the **POToolBar.ButtonClick** event. Develop the code that will invoke the appropriate action when a **ToolBar** button is clicked. | **a.** For more information about the **ToolBar.ButtonClick** event and determining which button on a toolbar caused the **ButtonClick** event to be raised, see the following resources:<br>• Lesson, Using Windows Forms Controls, in Module 2, “Working with Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice, Creating and Using a ToolBar Control, in Module 2, “Working with Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. For additional information about handling the **ButtonClick** event of a **ToolBar** control and to see an example demonstrating a method for determining which button of a toolbar raised the **ButtonClick** event, search by using the phrase **ToolBar.ButtonClick Event**. |

| Tasks | Additional information |
|---|---|
| **4.** Run your application to test the toolbar and the **ButtonClick** event handler. You can position the mouse over the **ToolBar** buttons to display ToolTips and click the buttons to make sure that your **ButtonClick** event handler is working correctly. | **a.** For more information about the purchase order sample application, see the following resources:<br>• Demonstration, Purchase Order Application, in Module 0, "Introduction," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. |
| **5.** Use the Task List to locate the code section 'TODO: create a purchase item', and then create an instance of the **PurchaseOrder.OrderItemControl** control named **tempProductOrder** that uses **ProductContextMenu** as a context menu. | **a.** For more information about using a **ContextMenu** control at run time, see the following resources:<br>• Lesson, Creating Menus, in Module 2, "Working with Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice, Updating Menus at Run Time, in Module 2, "Working with Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Adding Context Menus to Windows Forms**. |
| **6.** Use the Task List to locate the code section 'TODO: add a purchase item to ProductOrderPanel', and then add the new purchase item to the controls collection of the **ProductOrderPanel** control. | **a.** For more information about adding controls to a control collection, see the following resources:<br>• Lesson, Adding Controls at Run Time, in Module 2, "Working with Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice, Adding and Removing Controls at Run Time, in Module 2, "Working with Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Control.Controls Property**. |
| **7.** Use the Task List to locate the code section 'TODO: determine the index number of the control that will be deleted', and then assign the index number of the control that displayed the context menu to a variable named *currentControlIndex*. | **a.** For more information about determining which control displayed a context menu, see the following resources:<br>• Lesson, Creating Menus, in Module 2, "Working with Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice, Updating Menus at Run Time, in Module 2, "Working with Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **ContextMenu.SourceControl**. |

| Tasks | Additional information |
|---|---|
| **8.** Use the Task List to locate the code sections 'TODO: remove a control from the middle of the ProductOrderPanel control collection' and 'TODO: remove a control from the end of the ProductOrderPanel control collection' and then, in each case, create a code statement that will remove the control that displayed the context menu from the controls collection of **ProductOrderPanel**. | **a.** For more information about removing controls from **ControlCollection**, see the following resources:<br>• Lesson, Adding Controls at Run Time, in Module 2, “Working with Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice, Adding and Removing Controls at Run Time, in Module 2, “Working with Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Control.ControlCollection.Remove Method**. |
| **9.** Run your application to test the code that you just created. You should now be able to add and remove purchase items from the controls collection of **ProductOrderPanel**. | Additional information is not necessary for this task. |
| **10.** Save your changes, and then close Visual Studio .NET. | Additional information is not necessary for this task. |

msdn® training

# Module 3: Building Controls

**Contents**

Microsoft®

# Instructor Notes

**Presentation:**
**60 minutes**

**Lab:**
**30 minutes**

This module provides students with an overview of developing and authoring their own controls. In the module, students will learn about the various options for authoring controls. Students learn how to extend existing controls and create new controls. They also learn how to specify property attributes for controls. Finally, they also learn how to license controls.

After completing this module, students will be able to:

- Extend an existing control.
- Create a composite control by combining functionality of several existing Microsoft® .NET Framework Windows Forms controls.
- Describe the design-time support options for components provided by Microsoft Visual Studio® .NET.
- Add attributes that provide information to the Visual Designer.
- Create and validate licenses for controls.

**Required materials**

To teach this module, you need the Microsoft PowerPoint® file 2555A_03.ppt.

**Preparation tasks**

To prepare for this module:

- Read all of the materials for this module.
- Complete the practices, demonstrations, and lab.

# How to Teach This Module

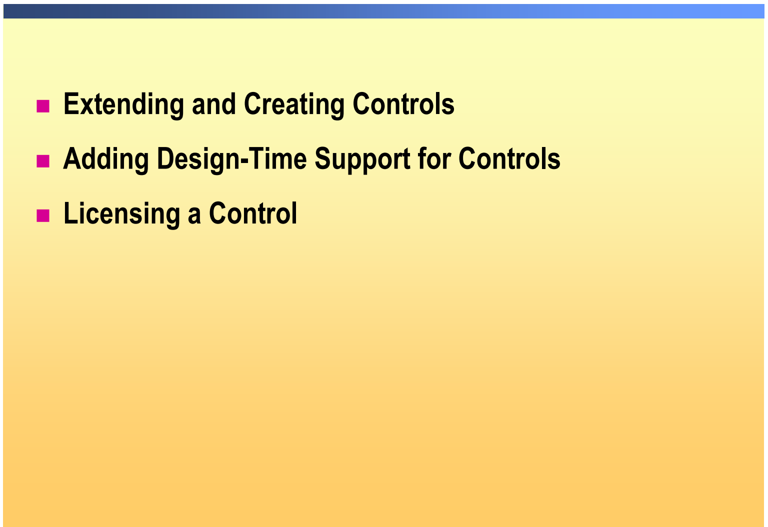
This section contains information that will help you to teach this module. The following are some tips on how to teach this module:

- Lab 3.1, "Building Controls" is based on the Purchase Order application in Course 2555A, *Developing Microsoft .NET Applications for Windows® (Visual C#™ .NET)* and is intended to simulate a real-world environment in which students will demonstrate what they learned during the lecture and practice portions of the module. The lab does not provide step-by-step detailed instructions; instead, the students are given tasks to complete in the left column and a list of resources that they can use (if they need help) in the right column. Students get the hands-on experience that they need by completing the practice activities at the end of each lesson.
- Find out from students if they are interested in learning about licensing controls. It students are not interested in learning about licensing controls, you can skip the lesson, "Licensing a Control."

## Lab: Building Controls

- Make sure that you have demonstrated the two lab applications—the Expense Report application and the Purchase Order application—in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)* before students begin the lab. For more information about how to demonstrate lab scenarios, see the Introduction module in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*
- Practice exercises will enable students to successfully complete the lab exercises. Therefore, make sure that students have completed all practice exercises before they begin the lab.

# Overview



- Extending and Creating Controls
- Adding Design-Time Support for Controls
- Licensing a Control

### Introduction

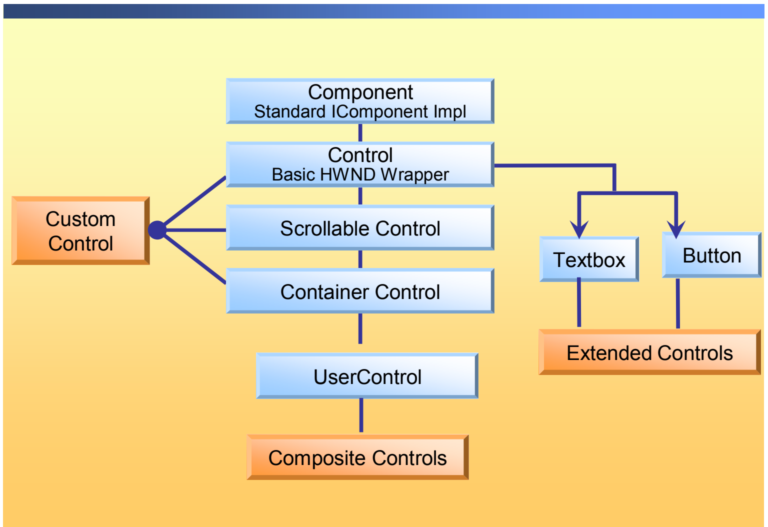
Microsoft® .NET Framework Windows Forms controls are reusable components that encapsulate user interface functionality and are used in client-side Microsoft Windows® applications. Windows Forms provide many ready-to-use controls, but they also provide the infrastructure for developing your own controls.

### Objectives

After completing this module, you will be able to:

- Extend an existing control.
- Create a composite control by combining functionality of several existing Windows Forms controls.
- Describe the design-time support options for components provided by Microsoft Visual Studio® .NET.
- Add attributes that provide information to the Visual Designer.
- Create and validate licenses for controls.

# Lesson: Extending and Creating Controls



**Introduction**

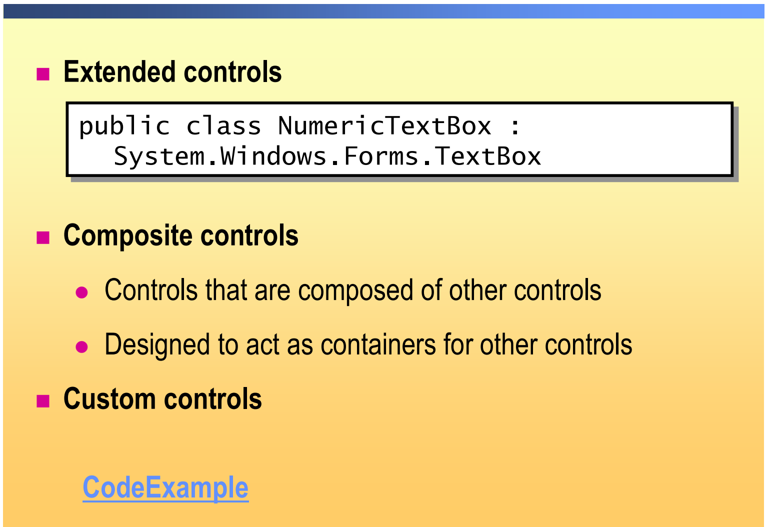
Windows Forms provides several options for authoring controls. You can combine existing controls, extend existing controls, or author your own custom controls. This lesson provides background information about the various options that are available for creating Windows Forms controls.

**Lesson objectives**

After completing this lesson, you will be able to:

- Extend an existing control.
- Create a composite control by combining functionality of several existing Windows Forms controls.
- Describe the design-time support options for components provided by Microsoft Visual Studio .NET.
- Add Attributes that provide information to the Visual Designer.
- Create and validate licenses for controls.

# Options for Building Controls



- **Extended controls**

```
public class NumericTextBox :
  System.Windows.Forms.TextBox
```

- **Composite controls**
  - Controls that are composed of other controls
  - Designed to act as containers for other controls
- **Custom controls**

CodeExample

**Introduction**

One of the options for creating a Windows Forms control is to extend an existing control. You can extend an existing control to customize it or add additional functionality to it. Extend an existing control if the control that you want to create is quite similar in functionality to an existing control.

**Extended controls**

You can customize any Windows Forms control by deriving from it and overriding its properties, methods, and events. This entire procedure of creating a customized control from an existing control is referred to as extending a control. When you inherit from an existing a control, you have the options of using the members of the inherited class, overriding the members of the inherited class, or creating your own custom members.

**Example of extending a control**

Some examples of extended controls include a text box that can accept only numeric values or a button that has some additional properties, such as a property that can record how many times the button has been clicked.

The following code example shows a numeric text box that is inherited from the Windows **TextBox** control. It also shows how the **OnKeyPress** event of the base **TextBox** class is overridden to provide custom functionality.

```
public class NumericTextBox : System.Windows.Forms.TextBox
  protected override void
OnKeyPress(System.Windows.Forms.KeyPressEventArgs e)
```

Because this control inherits from **TextBox**, all the members associated with the **TextBox** control are exposed in the extended control. For example, you can use the functionality of the **Clear()** method of the of **TextBox** control from the extended control.

```
myNumericControl = new NumericTextBox();
myNumericControl.Clear();
```

### Composite controls

Another way of creating your own controls is to combine existing controls to create composite controls. Create composite controls when you need complex functionality that requires the use of more than one control.

You can create new controls by using class composition to combine existing controls. A composite control renders a user interface that reuses the functionality of existing controls. A composite control can synthesize properties from the properties of its child controls and handle events raised by its child controls. It can also expose custom properties and events. All composite controls derive from **System.Windows.Forms.UserControl**. There is full design-time support for creating composite controls with the Visual Studio .NET Windows Forms Designer.

Composite controls can act as containers for other controls because they extend the **ContainerControl** class.

The **ContainerControl** class defines a logical boundary for controls that it contains. This class provides focus management, which means that it is aware of active controls and can change focus from one control to another. It also supports methods for adding, removing, and retrieving child controls. The **Form** class also inherits from the **ContainerControl** class.

For more information about the **ContainerControl** class, see “ContainerControl Class” in the Visual Studio .NET Help documentation.

### Custom controls

If you do not want to combine or extend existing controls, you have the option of creating your own custom controls.

Custom controls display user interface (UI) elements by making calls to a GDI+ **Graphics** object in the **OnPaint** event. Custom controls are generally derived from the base class **System.Windows.Forms.Control**.

```
public class VerticalLabel : System.Windows.Forms.Control
```

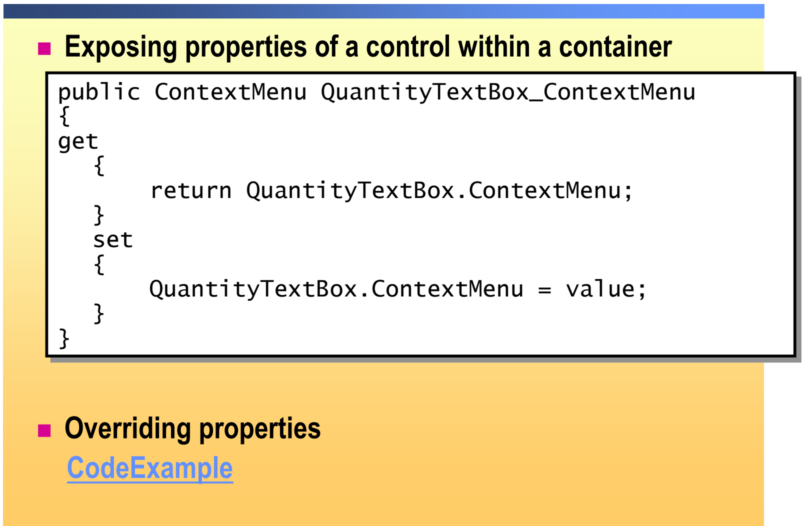
To create a custom control, you generally inherit from the **Control** class, which draws a blank square on your form, override the **OnPaint** event of the **Control** class and use GDI+ to draw your own UI. For more information about using GDI+ in greater detail, see Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C#™ .NET)*.

You can add as well as override properties, methods, and events of the base class (**Control**). The base class provides the plumbing required for visual display in client-side Windows applications. The **Control** class provides a window handle, manages message routing, and provides mouse and keyboard events as well as many other user interface events. It provides advanced layout and has properties specific to visual display, such as **ForeColor**, **BackColor**, **Height**, **Width**, and many others.

### Example of a Custom control

```
using System;
using System.Drawing;
using System.ComponentModel;
using System.Windows.Forms;

public class CustomControl : System.Windows.Forms.Control
{
  protected override void
OnPaint(System.Windows.Forms.PaintEventArgs e)
  {
      e.Graphics.DrawString("Written with GDI+ on OnPaint
event", new Font("Arial",12), new SolidBrush(Color.Red), 0,
0);
  }
}
```

# How to Expose and Override Properties for Controls



- Exposing properties of a control within a container

```
public ContextMenu QuantityTextBox_ContextMenu
{
get
   {
        return QuantityTextBox.ContextMenu;
   }
   set
   {
        QuantityTextBox.ContextMenu = value;
   }
}
```

- Overriding properties

  CodeExample

**Introduction**

A Windows Forms control inherits many properties form the base class **System.Windows.Forms.Control**. These include properties such as **Font**, **ForeColor**, **BackColor**, **Bounds**, **ClientRectangle**, **DisplayRectangle**, **Enabled**, **Focused**, **Height**, **Width**, **Visible**, as well as many others. You can override inherited properties in your control as well as define new properties.

A component should define properties instead of public fields because visual designers such as Visual Studio .NET display properties, but not fields, in the Properties window.

**Property definitions**

Property definitions include two parts: a private variable to hold the value of a property for an instance of a control, and a property definition that exposes the private variable.

While a property definition generally includes a private data member, this is not required. The Get accessor can return a value without accessing a private data member. An example is a property whose get method returns the system time. Properties enable data hiding, in which the accessor methods hide the implementation of the property.

**Procedure: Exposing properties of control in a container**

The properties for controls contained by composite controls are not directly exposed to developers. For example, to access the **Text** property of a text box in a composite control, you may have to iterate through the **Controls** collection of the composite control to access the property. A common practice with composite controls is to expose properties of child controls as properties of the composite control.

The following code example shows how to expose the **ContextMenu** property of a control contained in a composite control.

```
public ContextMenu QuantityTextBox_ContextMenu
{
  get
  {
      return QuantityTextBox.ContextMenu;
  }
  set
  {
      QuantityTextBox.ContextMenu = value;
  }
}
```

The **get** and **set** methods are generally not different from other methods. They can perform any program logic, throw exceptions, be overridden, and be declared with any modifiers allowed by the programming language

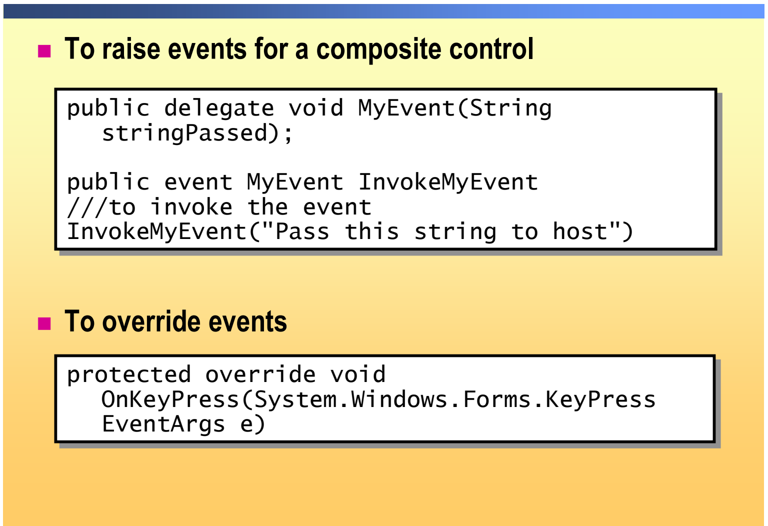
### Procedure: Overriding properties

When you extend an existing control, you can override the properties of the base class to provide custom functionality. The following code overrides the **Text** property of the **TextBox** control and allows only numeric characters to be set on the property.

```
public class NumericTextBox : System.Windows.Forms.TextBox

public override string Text
{
  get
  {
      return base.Text;
  }
  set
  {
      try
      {
          int.Parse(value);
          base.Text = value;
          return;
      }
      catch
      {}
          if (value == null)
          {
              base.Text = value;
              return;
          }
  }
}
```

When overriding members of an extended class, use the **base** keyword to implement the functionality of the base member. The previous example uses **base.Text** to refer to the property in the derived class.

# How to Raise and Override Events for Controls



- To raise events for a composite control

```
public delegate void MyEvent(String
   stringPassed);

public event MyEvent InvokeMyEvent
///to invoke the event
InvokeMyEvent("Pass this string to host")
```

- To override events

```
protected override void
   OnKeyPress(System.Windows.Forms.KeyPress
   EventArgs e)
```

### Introduction

When authoring controls, you can raise your own custom events or override the events of the base class.

### Procedure: Raising events

When creating controls, you use events to notify host applications when something occurs in a control. To create an event, create a delegate  and define any parameters to be passed to the host when the event is invoked. Then declare an event by using the **event** keyword and reference the delegate to invoke and name the event. To invoke the event in the control, call the event and pass the required parameters.

The following code defines an event called **MyEvent** that passes a string data type when invoked.

```
public delegate void MyEvent(String stringPassed);

public event MyEvent InvokeMyEvent;
///to invoke the event
InvokeMyEvent("Pass this string to host");
```

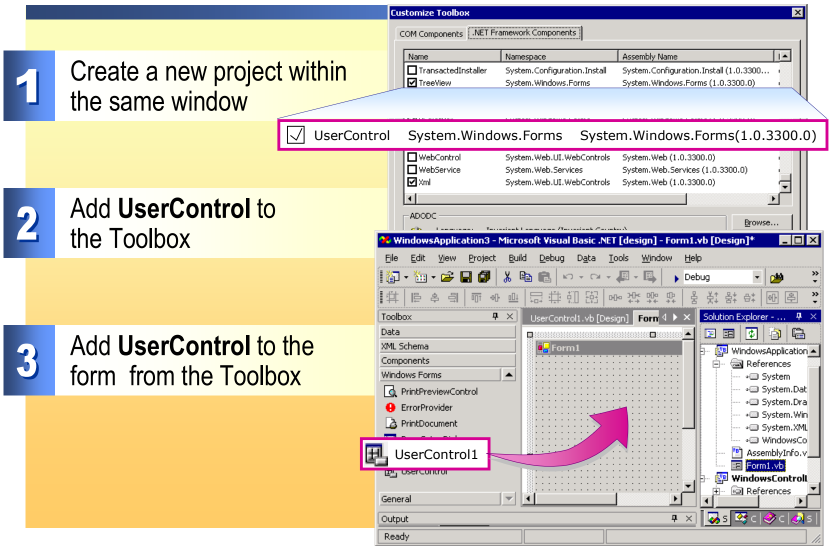
### Procedure: Overriding an event

The **Control** class provides a base set of events that allow you to monitor activities such as property modifications and user actions. When you create controls, you can access events exposed by the **Control** class and other base classes and override them. Each base event provides event-specific information with the *EventArgs* parameter. This parameter (and parameters that derive from it such as *KeyEventArgs* and *MouseEvtsArgs*) provides event specific information, such as which key was pressed or the X,Y position of the mouse cursor.

The following code shows how to override an event. The example overrides the **OnKeyPress** event of a **TextBox** control. The event contains a *KeyPressEventArgs* parameter that contains data about the keys that were pressed (e.KeyChar) and a Boolean value (e.Handled) that allows you to indicate if the event was handled (**True**) or if the event should be passed to Windows for processing (**False**). If the event is handled in the procedure, then it raises a custom event and passes an instance of the object along with the *KeyPressEventArgs* passed to the event.

```
protected override void
OnKeyPress(System.Windows.Forms.KeyPressEventArgs e)
{
  int asciiInteger = Convert.ToInt32(e.KeyChar);
  if (asciiInteger >= 47 && asciiInteger <= 57)
  {
      e.Handled = false;
      return;
  }
  if (asciiInteger == 8)
  {
      e.Handled = false;
      return;
  }
  e.Handled = true;
  if (InvalidUserEntry != null)
      InvalidUserEntry(this, e);
}
```

# How to Test a Control



1. Create a new project within the same window

2. Add **UserControl** to the Toolbox

3. Add **UserControl** to the form from the Toolbox

Customize Toolbox
COM Components
.NET Framework Components
☑ UserControl System.Windows.Forms System.Windows.Forms(1.0.3300.0)
WindowsApplication3 - Microsoft Visual Basic .NET [design] - Form1.vb [Design]*
Toolbox
Windows Forms
UserControl1
Form1
Solution Explorer

## Introduction

The Microsoft .Net Framework eliminates many of the original problems with versioning that existed with COM. For example, you can have more than one version of a component supporting different applications without having to be concerned with backwards compatibility.

When you create and compile a Windows Forms application by using the Visual Studio .NET integrated development environment (IDE), any external assemblies referenced in the application are built into the /Bin folder of the application. Because the application has its own copy of the component, any modification to the original component will not affect it. However, if you are testing a component in an application, you must remove the previous component and add the newer one every time you modify the component.

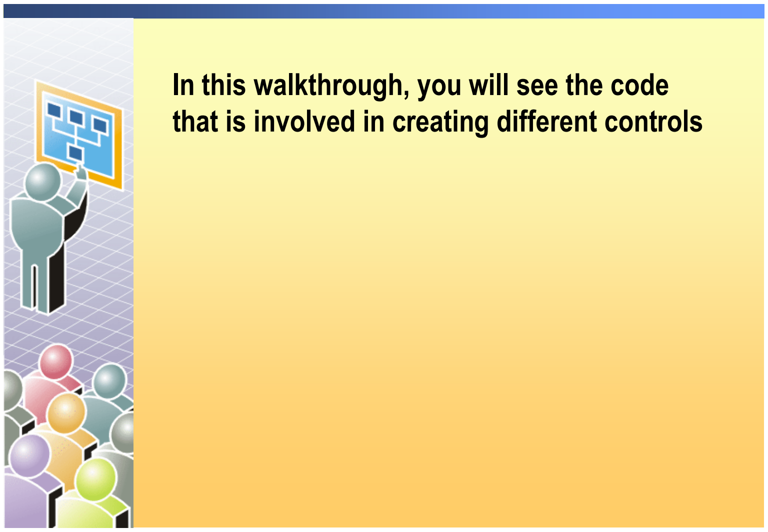
If you want to test a component without adding a new reference to the control every time you modify the control, you can add a forms application to the component solution. The Visual Studio .NET IDE is aware of the component project and adds an implicit reference to the component. This reference is automatically updated whenever you rebuild the component.

## Procedure

1. Create a new form in the same project.
2. Add **UserControl** to the Toolbox.
3. Add the **UserControl** to the form from the Toolbox.

   After you add **UserControl** to the toolbox, it is visible on the Toolbox. To add UserControl to the project, drag the control to the form, just as you do with other controls.

# Code Walkthrough: Creating Controls



In this walkthrough, you will see the code that is involved in creating different controls

### Introduction

In this walkthrough, you will see examples of extended, composite, and custom controls. The instructor will walk you through the code required to extend a control, create a composite control, and build a custom control.

### Instructions

► **Open the Controls.sln solution file**

- Open the Controls.sln solution in Visual Studio .NET from *install_folder*\Democode\ Mod03\Mod03_01\Starter\Controls.

**Note** If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

► **Walkthrough the NumericTextBox control**

1. Open NumericTextBox.cs in the Code Editor.
2. Point out the following line of code. Explain that to create an extended control, you inherit directly from an existing control.

```
public class NumericTextBox : System.Windows.Forms.TextBox
```

3. Scroll to the following line of code. You can override the events of the base control.

```
protected override void
OnKeyPress(System.Windows.Forms.KeyPressEventArgs e)
```

4. Scroll to the following line of code to explain that you can also override the properties of the base control.

```
public override string Text
```

### ► Walkthrough the OrderItemControl

1. Open the Design view of **OrderItemControl.cs**.
2. Composite controls are composed of a **UserControl** and one or more controls. Explain the significance of the controls contained in the **UserControl**.
3. Open OrderItemControl.cs in the Code Editor.
4. A composite control inherits from the **UserControl** class. Point out the following line of code.

```
public class OrderItemControl :
System.Windows.Forms.UserControl
```

5. To access the properties of constituent controls, you must expose them explicitly. Scroll down to the following lines of code to show how the properties of constituent controls are exposed.

```
public int OrderQuantity
{
   get
   {
       return Convert.ToInt32(QuantityNumericTextBox.Text);
   }
   set
   {
       QuantityNumericTextBox.Text = value.ToString();
   }
}
```

6. The **OrderItemControl** exposes a method that binds data to constituent controls. Explain the significance of the following procedure.

```
public void GetProductData(System.Data.DataTable
dsproducts)
```

### ► Walkthrough the VerticalLabel control

1. Open VerticalLabel.cs in the Code Editor.
2. Custom controls generally inherit from the **Control** class. Explain the significance of the following line of code.

```
public class VerticalLabel : System.Windows.Forms.Control
```

3. To implement a custom control you must override the **OnPaint** event and use GDI+ to draw the control. Explain the following procedure to show how to override the **OnPaint** event of the **Control** class.

```
protected override void
OnPaint(System.Windows.Forms.PaintEventArgs e)
```

4. The **Control** class exposes many properties, methods, and events that can be overridden. These properties, methods, and events are shared by all controls. Explain the significance of the following line of procedure.

```
public override string Text
```

### ► Display the controls at design time in the Controls_Host project

1. In Solution Explorer, right-click the **Controls** project, and then click **Build**.
2. In the Controls_Host project, open the Design view of Form1.cs.
3. On the **Tools** menu, click **Customize Toolbox**.
4. In the **Customize Toolbox** dialog box, click the **.NET Frameworks Components** tab, and then click **Browse**.
5. In the **Open** dialog box, select **Controls.dll** from *install_folder*\Democode\Mod03\Mod03_01\Starter\Controls\obj\Debug\, click **Open**, and then click **OK**.
6. Add a **NumericTextBox** to Form1 under the designated label. Attempt to set the **Text** property to a non-numeric value.

   The control enforces the property logic at design time.
7. Add an **OrderItemControl** to Form1 under the designated label.
8. In the Properties window, display the **OrderQuantity** property.

   The **OrderItemControl** property exposes the property of a constituent control.
9. Add a **VerticalLabel** to Form1 under the designated label.
10. Set the **ForeColor** property to **ActiveCaption** and the **Font** property to **Italic**.

    The **OnPaint** event fires at design time as it will at run time.

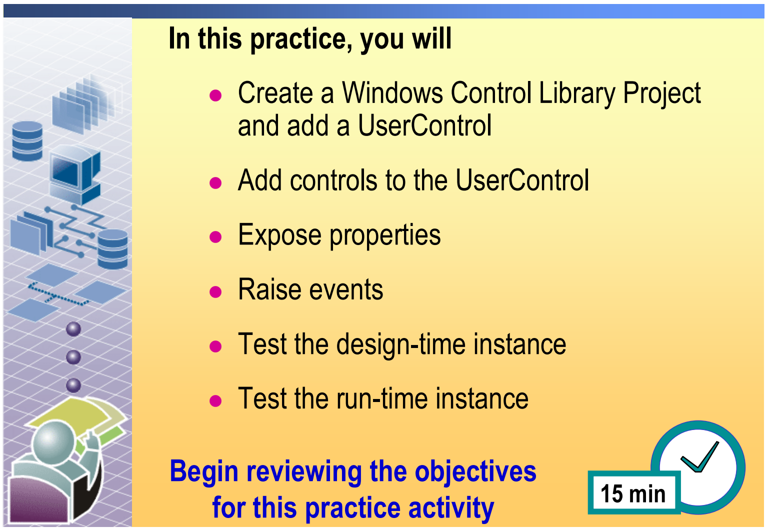
### ► Display the controls at run time in the Controls_Host project

1. In Solution Explorer, right-click the **Controls_Host** project, and then click **Set as Startup Project**.
2. Press F5 to run the application.
3. Attempt to enter non-numeric characters into the **NumericTextBox** control.

   The control enforces property logic at run time.
4. Navigate through the products in the **OrderItemControl**.

   The constituent controls are data bound.

# Practice: Creating a Composite Control



**In this practice, you will**

- Create a Windows Control Library Project and add a UserControl
- Add controls to the UserControl
- Expose properties
- Raise events
- Test the design-time instance
- Test the run-time instance

**Begin reviewing the objectives for this practice activity**

**Introduction**

In this practice, you will build a composite control that displays the current time. The composite control includes a **Label** to display the time and a **Timer** control. The **Timer** control has a **Tick** event that fires repeatedly with a period based on the **Interval** property. The default value of the **Interval** property is 100 milliseconds, meaning that the **Tick** event will fire every second.

You will create a composite control that exposes properties of the **Label** and **Timer** controls and raises the **Tick** event to host applications.

► **Create a Windows Control Library project and add a UserControl**

1. Open Visual Studio .NET.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **New Project** dialog box, click **Windows Control Library**, name it **DigitalClock**, and then click **OK**.
4. Delete **UserControl1** from the project.

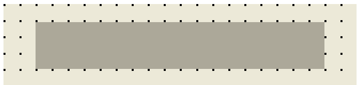
   You can also rename **UserControl1** to **DigitalClock** and skip step 5, however you must remember to change the file name and the class name to **DigitalClock**.
5. On the **Project** menu, click **Add UserControl**, name it **DigitalClock.cs**, and then click **Open**.

### ► Add controls to the UserControl

1. Add the following controls to the **UserControl**.

| Control | Name | Text |
|---|---|---|
| Label | LocalTimeLabel | “” |
| Timer | Timer1 | *N/A* |

2. Set **BackColor** of **LocalTimeLabel** to **ControlDark**.
3. Center **LocalTimeLabel** in the **UserControl**. Your **UserControl** should look like the following.



### ► Add code to expose the properties of LocalTimeLabel and Timer1 to the host application

1. Create a **Public** property named **Timer1_Enabled** that exposes the Boolean **Enabled** property of **Timer1**. Your code should look like the following.

```
public bool Timer1_Enabled
{
   get
   {
       return Timer1.Enabled;
   }
   set
   {
       Timer1.Enabled = value;
   }
}
```

2. Create a **Public** property named **LocalTimeLabel_BackColor** that exposes the **BackColor** property of **LocalTimeLabel**. Your code should look like the following.

```
public Color LocalTimeLabel_BackColor
{
   get
   {
       return LocalTimeLabel.BackColor;
   }
   set
   {
       LocalTimeLabel.BackColor = value;
   }
}
```

### ► Add code that raises the Timer1.Tick event to host applications

1. Declare a **Public** event named **RaiseTimer1_Tick** that passes uses the EventHandler delegate.
2. Create the **Timer1_Tick** event procedure and update the **Text** property of **LocalTimeLabel** to reflect the current time (use the **Now.ToString** function).
3. In the **Timer1_Tick** event procedure, raise the **RaiseTimer1_Tick** event and pass the **sender** and **e** event arguments.

   Your code should look like the following.

```
public event System.EventHandler RaiseTimer1_Tick;

private void Timer1_Tick(object sender, System.EventArgs e)
{
   LocalTimeLabel.Text = System.DateTime.Now.ToString();
   if (RaiseTimer1_Tick != null)
       RaiseTimer1_Tick(sender, e);
}
```

### ► Test the design-time instance of your Composite Control in a host application

1. On the **Build** menu, click **Build Solution** to compile the **DigitalClock** control.
2. On the **File** menu, click **Add Project**, and then click **New Project**.
3. Create a new Windows Application and name it **TestClock**.
4. Add the **DigitalClock** control from the Toolbox to **Form1** in **TestClock**.

**Note** Notice that two new controls are added to the Toolbox: **UserControl1**, which was added with the new Windows Control Library project, and **DigitalClock**.

5. In the Properties window, set the **Timer1_Enabled** property to **True**.

   The **DigitalClock** control is running while the host application is in Design view.
6. In the Properties window, set the **LocalTimeLabel_Color** property to **ControlLight**.

   This property would not be available to developers in the IDE if you did not explicitly expose it by using a property procedure.
7. Add a **Label** to Form1 and name it **UniversalTimeLabel**.

8. Create the **DigitalClock1_RaiseTimer1_Tick** event procedure and update the **Text** property of **UniversalTimeLabel** to reflect the coordinated universal time (use the **Now.UtcNow** function). Your code should look like the following.

```
private void DigitalClock1_RaiseTimer1_Tick(object sender,
System.EventArgs e)
{
   UniversalTimeLabel.Text =
DateTime.Now.ToUniversalTime().ToString();
}
```

9. Add a **Button** to Form1 and name it **StartStopButton**.
10. Create the **StartStopButton_Click** event procedure and write the code to switch the values of the **Timer1_Enabled** property of **DigitalClock1** in the procedure. (If the Timer1_Enabled property is **true** make it **false** and if it is **false** make it **true**.) Your code should look like the following:

```
private void StartStopButton_Click(object sender,
System.EventArgs e)
{
   DigitalClock1.Timer1_Enabled =
!(DigitalClock1.Timer1_Enabled);
}
```

► **Test the run-time instance of your control in a host application**

1. Place a breakpoint on the **Get** and **Set** statements of the **Timer1_Enabled** event procedure in the **DigitalClock** composite control.
2. In Project Explorer, right-click **TestClock** project, and then click **Set as Startup Project**.
3. Press F5 to compile and run the application.
4. Step through each line of the code in the debug mode by pressing F11.
5. Click **StartStopButton** and step through each line of code in debug mode by pressing F11.

   By adding a test host project to the control project, you can easily debug the control. In addition, you will not be required to refresh the reference to the control every time you rebuild it.

**If time permits**

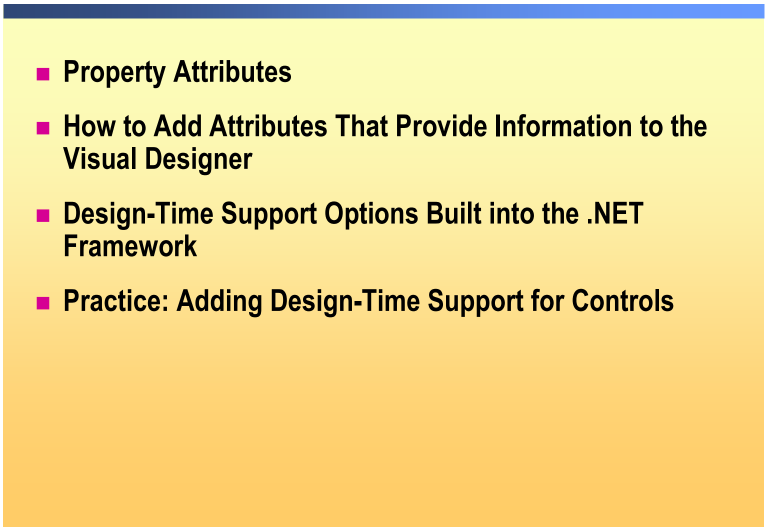
► **Test the DigitalClock in separate instances of Visual Studio .NET**

1. Open a new instance of Visual Studio .NET.
2. On the **File** menu, click **Add Project**, and then click **New Project**.
3. Create a new Windows application and name it **TestClock2**.
4. On the **Tools** menu, click **Customize Toolbox**.
5. Click the **.NET Framework Components** tab, and then click **Browse**.
6. In the **Open** dialog box, navigate to the **DigitalClock.dll** located in the **\bin** directory of the **DigitalClock** project, click **Open**, and then click **OK**.
7. Add a **DigitalClock** control to **Form1**.

8. Switch back to the Visual Studio .NET instance that includes the DigitalClock source code and set the **BackColor** property of **LocalTimeLabel** to **ControlLightLight**.
9. On the **Build** menu, click **Rebuild Solution**.
10. Switch back to the Visual Studio instance that includes the **TestClock2** project and run it by pressing F5.

    The control displays the original value (**ControlDark**) and not the most recent one (**ControlLightLight**). To update the **TestClock2** project to use the most recent **DigitalClock** control, you must refresh the reference to the control.

# Lesson: Adding Design-Time Support for Controls



- Property Attributes
- How to Add Attributes That Provide Information to the Visual Designer
- Design-Time Support Options Built into the .NET Framework
- Practice: Adding Design-Time Support for Controls

**Introduction**

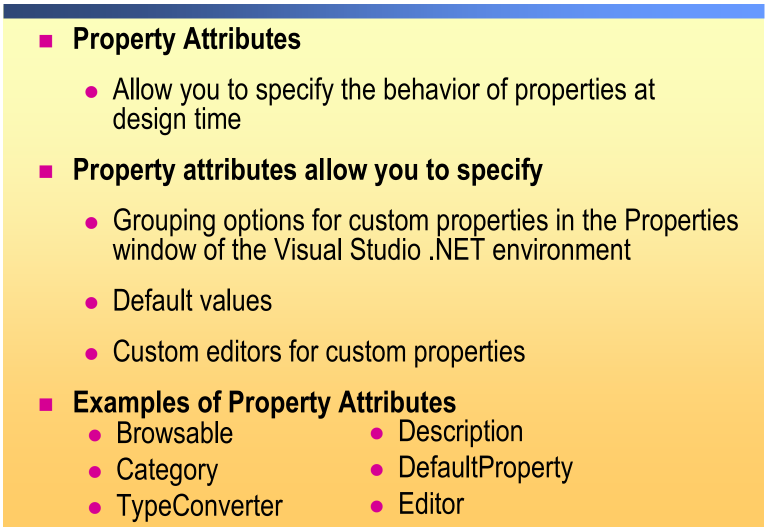
Design-time attributes are essential to display a control and its members correctly at design time because they provide valuable information to a visual design tool. This lesson describes some of the design-time options that are built into the .NET Framework. The lesson also describes how to add the attributes that provide information to the visual design tool.

**Lesson objectives**

After completing this lesson, you will be able to:

- Describe property attributes.
- Add attributes that provide information to the Visual Designer.
- Describe the built-in design-time options for components in Visual Studio .NET.

# Property Attributes



- **Property Attributes**
  - Allow you to specify the behavior of properties at design time
- **Property attributes allow you to specify**
  - Grouping options for custom properties in the Properties window of the Visual Studio .NET environment
  - Default values
  - Custom editors for custom properties
- **Examples of Property Attributes**
  - Browsable
  - Category
  - TypeConverter
  - Description
  - DefaultProperty
  - Editor

### Introduction

Attributes associate design-time functionality with a component. Design-time attributes can be applied to properties, events, classes, and even assemblies.

### Definition

Attributes are the glue that binds design-time functionality to a component. Property attributes allow you to specify the behavior of properties at design time. Attributes can also be applied at the control level and to events as well. In the .NET Framework, design-time functionality is implemented not in a component but outside the component, and it is associated with the component through metadata supplied by custom attributes.

Property attributes allow you to specify:

- Grouping options for custom properties in the Properties window of Visual Studio .NET.
- Default values.
- Custom editors for custom properties.

### List of property attributes

The following table lists some of the property attributes and their descriptions.

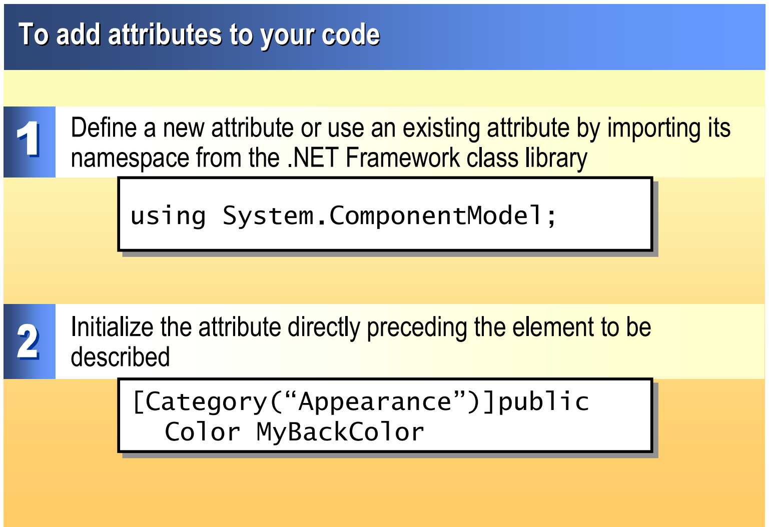
| Property Attributes | Description |
|---|---|
| **Browsable** | Specifies whether a property or an event should be displayed in the Properties window. |
| **Category** | Specifies the name of the category in which to group a property or event. When categories are used, component properties and events can be displayed in logical groupings in the Properties window. |
| **Description** | Defines a small block of text to be displayed at the bottom of the Properties window when the user selects a property or event. |

(***continued***)

| Property Attributes | Description |
|---|---|
| **DefaultProperty** | Specifies the default property for the component. This property is selected in the Properties window when a user clicks the control. |
| **DefaultValue** | Sets a default value for a property. |
| **TypeConverter** | Specifies the type converter to use for converting the type of the property to another data type. |
| **Editor** | Specifies the editor to use for editing (changing) a property in the Visual Designer. |
| **RefreshProperties** | Indicates how a designer refreshes when the associated property value changes. This class cannot be inherited. |

In addition to property attributes, there are some control attributes, such as the **ToolBoxBitMap** attribute, that allow you to specify control behavior. The **ToolBoxBitMap** allows you to specify a bitmap or icon image to represent a control in the Toolbox of the Visual Studio .NET IDE.

# How to Add Attributes That Provide Information to the Visual Designer



To add attributes to your code

1 Define a new attribute or use an existing attribute by importing its namespace from the .NET Framework class library

```
using System.ComponentModel;
```

2 Initialize the attribute directly preceding the element to be described

```
[Category(“Appearance”)]public
  Color MyBackColor
```

****************************ILLEGAL FOR NON-TRAINER USE******************************

**Introduction**

Attributes associate design-time functionality with a component. Design-time attributes can be applied to properties, events, classes, and even assemblies.

**Procedure**

To add attributes:

1. Define a new attribute or use an existing attribute by importing its namespace from the .NET Framework class library.

   ```
   using System.ComponentModel;
   ```

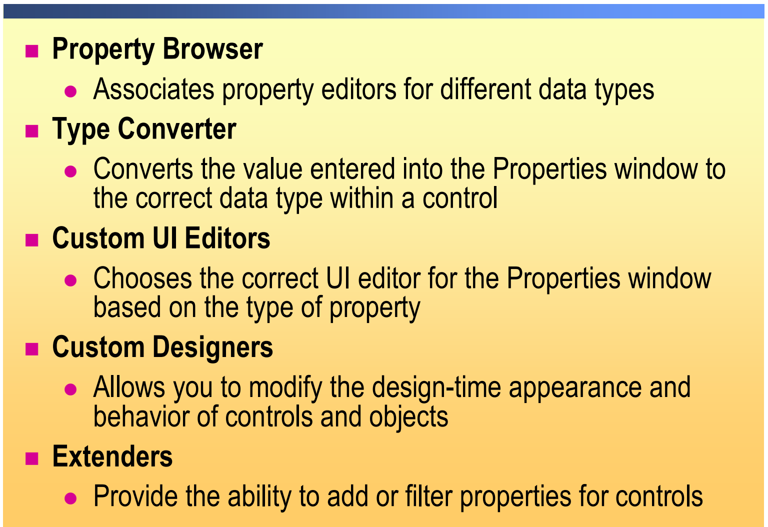
   When you derive a component or control from a base component that has design-time attributes, your component inherits the design-time functionality of the base class. If the base functionality is adequate for your purposes, you do not have to reapply the attributes. However, you can always override attributes or apply additional attributes to the derived component. Only classes that directly or indirectly implement **System.ComponentModel.IComponent** have design-time support in the Visual Designer.

2. Initialize the attribute directly preceding the element to be described.

   In the following code fragment, the **Category** attribute enables the Properties window to display the **Color** property in the Appearance category.

   ```
   [Category(“Appearance”)]public Color MyBackColor
   ```

# Design-Time Support Options Built into the .NET Framework



- **Property Browser**
  - Associates property editors for different data types
- **Type Converter**
  - Converts the value entered into the Properties window to the correct data type within a control
- **Custom UI Editors**
  - Chooses the correct UI editor for the Properties window based on the type of property
- **Custom Designers**
  - Allows you to modify the design-time appearance and behavior of controls and objects
- **Extenders**
  - Provide the ability to add or filter properties for controls

### Introduction

Design-time functionality refers to the display and behavior of a component or control in a visual designer. The .NET Framework includes several features that support the development of design-time behavior for controls.

### Design-time support options

- Property browser

  When a control exposes properties, the Visual Studio .NET IDE displays them at design time in the Properties window. Most properties are modified by typing string values into the property browser. The property browser then converts the string values to the correct type. So, for example, if you expose a property of type integer and attempt to enter a non-numeric value for it in the property browser, it will raise an error.

  The property browser is also capable of associating more advanced property editors for different data types. For example, the property editor exposes a color palette for **Color** types and a **DateTimePicker** control for **Date** types.

- Type converter

  A type converter is used to convert the value entered into the Properties window. A simple example is the conversion from a string to an integer and back. Type converters are primarily used for string-to-value conversions and for validation at design time and at run time. Most native data types (Int32, String, enumeration types, and others) have default type converters. You can create a custom type converter by inheriting from the **System.ComponentModel.TypeConverter** class, if a custom property does not have an associated type converter.

  After you create a type converter, you can apply the **TypeConverter** attribute to a property to associate the property with the type converter. For example, you can use a **TypeConverter** to convert two strings entered in the Properties window to a **Point** type.

- Custom UI editors

  In some situations, a simple value-to-string conversion that allows a property to be displayed as text in the Properties window might not be adequate. For instance, in the case of a color property, a visual representation is more desirable. A UI type editor allows such a representation and is intended for use with property browsers and other advanced design-time hosts.

  To implement a custom UI type editor for Windows Forms, you define a class that is derived from **System.Drawing.Design.UITypeEditor**. You then apply the **Editor** attribute to a property to associate the property with the UI editor.

- Custom Designers

  Designers are classes that allow you to modify the design-time appearance and behavior of components and controls. Although the usual goal of any WYSIWYG form designer is to minimize differences between design-time and run-time appearance, some special design-time cues are necessary. For example, a **System.Windows.Forms.Panel** object might not have a visible border at run time. However, without a border the panel is invisible to the developer designing a form that contains the panel. Therefore, the designer for the **System.Windows.Forms.Panel** object draws a dotted line border around the panel.

  To create a custom designer, create a class that implements the **IDesigner** interface and apply the **Designer** attribute to the control.

- Extenders

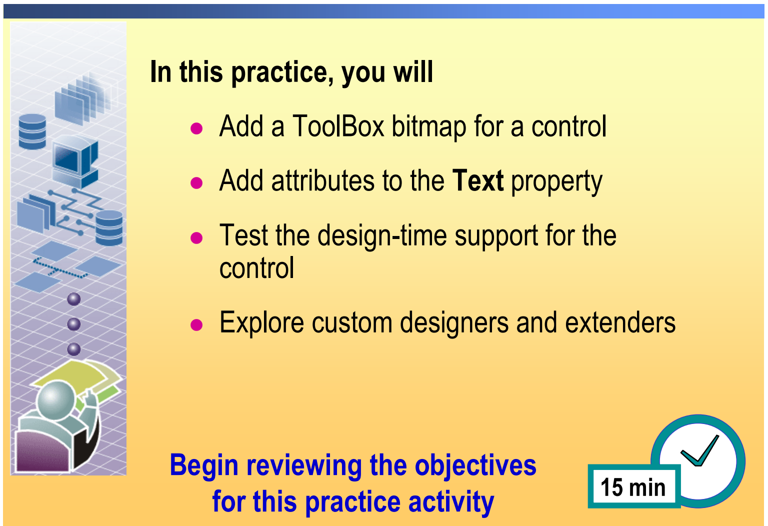
  Extender providers add properties to other controls. In the .NET Framework, extender providers do not require any special support. At design time, extender properties appear in the Properties window as properties on the objects that they extend, rather than on the actual extender object.

  An example of an extender is the **ToolTip** control. The **ToolTip** property is not available to controls on a form until you add the **ToolTip** control to the form.

  To create an extender control, create a class that implements the **System.ComponentModel.IExtenderProvider** interface.

**Note** Controls support two modes of behavior, design-time and run-time. You can use the **DesignMode** property to determine the behavior mode of a control.

# Practice: Adding Design-Time Support for Controls



**In this practice, you will**

- Add a ToolBox bitmap for a control
- Add attributes to the **Text** property
- Test the design-time support for the control
- Explore custom designers and extenders

Begin reviewing the objectives for this practice activity

**Introduction**

In this practice, you will learn how to add design-time support for controls. The exercises involve adding a special icon for a vertical label control on the Toolbox. In addition, you will add an attribute to the **Text** property of the vertical label.

**Instructions**

► **Open the practice project**

1. Use Windows Explorer to browse to *install_folder*\Practices\Mod03\Mod03_02\Starter\VerticalLabel.

> **Note** If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

2. Double-click the **VerticalLabel.sln** solution file to open the project.
3. On the **Build** menu, click **Build Solution**.

► **Add a ToolBox bitmap for a control**

1. On the **Project** menu, click **Add Existing Item**, change the File Type filter to **All Files**, select the **VerticalLabel.ico** file located in *install_folder*\Practices\Mod03\Mod03_02\Starter\VerticalLabel, and then click **Open**.
2. In Solution Explorer, click **VerticalLabel.ico**. In the Properties window, set the **Build Action** property to **Embedded Resource**.

   There are various options for associating an image with a control's ToolBox bitmap. In this exercise, you will embed the image in the compiled project. In this case, the **ToolBoxBitmap** attribute requires two arguments, the **Type** where the image is located and a **String** that represents the name of the image.

3. Open VerticalLabel.cs in the Code Editor.
4. Add a **ToolBoxBitmap** attribute to the **VerticalLabel** class. Use the **typeof** function to return a **VerticalLabel** type for the first argument and **VerticalLabel.ico** as a string for the second argument. Your code should look like the following.

```
[ToolBoxBitmap(typeof(VerticalLabel), "VerticalLabel.ico")]
   public class VerticalLabel :
System.Windows.Forms.Control
```

► **Add attributes to the Text property**

1. Import the **System.ComponentModel** namespace.
2. Add a **Category** attribute to the **Text** property and set it to **VerticalLabel**.
3. Add a **Description** attribute to the **Text** property and set it to **Text is displayed vertically in container**. Your code should look like the following.

```
using System.ComponentModel;
…
[Category("VerticalLabel"), Description("Text is displayed
in container")]
public override string Text
```

4. On the **Build** menu, click **Build Solution**.

► **Test the design-time support for the control**

1. On the **File** menu, click **Add Project**, and then click **New Project**.
2. Click the **Windows Application** template, name it **Test**, and then click **OK**.
3. On the **Tools** menu, click **Customize Toolbox**.
4. In the **Customize Toolbox** dialog box, click the **.NET FrameworksComponents** tab, and then click **Browse**.
5. In the **Open** dialog box, click *install_folder*\Practices\Mod03\Mod03_02\Starter\VerticalLabel\obj\Debug\VerticalLabel.dll, click **Open**, and then click **OK**.
6. Ensure that the control is correctly represented by the VerticalLabel.ico in the Toolbox.
7. Add a **VerticalLabel** control to Form1 from the Toolbox. View the **Text** property in the Properties window to ensure that **Category** and **Description** are displayed.

► **Explore custom designers and extenders**

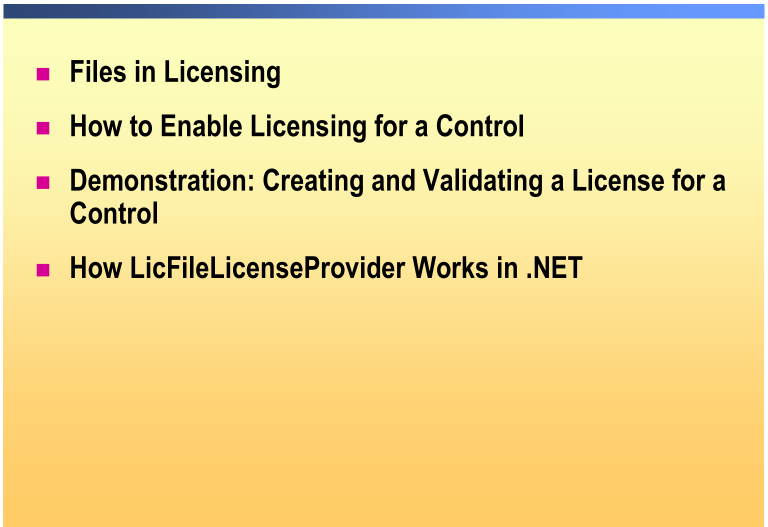
1. Add a **Panel** control to **Form1**.

   The **Panel** control displays an outline of its borders at design time. At run time the outline is not displayed. This is an example of a custom designer.

2. Add a **ToolTip** control to **Form1** and view the **ToolTip on ToolTip1** property that is exposed on both the **Panel** control and the **VerticalLabel** control.

   The **ToolTip** control extends the **ToolTip** property to other controls. This is an example of an extender control.

# Lesson: Licensing a Control



- Files in Licensing
- How to Enable Licensing for a Control
- Demonstration: Creating and Validating a License for a Control
- How LicFileLicenseProvider Works in .NET

### Introduction

The .NET Framework provides a licensing model that is identical for all components including Windows Forms controls and ASP.NET server controls, although the implementation is different.

Licensing allows control authors to protect their property by checking that a user is authorized to use the control. This check is more important at design time, when the control is incorporated into an application, than at run time. When a licensed control is used legally at design time, the application gets a run-time license that can be freely distributed.

### Components of the licensing model

Licensing is built into the .NET Framework, and developers use classes built into the **System.ComponentModel** namespace to provides custom licensing solutions. There are two components of the licensing model: the **LicenseManager** and the **LicenseProvider**.

- The **LicenseManager** class is responsible for validating licensed controls at both run time and design time. The **LicenseManager** class is part of the runtime, and when a class is instantiated, **LicenseManager** uses the proper validation mechanism for the control or component.
- The **LicenseProvider** class is the place to put custom validation code. You can create unique licensing by extending the **System.ComponentModel.LicenseProvider** class. By extending this class, you can create unique licensing models. For example, you can create the license for a control once and allow it to be reused as many times as necessary, or you could create a licensing mechanism that expires after a certain period of time.

  Visual Studio .NET ships with the **LicFile LicenseProvider**, which is derived from the **System.ComponentModel.LicenseProvider** object. The **LicFile LicenseProvider** is built for demonstration purposes and is not intended for production licensing behavior. You can build more advanced licensing scenarios by building additional classes that derive from **System.ComponentModel.LicenseProvider**.

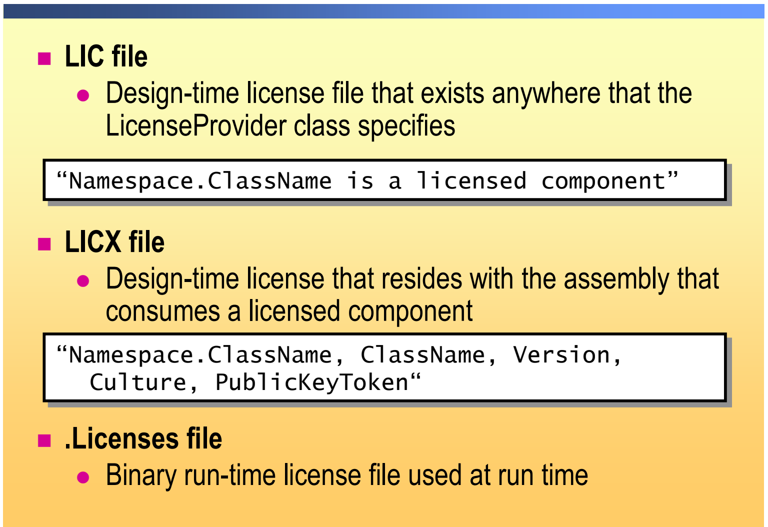
**Note** There are two modes of consumption for licensed controls, design-time and run-time. At design time, a license provider can obtain a valid license from anywhere, such as a .LIC file from a hard disk or from an XML Web service. Then at run time for client apps, this license is converted into a license key and embedded into the executing assembly.

**Lesson objectives**

After completing this lesson, you will be able to:

- Describe the files needed for licensing a control.
- Enable licensing for a control.
- Explain how licensing works in the .NET Framework.

# Files in Licensing



- **LIC file**
  - Design-time license file that exists anywhere that the LicenseProvider class specifies

```
“Namespace.ClassName is a licensed component”
```

- **LICX file**
  - Design-time license that resides with the assembly that consumes a licensed component

```
“Namespace.ClassName, ClassName, Version,
  Culture, PublicKeyToken“
```

- **.Licenses file**
  - Binary run-time license file used at run time

### Introduction

Most controls should support design-time licensing and run-time licensing. Design-time licensing ensures that a developer is building an application with a legally purchased control; run-time licensing ensures that a user is running an application that contains a legally purchased control. There are several licensing options available, such as time-based expiration licensing, per-processor licensing, or per-use licensing.

### Files involved in licensing

Visual Studio .NET uses several files that are required for design-time and run-time licensing.

- .LIC file

  This is the design-time license file. It can exist anywhere that the **LicenseProvider** class specifies. So, for example, the .LIC file can exist with the assembly, somewhere on the hard drive, or it can be retrieved by using a Web service.

  If you use the **LicLicenseFileProvider** file, the .LIC file is a text file that resides in the same folder as the built assembly and has a specific format as shown below.

  ```
  “Namespace.ClassName is a licensed component”
  ```
  Currently you have to manually add this text file to the assembly folder with this correct format and the .LIC extension.

  **Note** If the control is in the same solution project (like when you are testing it) as the project that is using it, the .LIC file must reside in the \obj\Debug folder.

- .LICX file

  This is a design-time file that resides with the assembly that consumes a licensed component. It is a text file that resides in the Project folder of an application. It lists all the licensed components used in an application. The .LICX file has a specific format as shown below.

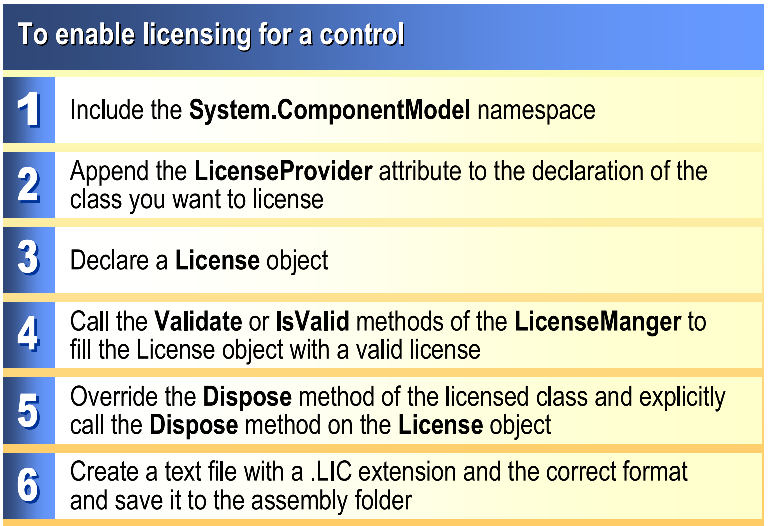
  ```
  "Namespace.ClassName, ClassName, Version, Culture,
  PublicKeyToken"
  ```

  This file is created automatically when you add a licensed component to an application.

- .Licenses file

  This is the binary runtime license file used at run time. This file is created automatically in Visual Studio .NET. It can be generated manually by using the LC.exe utility. This file resides in the same folder as the built assembly.

# How to Enable Licensing for a Control



To enable licensing for a control

1 Include the **System.ComponentModel** namespace

2 Append the **LicenseProvider** attribute to the declaration of the class you want to license

3 Declare a **License** object

4 Call the **Validate** or **IsValid** methods of the **LicenseManger** to fill the License object with a valid license

5 Override the **Dispose** method of the licensed class and explicitly call the **Dispose** method on the **License** object

6 Create a text file with a .LIC extension and the correct format and save it to the assembly folder

### Introduction

To enable licensing for a control, you need the licensing files as well as some licensing objects and attributes such as **LicenseProvider** attribute, a class that derives from **System.ComponentModel.LicenseProvider**, **License** object, and **LicenseManager** object.

### Procedure

To enable licensing for a control:

1. Include the **System.ComponentModel** namespace.

   All of the licensing mechanisms reside in the **System.ComponentModel** namespace.

   ```
   using System.ComponentModel;
   ```

2. Apply a **LicenseProviderAttribute** to the class.

   The granting of licenses and the validation logic is performed by a license provider, which is a class that derives from **System.ComponentModel.LicenseProvider**. When you create a component that you want to license, you must specify the type of **LicenseProvider** by marking the component with a **LicenseProviderAttribute**. Visual Studio .NET supplies the default **LicFileLicenseProvider LicenseProvider** class. This **LicFileLicenseProvider** is easy to use because it is built into the .NET Framework, but its validation logic is not very interesting or secure. You can extend this class to provide more robust licensing scenarios.

   ```
   [LicenseProvider(typeof(LicFileLicenseProvider))] public
   class MyControl
   ```

3. Declare a license object.

   The **License** object is used at run-time to hold the validated license. You use the **LicenseManager** object to fill the **License** object with a validated license.

   ```
   private License validatedLicense = null;
   ```

4. Validate the license by using **LicenseManager.Validate** or **LicenseManager.IsValid** in the constructor of the control.

   **LicenseManager** provides properties and methods to add a license to a component and to manage a **LicenseProvider**. Call **Validate** or **IsValid** in the constructor of the component to validate a license. **Validate** throws a **LicenseException** when it tries to create an instance without a valid license and makes the control unavailable. The **IsValid** method returns a Boolean value that identifies if a valid license was available. **IsValid** does not throw an exception and the control can still be used. You could use this method to modify the behavior of the control based on whether or not a license was available. For example, you could limit features of the control if no license is available.

   ```
   //using the Validate method of the LicenseManager
   private License validatedLicense = null;
   validatedLicense =
   LicenseManager.Validate(typeof(LicensedControl), this);

   //using the IsValid method of LicenseManager
   // private bool validatedLicense ;
   validatedLicense =
   LicenseManager.IsValid(typeof(LicensedControl));
   ```

5. Dispose of any license that is granted when the component is disposed or finalized.

   Call the **Dispose** method when you are finished by using the license. The **Dispose** method leaves the license in an unusable state. After calling **Dispose**, you must release all references to the license so that the memory it was occupying can be reclaimed by garbage collection. This is an important step to take so that your control does not leak licenses. Not doing this might allow malicious code to get the license before the garbage collector does.

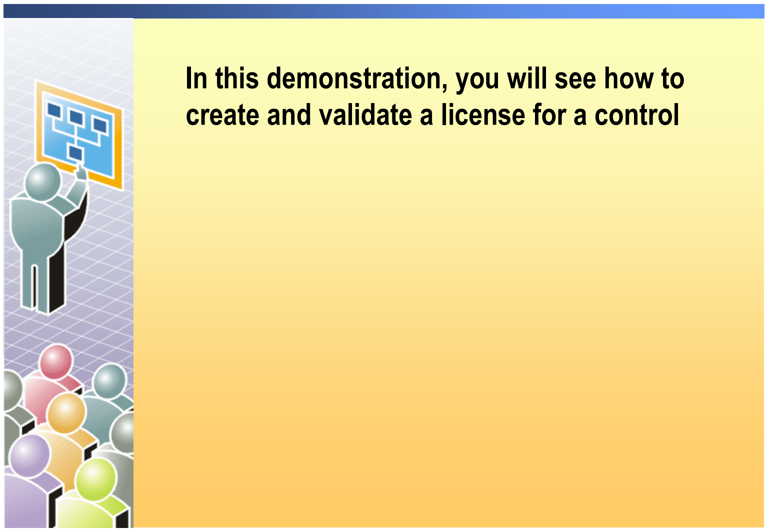
6. Create a text file with the .LIC extension and the correct format and save it to the assembly folder.

**Example**

The following code is a simple example of licensing a Windows Forms control.

```
using System;
using System.ComponentModel;
using System.Windows.Forms;
public class MyControl : Control
{
   private License license = null;
   public MyControl ()
   {
      license = LicenseManager.Validate(typeof(MyControl),
this);
   }
   protected override void Dispose(bool disposing)
   {
      if (disposing)
      {
          if (license != null)
          {
              license.Dispose();
              license = null;
          }
      }
      base.Dispose(disposing);
   }
   ~MyControl()
   {
      Dispose();
   }
}
```

# Demonstration: Creating and Validating a License for a Control



In this demonstration, you will see how to create and validate a license for a control

### Introduction

In this demonstration, you will see how to enable licensing for a control. The demonstration uses the example of a **NumericTextBox** control, which is a **TextBox** that accepts only numeric characters in the **Text** property.

### Instructions

► **Open the NumericTextBox.sln solution file**

- Open the NumericTextBox.sln solution in Visual Studio .NET from *install_folder*\Democode\ Mod03\Mod03_02\Starter.

► **Apply a LicenseProviderAttribute to the NumericTextBox class**

1. Open the NumericTextBox.cs file in the Code Editor and import the **System.ComponentModel** namespace.
2. Append the **LicenseFileProviderAttribute** to the class and use the **LicFileLicenseProvider LicenseProvider**. Your code should look like the following.

```
using System.ComponentModel;

[ToolBoxBitmap(typeof(NumericTextBox),
"NumericTextBox.ico")]
[LicenseProvider(typeof(LicFileLicenseProvider))]
public class NumericTextBox : System.Windows.Forms.TextBox
```

### ► Write the code to create and destroy a license

1. Declare a class level **License** called **validatedLicense**.
2. In the control constructor, use the **Validate** method of the **LicenseManager** to generate a **License** and return it to the *validatedLicense* variable. Your code should look like the following.

```
validatedLicense =
LicenseManager.Validate(typeof(NumericTextBox), this);
```

3. In the control destructor, explicitly destroy the **License** object. Your code should look like the following.

```
if (validatedLicense != null)
{
   validatedLicense.Dispose();
   validatedLicense = null;
}
```

### ► Add the NumericTextBox to the Toolbox and test it

1. On the **Build** menu, click **Build Solution**.
2. On the **File** menu, click **Add Project**, and then click **New Project**.
3. Create a new Windows application and name it **Test**.
4. On the **Tools** menu, click **Customize Toolbox**.
5. Click the **.NET Framework Components** tab, and then click **Browse**.
6. Open **the NumericTextBox.dll** file from *install_folder*\Democode\Mod03\Mod03_02\Starter\bin, and then click **OK**.
7. Attempt to add a **NumericTextBox** to the form. You will get an error stating that a valid license could not be granted.

### ► Add a .LIC license file to the NumericTextBox

1. Expand the **References** folder in the Test Windows application and display the **Path** property of the **NumericTextBox** reference. This is where you will need to create the .lic license file.
2. Open Microsoft Notepad.
3. Type the following code.

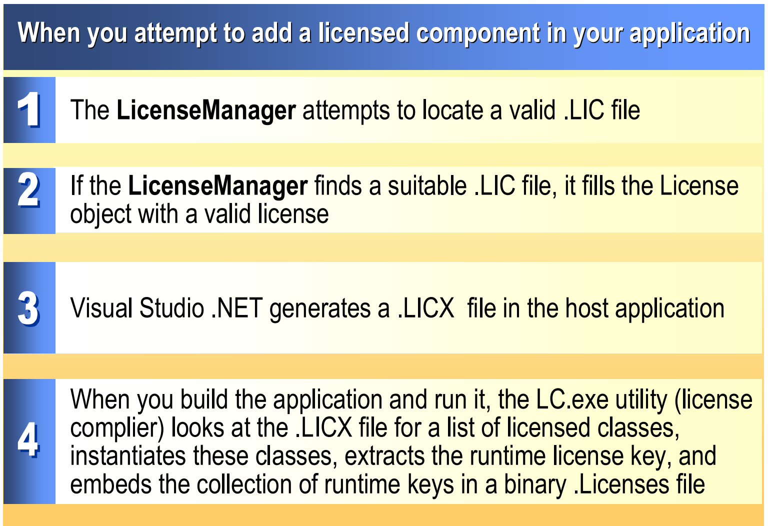
```
NumericTextBox is a licensed component.
```

4. Save the file in *install_folder*\Democode\Mod03\Mod3_02\Starter\obj\Debug, and name it **NumericTextBox.lic**.
5. Close Notepad.

### ► Test the licensed NumericTextBox and examine the supporting files

1. Add a licensed **NumericTextBox** to the form in the Windows application.
2. In Solution Explorer, click **Show All Files**.
3. Open the **Licenses.licx** file and review the syntax.
4. Expand the \bin folder and observe that there is no run-time .license file for the application.
5. Set the Test Windows application as the startup project, and run the application.
6. Close the application.
7. In Solution Explorer, click **Refresh**, and then display the run-time **Text.exe.licenses** file in *install_folder*\Democode\Mod03\Mod3_02\Starter\obj\Debug.

# How LicFileLicenseProvider Works in .NET



When you attempt to add a licensed component in your application

1 The **LicenseManager** attempts to locate a valid .LIC file

2 If the **LicenseManager** finds a suitable .LIC file, it fills the License object with a valid license

3 Visual Studio .NET generates a .LICX file in the host application

4 When you build the application and run it, the LC.exe utility (license complier) looks at the .LICX file for a list of licensed classes, instantiates these classes, extracts the runtime license key, and embeds the collection of runtime keys in a binary .Licenses file

**Introduction**

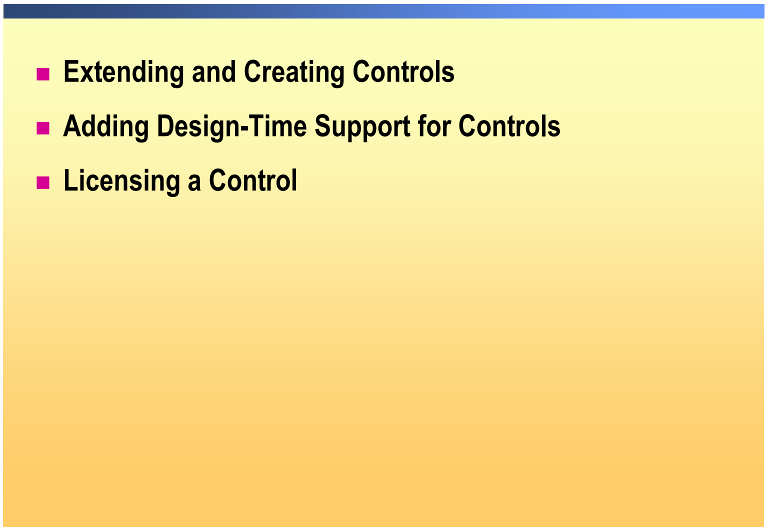
When you attempt to add a licensed component to your application, a series of steps take place in Visual Studio .NET to find a licensed file.

**Procedure**

When you attempt to add a licensed component in your application, the following steps take place in Visual Studio .NET.

1. The **LicenseManager** attempts to locate a valid .LIC file in the same folder as the referenced licensed assembly.
2. If the **LicenseManager** finds a suitable .LIC file, it fills the **License** object with a valid license.
3. Visual Studio .NET generates a .LICX file in the host application that lists attributes of all the licensed controls used by the host application.
4. When you build the application and run it, the LC.exe utility (license complier) looks at the .LICX file for a list of licensed classes, instantiates these classes, extracts the run-time license key, and embeds the collection of run-time keys in a binary .Licenses file.

# Review



- Extending and Creating Controls
- Adding Design-Time Support for Controls
- Licensing a Control

*****************************ILLEGAL FOR NON-TRAINER USE******************************

1. Distinguish between a composite control and a custom control.

   **You combine existing controls to create composite controls. Create composite controls when you need complex functionality that requires the use of more than one control.**

   **If you do not want to combine or extend existing controls, you have the option of creating your own custom controls. Custom controls display user interface (UI) elements by making calls to a GDI+ Graphics object in the OnPaint event. Custom controls are generally derived from the base class System.Windows.Forms.Control.**

2. How do you test a control?

   1. **Create a new form in the same project.**
   2. **Add the control to the Toolbox.**
   3. **Add the control to the form from the Toolbox.**

3. What are property attributes?

   **Attributes are the glue that binds design-time functionality to a component. Property attributes allow you to specify the behavior of properties at design time. Property attributes allow you to specify grouping options for custom properties in the Properties window of the Visual Studio .NET environment, default values for properties, and custom editors for custom properties.**

4. List some property attributes built into Visual Studio .NET.

   **Browsable**

   **Category**

   **Description**

   **DefaultProperty**

   **DefaultValue**

   **TypeConverter**

   **Editor**

   **RefreshProperties**

5. What are extenders?

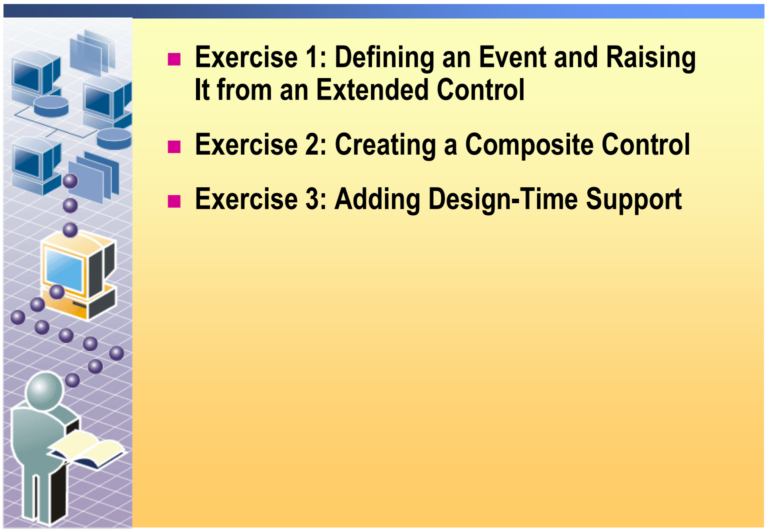
   **Extender providers add properties to other controls. An example of an extender is the ToolTip control. The ToolTip property is not available to controls on a form until you add the ToolTip control to the form.**

6. What is the purpose of LicenseManager and LicenseProvider in licensing controls?

   **The LicenseManager class is responsible for validating licensed controls at both run time and design time.**

   **The LicenseProvider class is the place to put custom validation code.**

# Lab 3.1: Building Controls



- Exercise 1: Defining an Event and Raising It from an Extended Control
- Exercise 2: Creating a Composite Control
- Exercise 3: Adding Design-Time Support

****************************ILLEGAL FOR NON-TRAINER USE******************************

### Objectives

After completing this lab, you will have demonstrated your ability to:

- Declare and raise an event in a custom control to a host application.
- Create a composite control.
- Define property procedures in a composite control to read, write, and format properties of constituent controls.
- Add design-time support to custom controls.
- Test custom controls.

**Note** This lab focuses on the concepts in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. As a result, this lab may not comply with Microsoft security recommendations.

### Prerequisites

Before working on this lab, you must have:

- The knowledge and skills to develop a simple Windows Forms application by using a Visual Studio .NET–compatible programming language.
- The knowledge and skills to use intrinsic controls in a Visual Studio .NET–based application.
- The knowledge and skills to create custom controls in a Visual Studio .NET–based application.
- The knowledge and skills to add design-time support custom controls in a Visual Studio .NET–based application.
- The knowledge and skills to debug a custom control project in a Visual Studio .NET–based application.

### Scenario

You are a developer in a trading company called Northwind Traders. The department you work in is developing a purchase order application that will be used by the Northwind Traders sales force. When developing purchase order applications that cater to different people, you often write the same code (code that has the same functionality) for different applications. Instead of writing the same code repeatedly, you decide to take advantage of component-based development and develop classes and controls that can be reused in other applications.

A common task that you perform in applications is writing code that prevents users from entering non-numeric values in text boxes that display information such as account balances and telephone numbers. You decide to create a group of extended controls that inherit from the **TextBox** class and enforce the required logic. Some may enforce numeric constraints; others may enforce string formatting.

In addition, to increase the efficiency of your development time and provide a consistent interface for the purchase order applications, you decide to create a custom composite control that includes all the constituent controls required to display order information.

### Estimated time to complete this lab: 30 minutes

# Exercise 1
# Defining an Event and Raising It from an Extended Control

In this exercise, you will define an event for the **NumericTextBox** control, raise it from the control, pass event information, and respond to it from a host application.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab03_1\Ex01\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab03_1\Ex01\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

## Scenario

The **NumericTextBox** control is an extended control that inherits from the **TextBox** class. The control accepts only numeric characters and allows the use of the BACKSPACE key. Your users are complaining because the current version of the control does not provide feedback when an invalid character is entered. You have been asked to expose an event on the **NumericTextBox** control that passes event arguments to host applications when an invalid key is pressed. The host applications can then display the source and details of the error in the control to the user.

| Tasks | Additional information |
|---|---|
| **1.** Open Visual Studio .NET, and open the NumericTextBox.sln file. To open the solution file, browse to *install_folder*\Labfiles\Lab03_1\Ex01\Starter\NumericTextBox. | **a.** For more information about opening a project file, see the following resources:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. |
| **2.** Use the Task List in the NumericText.cs file to locate the code section 'TODO: 1. Modify the statement so that the NumericTextBox class inherits from the System.Windows.Forms. TextBox class. | **a.** For more information about creating extended controls and why you should use them, see the following resources:<br>• Lesson: Extending and Creating Controls in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. For additional information about creating extended controls, search by using the phrase **Windows Forms Control Development C#**. |
| **3.** Use the Task List in the NumericText.cs file to locate the code section 'TODO: 2. Declare the **InvalidUserEntry** event that passes an Object type and **KeyPressEventArgs** event arguments. | **a.** For more information declaring an event, see the following resources:<br>• Lesson: Extending and Creating Controls in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Creating a Composite Control in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. For additional information about raising events from custom controls, search by using the phrase **Defining an Event**. |

| Tasks | Additional information |
|---|---|
| **4.** Use the Task List in the NumericText.cs file to locate the code section 'TODO: 3. Raise the **InvalidUserEntry** event, passing the current instance of the NumericTextBox control and the instance of the KeyPressEventArgs event.<br>The **InvalidUserEntry** event will fire every time a user enters an invalid character. | **a.** For more information about how to raise an event, see the following resources:<br>• Lesson: Extending and Creating Controls in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Creating a Composite Control in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. For additional information about raising events from custom controls, search by using the phrase **Defining an Event**. |
| **5.** Rebuild the NumericTextBox project. | **a.** For more information about building your application, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Default and Custom Builds**. |
| **6.** Add a Windows Application project to the NumericTextBox solution, and name it **Test**. | **a.** For more information about adding a project to an existing solution for testing, see the following resources:<br>• Lesson: Extending and Creating Controls in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Creating a Composite Control in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. For help with the code editor, search by using the phrase **Debugging Preparation: Windows Control Libraries**. |
| **7.** Open Design view of Form1 in the Test project, and add the **NumericTextBox** control to the Toolbox. Then, add a **NumericTextBox** control from the Toolbox to Form1.<br>Make sure to reference the version of the control in the Debug folder. | **a.** For more information about customizing the Toolbox, see the following resources:<br>• Code Walkthrough: Creating Controls in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Using the Toolbox**. |

| Tasks | Additional information |
|---|---|
| **8.** Add an event handler to Form1 for the NumericTextBox **InvalidUserEntry** event. In the event handler, use the **KeyPressEventArgs** event arguments to display the value of the invalid key that is pressed in a message box. | **a.** For more information about creating event handlers, see the following resources:<br>• Lesson: Creating an Event Handler for a Control in Module 2, "Working with Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Creating an Event Handler for a Control in Module 2, "Working with Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. Search by using the phrase **Consuming Events**. |
| **9.** Set the Test project as the **Startup Project**, rebuild the project, and then start the NumericTextBox solution. Enter a non-numeric character in the **NumericTextBox** control.<br>The **InvalidUserEntry** event procedure should run and display a message box that includes the value of the invalid key that was pressed. | **a.** For more information about building and debugging your applications, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Default and Custom Builds** and **Using the Debugger**. |

# Exercise 2
# Creating a Composite Control

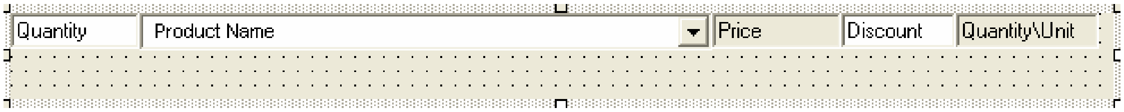
In this exercise, you will create a composite control by using the Windows Control Library template and define properties procedures that read, write, and format the properties of constituent controls. You will then test the composite control in a host application.

## Scenario

As a developer at Northwind Traders, you realize that many applications are being built to enable the sales force to create purchase orders. Rather than create individual controls that offer this functionality in each application, you decide to create a single composite control that contains all the controls required to accomplish this task. You decide to first create the composite control and define all the properties. You will add the data binding code to the control afterward.

The composite control will be called **OrderItemControl**. The **OrderItemControl** control includes four **TextBox** controls and a **ComboBox** control that will be used to display the quantity, product name, price, discount, and quantity/unit of orders.

The following image shows what the **OrderItemControl** control looks like.



There are solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab03_1\Ex02\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open Visual Studio .NET, create a new Windows Control Library project, and name it **OrderItemControl**. Save the project in *install_folder*\Labfiles\Lab03_1\Ex02. | **a.** For more information about creating a project, see the following resources:<br>• Practice: Creating a Composite Control in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Adding Projects and Items to the New Application**. |
| **2.** Add a new UserControl component to the OrderItemControl project, and name it **OrderItemControl.cs**. Delete **UserControl1** from the project.<br>You need to delete the existing **UserControl** from the project or else it will generate errors. | **a.** For more information about adding new items to a project, see the following resources:<br>• Practice: Creating a Composite Control in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Adding Projects and Items to the New Application**. |

| Tasks | Additional information |
|---|---|
| **3.** Add a **TextBox** control to OrderItemControl. Set the **Name** property to **QuantityTextBox**. Use the image of the OrderItemControl shown in the illustration to determine where to place the **QuantityTextBox** control. | **a.** For more information about adding controls to a User Control and configuring properties, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrases **User Control Designer** and **Setting Properties for Controls, Documents, and Forms**. |
| **4.** Add a ComboBox to OrderItemControl. Set the **Name** to **ProductNameComboBox**. Set the **Items** property to **A**, **B**, **C**—placing each value on a separate line. Use the image of the OrderItemControl shown in the illustration to determine where to place **ProductNameComboBox**. | **a.** For more information about adding controls to a User Control, configuring properties, and setting the Items collection of a ComboBox, see the following resources:<br>• Lesson: Extending and Creating Controls in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Creating a Composite Control in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrases **User Control Designer, Setting Properties for Controls, Documents, and Forms** and **String Collection Editor**. |
| **5.** Add a TextBox to OrderItemControl. Set the **Name** to **PriceTextBox**. Set the **Enabled** property to **False**. Use the image of the OrderItemControl shown in the illustration to determine where to place **PriceTextBox**. | Additional information is not necessary for this task. |
| **6.** Add a TextBox to OrderItemControl. Set the **Name** to **DiscountTextBox**. Use the image of the OrderItemControl shown in the illustration to determine where to place **DiscountTextBox**. | Additional information is not necessary for this task. |
| **7.** Add a TextBox to OrderItemControl. Set the **Name** to **QuantityPerUnitTextBox**. Set the **Enabled** property to **False**. Use the image of the OrderItemControl shown in the illustration to determine where to place **QuantityPerUnitTextBox**. | Additional information is not necessary for this task. |

| Tasks | Additional information |
|---|---|
| **8.** In **OrderItemControl**, create a **Public** property named **OrderQuantity** that returns a **String** and gets and sets the **Text** property of **QuantityTextBox**. | **a.** For more information about creating property procedures, see the following resources:<br>• Lesson: Extending and Creating Controls in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Creating a Composite Control in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. Search by using the phrase **Properties Overview**. |
| **9.** In **OrderItemControl**, create a **Public** property named **OrderProductName** that returns a **String** and gets and sets the **Text** property of **ProductNameComboBox**. | **a.** For more information about creating property procedures, see the following resources:<br>• Lesson: Extending and Creating Controls in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Creating a Composite Control in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. Search by using the phrase **Properties Overview**. |
| **10.** In **OrderItemControl**, create a **Public** property named **OrderPrice** that returns a **String** and gets and sets the **Text** property of **PriceTextBox**. In the **Set** block, use the **Format** function to convert **Value** to **Currency**.<br>Using property procedures gives you greater control over how properties are set and retrieved. | **a.** For more information about creating property procedures and using the **Format** function, see the following resources:<br>• Lesson: Extending and Creating Controls in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Creating a Composite Control in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. Search by using the phrases **Properties Overview** and **Format Function**. |
| **11.** In **OrderItemControl**, create a **Public** property named **OrderDiscount** that returns a **String** and gets and sets the **Text** property of **DiscountTextBox**. | **a.** For more information about creating property procedures, see the following resources:<br>• Lesson: Extending and Creating Controls in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Creating a Composite Control in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. Search by using the phrase **Properties Overview**. |

| Tasks | Additional information |
|---|---|
| **12.** In **OrderItemControl**, create a **Public** property named **OrderQuantityPerUnit** that returns a **String** and gets and sets the **Text** property of **QuantityPerUnitTextBox**. | **a.** For more information about creating property procedures, see the following resources:<br>• Lesson: Extending and Creating Controls in Module 3, "Building Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Creating a Composite Control in Module 3, "Building Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Properties Overview**. |
| **13.** Add a Windows Application project to the OrderItemControl solution, and name it **Test**. | **a.** For more information about adding a project to an existing solution for testing, see the following resources:<br>• Lesson: Extending and Creating Controls in Module 3, "Building Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Creating a Composite Control in Module 3, "Building Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. For help with the code editor, search by using the phrase **Debugging Preparation: Windows Control Libraries**.**Control Libraries**. |
| **14.** Open Design view of Form1 in the Test project, and add an **OrderItemControl** control from the Toolbox. Resize Form1 to make the control fit.<br>When you use User Controls to create composite controls, they are automatically added to the Toolbox in the design time environment of the solution. When you develop extended controls and custom controls, you must manually add them to the ToolBox. | **a.** For more information about adding user controls from the Toolbox, see the following resources:<br>• Lesson: Extending and Creating Controls in Module 3, "Building Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Creating a Composite Control in Module 3, "Building Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Using the Toolbox**. |

| Tasks | Additional information |
|---|---|
| **15.** Add a **Button** control to Form1, accept the default name, and in the **Click** event procedure, perform the following steps:<br>• Assign the **OrderItemProduct Name** property of the **OrderItemControl** control to the **OrderItemQuantity** property.<br>• Assign the **OrderDiscount** property of the **OrderItemControl** control to the **OrderPrice** property.<br>• Assign the **OrderDiscount** property of the **OrderItemControl** control to the **OrderQuantityPerUnit** property.<br>This is only to test the **Get** and **Set** statements of the property procedures that you created. | **a.** For more information about creating event handlers, see the following resources:<br>• Lesson: Creating an Event Handler for a Control in Module 2, "Working with Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Creating an Event Handler for a Control in Module 2, "Working with Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. Search by using the phrase **Consuming Events**. |
| **16.** Set a breakpoint on the first line in the **Button1** click event procedure created in the step 15. Set the Test project as **Startup Project**, rebuild the solution, and then run the OrderItemControl solution. Select **B** in the **ProductNameComboBox** control, and type **12** in the **DiscountTextBox** control. Click **Button1**, and view the results. | **a.** For more information about building and debugging your applications, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Default and Custom Builds** and **Using the Debugger**. |

# Exercise 3
# Adding Design-Time Support

In this exercise, you will add attributes to properties of the **OrderItemControl** control. You will then test the design-time support features that you added.

## Scenario

As the developer of the OrderItemControl, you do not know how developers will use it and the kind of support that they will require to be able to use the control. You decide to add some design-time support to the control to enable other developers to use the control easily. You will add description and category information to the properties that you created in the previous exercise so that it is easy for other developers to understand the function of each property.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab03_1\Ex03\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab03_1\Ex03\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open Visual Studio .NET, and open the OrderItemControl.sln file. To open the solution files, browse to *install_folder*\Labfiles\ Lab03_1\Ex03\Starter\ OrderItemControl. | **a.** For more information about opening a project file and starting an application, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. |
| **2.** Add **Category** and **Description** attributes to each property of the OrderItemControl. Use **OrderItemControl** as the **Category**. | **a.** For more information about adding design-time support to control properties, see the following resources:<br>• Lesson: Adding Design-Time Support for Controls in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Adding Design-Time Support for Controls in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Attributes and Design-Time Support**. |

| Tasks | Additional information |
|---|---|
| **3.** Build the OrderItemControl project. Add a Windows Application project to the OrderItemControl solution, and name it **Test**. Add an OrderItemControl from the ToolBox, and view the custom properties in the Properties window. Make sure to enable the **Categorized** button so that you can see how your control properties are sorted. | **a.** For more information about building and debugging your applications, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Default and Custom Builds** and **Using the Debugger**. |

msdn® training

# Module 4: Using Data in Windows Forms Applications

**Contents**

Microsoft®

# Instructor Notes

**Presentation:**
**180 minutes**

**Labs:**
**60 minutes**

The module provides an overview of how to use data in a Windows Forms application. Windows Forms are a part of the new Microsoft® .NET Framework. The module explains how to use the Microsoft ADO.NET object to access, display, and update data from a database. In this module, students also learn how to create and test a simple XML Web service client application and how to persist application settings.

After completing this module, students will be able to:

- Describe the objects in the ADO.NET object model.
- Add and configure ADO.NET objects in a Windows Forms application.
- Access and modify data from a database by using DataSets.
- Bind data to controls.
- Describe the XML Web services model and the roles of HTML, Simple Object Access Protocol (SOAP), and XML in the Web services model.
- Create and test a simple XML Web service client application.
- Persist data to files, serialize objects, use isolated storage, and persist application settings.

**Required materials**

To teach this module, you need the Microsoft PowerPoint® file 2555A_04.ppt.

**Preparation tasks**

To prepare for this module:

- Read all of the materials for this module.
- Review the animation for this module.
- Complete the demonstrations, practices, and labs.

# How to Teach This Module

This section contains information that will help you to teach this module.

- If students are interested in referencing code in other languages, point them to “Language Equivalents” in the Help documentation for the Microsoft Visual Studio® .NET development system. This section provides examples in languages such Microsoft Visual Basic® .NET, Microsoft Visual C#™, and Java.
- Lab 4.1: Accessing Data by Using ADO.NET is based on the Purchase Order application in Course 2555A, *Developing Microsoft .NET Applications for Windows® (Visual C# .NET)* and is intended to simulate a real-world environment in which students will demonstrate what they learned during the lecture and practice portions of the module. The lab does not provide step-by-step detailed instructions; instead, the students are given tasks to complete in the left column and a list of resources that they can use (if they need help) in the right column. Students get the hands-on experience that they need by completing the practice activities at the end of each lesson.
- Lab 4.2: Calling an XML Web Service is based on the Expense Report application in Course 2555A, *Developing Microsoft .NET Applications for Windows® (Visual C# .NET)*.

# Lesson: Overview of XML Web Services

This section describes the instructional methods for teaching this lesson.

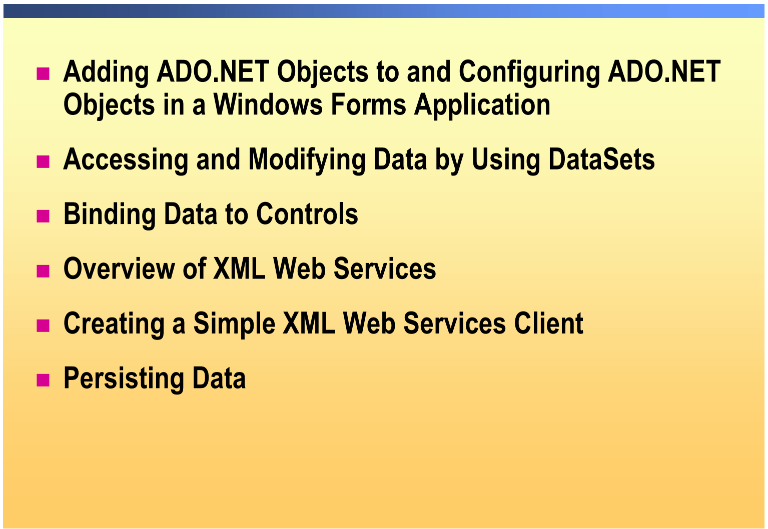
This is an overview lesson that does not cover implementation details. Implementation details are included in the Creating a Simple XML Web Services Client lesson.

# Lesson: Creating a Simple XML Web Services Client

This section describes the instructional methods for teaching this lesson.

Ask students if they have ever created a simple XML Web service. If they have not, you may want to do an informal demonstration of creating a simple XML Web service so they can see how easy it is.

# Overview



- Adding ADO.NET Objects to and Configuring ADO.NET Objects in a Windows Forms Application
- Accessing and Modifying Data by Using DataSets
- Binding Data to Controls
- Overview of XML Web Services
- Creating a Simple XML Web Services Client
- Persisting Data

****************************ILLEGAL FOR NON-TRAINER USE******************************

**Introduction**

Data binding provides a simple, convenient, powerful, and transparent way for developers to create a read/write link between the controls on a form and the data in their applications.

Windows Forms, which are part of the new Microsoft® .NET Framework, support binding data to Microsoft ADO.NET DataSets, arrays, collections, and other controls. A control can be bound to any collection that supports indexed access to the elements in that collection.

XML Web services enable the exchange of data and the remote invocation of application logic by using XML messaging to move data through firewalls and among many heterogeneous systems.

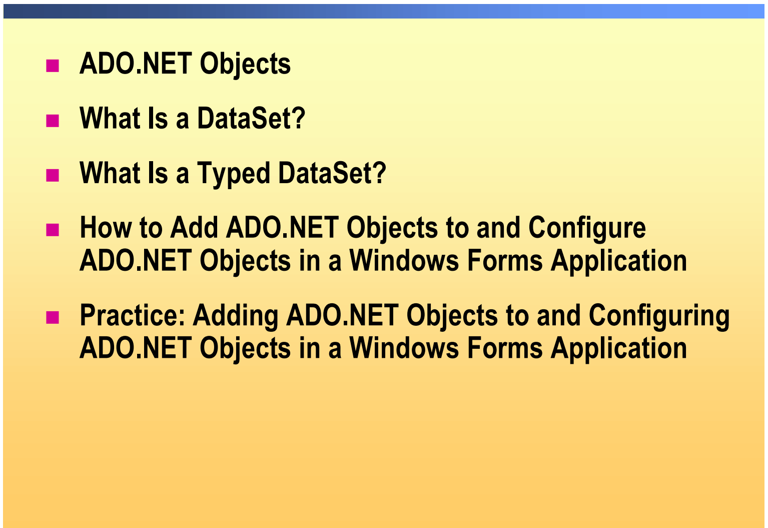
Developers can create applications that weave together XML Web services from a variety of sources in much the same way that developers traditionally use components when creating a distributed application.

**Objectives**

After completing this module, you will be able to:

- Describe the objects in the ADO.NET object model.
- Add and configure ADO.NET objects in a Windows Forms application.
- Access and modify data from a database by using DataSets.
- Bind data to controls.
- Describe the XML Web services model and the roles of HTML, SOAP, and XML in the XML Web services model.
- Create and test a simple XML Web service client application.
- Persist data to files, serialize objects, use isolated storage, and persist application settings.

# Lesson: Adding ADO.NET Objects to and Configuring ADO.NET Objects in a Windows Forms Application



- ADO.NET Objects
- What Is a DataSet?
- What Is a Typed DataSet?
- How to Add ADO.NET Objects to and Configure ADO.NET Objects in a Windows Forms Application
- Practice: Adding ADO.NET Objects to and Configuring ADO.NET Objects in a Windows Forms Application

### Introduction

As application development has evolved, a lot of new applications are based on the Web application model. An increasing number of applications use XML to encode data to be passed over network connections. ADO.NET provides a programming model that incorporates features of both XML and ADO.NET in the .NET Framework.

ADO.NET is a set of classes that allow .NET-based applications to read and update information in databases and other data stores. You can access these classes through the .NET Framework **System.Data** namespace.

ADO.NET provides consistent access to a wide variety of data sources, including Microsoft SQL Server™ databases, OLE DB–compliant databases, non-relational sources such as Microsoft Exchange Server, and XML documents.

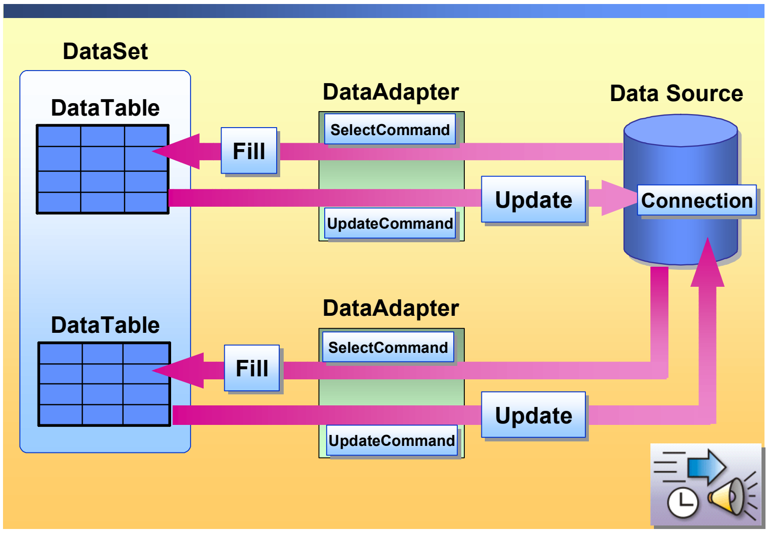
ADO.NET is designed for working with disconnected data in a multi-tier environment. ADO.NET uses XML as the format for transmitting disconnected data, which makes it easier to communicate with client applications that are not based on Windows Forms.

### Lesson objectives

After completing this lesson, you will be able to:

- Describe the objects in the ADO.NET object model.
- Describe datasets.
- Describe typed datasets.
- Add ADO.NET objects and configure ADO.NET objects in a Windows Forms application.

# ADO.NET Objects



**Introduction**

ADO.NET evolved from the ADO data access model. By using ADO.NET, you can develop applications that are robust and scalable and that can use XML.

ADO.NET has some of the same objects as ADO (like the **Connection** and **Command** objects), and introduces new objects, such as **Dataset**, **DataReader**, and **DataAdapter**.

**Connection object**

To move data between a data store and your application, you must first have a connection to the data store. In ADO.NET you can create and manage a connection by using a **Connection** object. Applications use **Connection** objects to communicate with databases. The **ConnectionString** property determines connection settings. These connection settings access a particular **DataSource**. A typical **ConnectionString** property might look like the following code:

```
Provider=SQLOLEDB.1;Data Source=MySQLServer;Initial
Catalog=NORTHWIND;Integrated Security=SSPI
```

There are two kinds of connection objects in ADO.NET: **SqlConnection** and **OleDbConnection**. The **SqlConnection** object manages a connection to Microsoft SQL Server version 7.0 or later. The **SqlConnection** object is optimized for use with SQL Server 7.0 or later by bypassing the OLE DB layer. The **OleDbConnection** object manages a connection to any data store accessible through OLE DB. The **OleDbConnection** object interacts with OLE DB to expose a consistent API for a variety of data sources—everything from simple text files to spreadsheets and full-featured databases.

**Note** The **Connection** object is one component of a .NET Framework data provider. A data provider in the .NET Framework serves as a bridge between an application and a data source and is used to retrieve data from a data source and to reconcile changes to that data back to the data source.

### Command object

You can use **Command** objects to access data directly in the database in a connected environment. **Command** objects use SQL statements or stored procedures to retrieve data. Commands travel across connections, and result sets are returned in the form of streams that can be read by **DataReaders** or pushed into **DataSet** objects. **Command** objects contain a **Parameters** collection that populates the input and output arguments of SQL statements or stored procedures. For example, if you have a SQL statement that returns all of the rows in the **Orders** table where the **EmployeeID** is equal to a value determined at run time, you add the value **EmployeeID** to the **Parameters** collection of the **Command** object.

A **Command** object contains a reference to a SQL statement or stored procedure that you can execute directly. The two **Command** classes are described in the following table.

| Command class | Description |
|---|---|
| System.Data.SqlClient.SqlCommand | SQL Server .NET Data Provider command |
| System.Data.OleDb.OleDbCommand | OLE DB .NET Data Provider command |

### DataReader object

The **DataReader** is a fast, forward-only cursor that loops through a stream of rows. When you execute a **Command** object that returns a set of rows, you use a **DataReader** to loop through the set of rows. You can use a **Command** object and the **ExecuteReader** method to return a **DataReader**. You can execute any **SELECT** statement or a stored procedure that contains a **SELECT** statement.

The **DataReader** provides strongly typed methods to get the value of a specific column in the current row. You can also obtain metadata about the rows, such as the column name and the column data type.

When you process a result set with a **DataReader**, the associated connection is kept busy until you close the **DataReader**. For this reason, you should close the **DataReader** as soon as you finish processing the result set.

### DataSet object

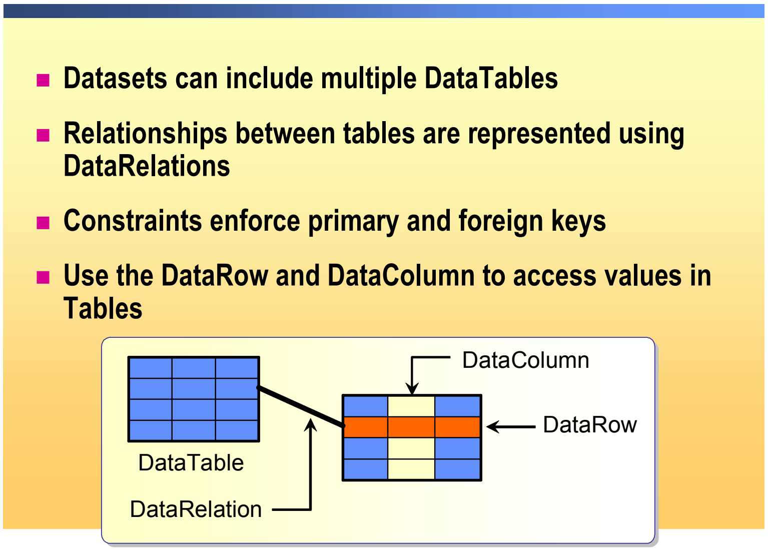
Datasets store data in a disconnected cache. The structure of a dataset is similar to that of a relational database; it exposes a hierarchical object model of tables, rows, and columns. In addition, it contains constraints and relationships defined for the dataset.

### DataAdapter object

The **DataSet** object represents a local copy of data from a data source and is one of the key innovations of the Microsoft .NET Framework. By itself, the **DataSet** object is useful for reference. However, to serve as a true data-management tool, a **DataSet** must be able to interact with a data source. To accomplish this, the .NET Framework provides the **DataAdapter** class. A **DataAdapter** object serves as a bridge between a **DataSet** and a data source for retrieving and saving data. The **DataAdapter** class represents a set of database commands and a database connection that you use to fill a **DataSet** and update the data source. **DataAdapter** objects are part of the Microsoft ADO.NET data providers, which also include connection objects, data-reader objects, and command objects. Microsoft Visual Studio® .NET makes two primary **DataAdapters** available for use with databases. In addition, other **DataAdapters** can be integrated with Visual Studio. The primary **DataDdapters** are:

- **OleDbDataAdapter**, which is suitable for use with any data source that is exposed by an OLE DB provider.
- **SqlDataAdapter**, which is specific to a SQL Server version 7.0 or later database. The **SqlDataAdapter** is faster than the **OleDbDataAdapter** because it works directly with SQL Server and does not go through an OLE DB layer.

# What Is a DataSet?



- Datasets can include multiple DataTables
- Relationships between tables are represented using DataRelations
- Constraints enforce primary and foreign keys
- Use the DataRow and DataColumn to access values in Tables

****************************ILLEGAL FOR NON-TRAINER USE******************************

### Introduction

In ADO.NET, by using DataSets you can represent data in a local cache and provide a relational programming model for the data regardless of its source.

### Definition

The ADO.NET DataSet is an in-memory cache of data that functions as a disconnected relational view of the data. The connection to the data source does not need to be active for an application to view and manipulate data in a DataSet. This disconnected architecture enables greater scalability by using database server resources only when reading from, or writing to, the data source.

DataSets store data similarly to the way data is stored in a relational database, with a hierarchical object model of tables, rows, and columns. Additionally, you can define constraints and relationships for the data in the DataSet.

### DataTables in DataSets

**DataTable** objects represent the tables in a DataSet. A **DataTable** represents one table of in-memory relational data. The data is local to the .NET-based application in which it resides, but it can be populated from an existing data source. A **DataTable** is composed of **DataColumns**.

### DataColumns in DataTables

A DataColumn is the building block for creating the schema of a **DataTable**. Each **DataColumn** has a **DataType** property that determines the kind of data that each **DataColumn** contains. For example, you can restrict the data type to integers, strings, or decimals. Because data contained in the **DataTable** is typically merged back into the original data source, you must match the data types to those in the data source.

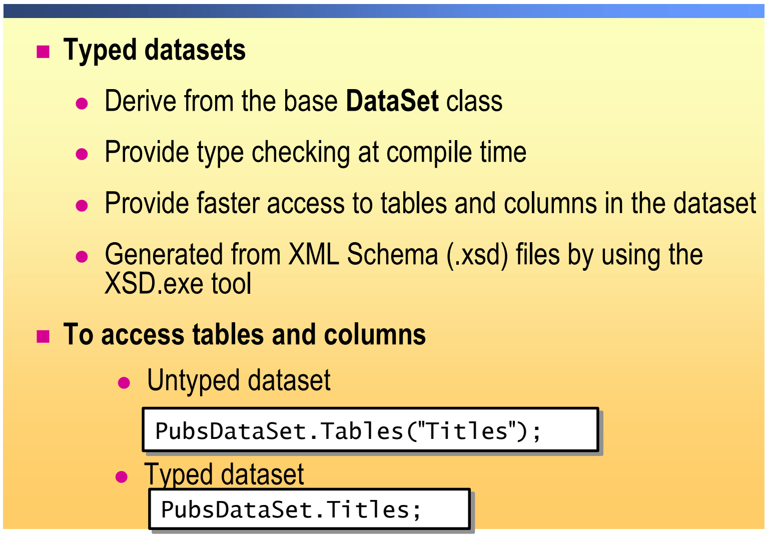
### Tables in a DataSet

The **DataSet** class has a **Tables** property that gets a collection of **DataTable** objects in the DataSet, and a **Relations** property that gets a collection of the **DataRelation** objects in the DataSet.

A **DataTable** object contains several collections that describe the data in the table and cache the data in memory. The following table describes the most important collections.

| Collection name | Type of object in collection | Description of object in collection |
|---|---|---|
| Columns | DataColumn | Contains metadata about a column in the table, such as the column name, data type, and whether rows can contain a NULL value in this column. |
| Rows | DataRow | Contains a row of data in the table. A **DataRow** object also maintains the original data in the row, before any changes were made by the application. |
| Constraints | Constraint | Represents a constraint on one or more **DataColumn** objects. **Constraint** is an abstract class. There are two concrete subclasses: **UniqueConstraint** and **ForeignKeyConstraint**. |
| ChildRelations | DataRelation | Represents a relationship to a column in another table in the DataSet. You use **DataRelation** objects to create links between primary keys and foreign keys in your tables. |

# What Is a Typed DataSet?



- **Typed datasets**
  - Derive from the base **DataSet** class
  - Provide type checking at compile time
  - Provide faster access to tables and columns in the dataset
  - Generated from XML Schema (.xsd) files by using the XSD.exe tool
- **To access tables and columns**
  - Untyped dataset

```
PubsDataSet.Tables("Titles");
```

  - Typed dataset

```
PubsDataSet.Titles;
```

**Introduction**

Datasets can be typed or untyped. Typed datasets are classes that are generated from XML Schema (.xsd) files. An untyped dataset, in contrast, has no corresponding built-in schema.

**Definition**

A typed dataset is a class that derives from the base **DataSet** class. It inherits all the methods, events, and properties of a dataset. In addition, a typed dataset provides strongly typed methods, events, and properties. This means that you can access tables and columns by name, instead of by using collection-based methods. For example, to access the Titles table from a dataset named pubsDataSet1, you use the following code:

```
PubsDataSet.Tables("Titles");
```

However, with a typed dataset, you can directly access the Titles table by using the following code:

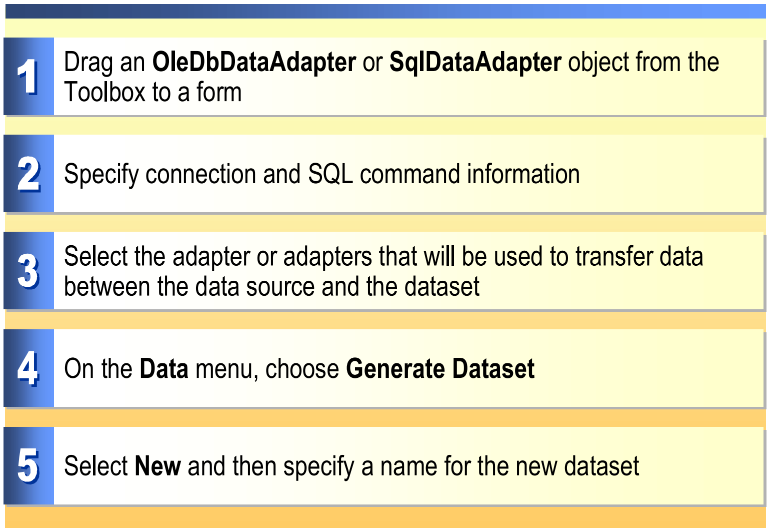
```
PubsDataSet.Titles;
```

**Advantages of typed datasets**

Typed datasets are not only easier to read, but they are also fully supported by the Microsoft IntelliSense® technology in the Code Editor in Visual Studio .NET. In addition to being easier to work with, the syntax for the typed dataset provides type checking at compile time; this greatly reduces the possibility of errors in assigning values to dataset members. Access to tables and columns in a typed dataset is also slightly faster at run time because access is determined at compile time—not through collections at run time.

You can generate a strongly typed dataset from within the Visual Studio .NET integrated development environment (IDE) by selecting tables from an existing database or by creating one using the XML Designer.

# How to Add ADO.NET Objects to and Configure ADO.NET Objects in a Windows Forms Application



1 Drag an **OleDbDataAdapter** or **SqlDataAdapter** object from the Toolbox to a form

2 Specify connection and SQL command information

3 Select the adapter or adapters that will be used to transfer data between the data source and the dataset

4 On the **Data** menu, choose **Generate Dataset**

5 Select **New** and then specify a name for the new dataset

**Introduction**

The data design tools in Visual Studio .NET provide a simple way to connect to a database, retrieve data from it, and generate a typed dataset.

You can use either of the following data design tools to connect to a data source:

- DataAdapter Configuration Wizard

  This wizard prompts you for information to create a connection that is in turn linked to a data adapter.

- Data Form Wizard

  This wizard creates the connection object as part of the form that it is configuring.

This topic explains how to connect to a data source by using DataAdapter Configuration Wizard.

**Note** When you use data design tools, you need not create explicit connections to a data source; however, there are times when you need to create just a connection. For more information about how to create a connection, in the Visual Studio .NET documentation, search by using the phase **Creating ADO.NET Connection Objects**.

### Procedure: Adding ADO.NET objects

Data Adapter Configuration Wizard helps you set the properties of a new or existing data adapter. A data adapter contains SQL commands or stored procedures that your application can use to read data into a dataset from a database and write it back again. The wizard can also create a data connection that allows the adapter to communicate with a database.

To use DataAdapter Configuration Wizard:

1. Drag an **OleDbDataAdapter** or **SqlDataAdapter** object from the Toolbox onto a form or component.
2. Specify connection and SQL command information.

   The wizard displays several dialog boxes:

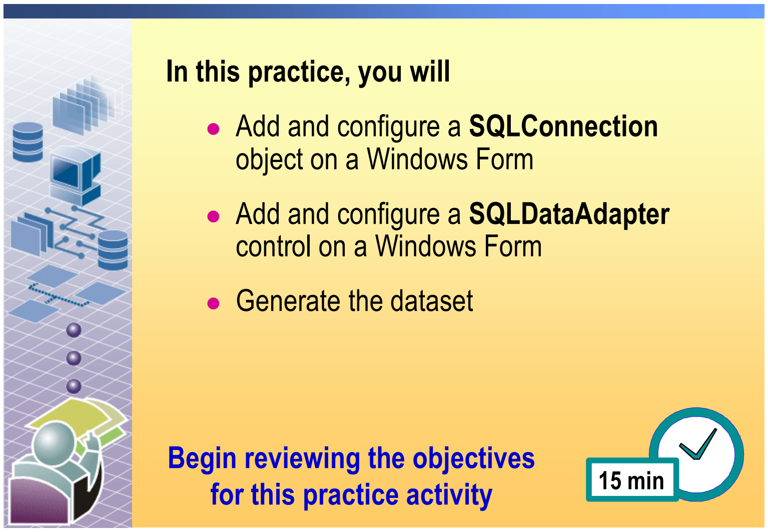
   - If you ask to create a connection, the wizard displays the **Connection** tab of the **Data Link Properties** dialog box, which allows you to specify a provider, server name, database name, user name, and password for the connection.
   - To help you create SQL statements, the wizard provides the Query Builder, a utility that allows you to create and test a **Select** statement by using visual tools. To launch it, click the **Query Builder** button when asked for a SQL statement.
3. In Component Designer, select the adapter or adapters that will transfer data between the data source and the dataset.

   Typically, each data adapter accesses data in a single table. Therefore, to create a dataset that contains multiple data tables, select all the adapters for the tables that you want to work with.
4. On the **Data** menu, choose **Generate Dataset**.

   The **Generate DataSet** dialog box appears.
5. Click **New**, and then specify a name for the new dataset. If you want to add the dataset to your form or component, click **Add an instance of this DataSet to the designer**.

   This generates a typed dataset.

# Practice: Adding ADO.NET Objects to and Configuring ADO.NET Objects in a Windows Forms Application



In this practice, you will

- Add and configure a **SQLConnection** object on a Windows Form
- Add and configure a **SQLDataAdapter** control on a Windows Form
- Generate the dataset

Begin reviewing the objectives for this practice activity

**Introduction**

In this practice, you will add ADO.NET objects to and configure ADO.NET objects on a form by using Data Adapter Configuration Wizard.

**Instructions**

► **Create a Windows application project**

1. Open Visual Studio .NET.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **New Project** dialog box, select **Windows Application**, name it **BuildingDataSets**, and then click **OK**.

► **Add a SQLConnection control to the form and configure it**

1. Drag a **SQLConnection** control from the ToolBox to Form1.

   The SQLConnection control is on the **Data** tab of the Toolbox.
2. In the Properties window, click the **ConnectionString** property, click the arrow, and then click **New Connection**.
3. In the **Data Link Properties** dialog box, type <*computername*>**\MOC** where <*computername*> is the name of your computer.
4. Select the **Use Windows NT Integrated Security** option.
5. Select the **pubs** database from the drop-down list, click **Test Connection** to ensure that you can access the **pubs** database, and then click **OK**.
6. Click **OK** to close the **Data Link Properties** dialog box.
7. Review the **ConnectionString** text that is generated.

### ► Add a SQLDataAdapter control to the form and configure it

1. Drag a **SQLDataAdapter** control from the ToolBox to Form1.
2. In opening page of the Data Adapter Configuration Wizard, click **Next**.
3. In the data connection drop-down list, click <*computername*>**\moc\pubs.dbo**, and then click **Next**.
4. In the **Choose a Query Type** dialog box, click **Next**.

   You can use this wizard create SQL statements and stored procedures or modify existing stored procedures.
5. On the Toolbar, click the **Query Builder** button.
6. In the **Add Table** dialog box, select the **Titles** tables, click **Add**, and then click **Close**.
7. In the **Query Builder** dialog box, in the Titles table, select **All Columns**, and then click **OK**.
8. Click Next to open the **View Wizard Results** dialog box.
9. Review the information in the **View Wizard Results** dialog box, and then click **Finish**.

### ► Review the code generated

1. Review the **SelectCommand**, **DeleteCommand**, **InsertCommand** and **UpdateCommand** properties of the **SQLDataAdapter1** control in the Properties window.

   Each of these properties is associated with a **SQLCommand** object that was generated by the wizard. Each operation (SELECT, INSERT, UPDATE, DELETE) uses a specific **SQLCommand** to perform its task.
2. Expand the **DeleteCommand** property and review the **CommandText** property.

   Each **SQLCommand** object has a unique **CommandText** property that is generated by the wizard and enables the object to perform its task.

### ► Generate a typed DataSet

1. Right-click the **SQLDataAdapter1** control, and click **Generate DataSet**.

   **Note** If the **Generate DataSet** option is disabled, then click anywhere outside the Properties window before generating the dataset.

2. Click **OK** in the **Generate Dataset** dialog box.

   A dataset control named **DataSet11** is added to Form1. **DataSet11** is an instance of the typed dataset **DataSet1**.
3. In Solution Explorer, double-click **DatSet1.xsd**.

   This is the visual representation of your dataset. In this view you can add new or existing tables to your dataset and create relationships between them.

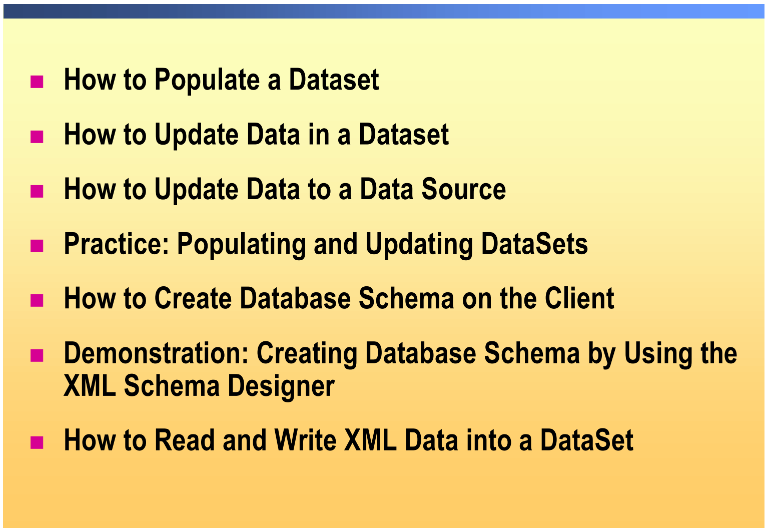
4. Click **XML** in the lower left of the designer.

   This is the XML representation of your dataset.

5. In Solution Explorer, click **Show All Files**.
6. In Solution Explorer, expand **DataSet1.xsd**, and then double-click **DataSet1.cs**.
7. Review the code.

   The object created is a typed dataset that inherits from the **DataSet** class. The data information in the .xsd file is used to create the typed dataset.

# Lesson: Accessing and Modifying Data by Using DataSets



- How to Populate a Dataset
- How to Update Data in a Dataset
- How to Update Data to a Data Source
- Practice: Populating and Updating DataSets
- How to Create Database Schema on the Client
- Demonstration: Creating Database Schema by Using the XML Schema Designer
- How to Read and Write XML Data into a DataSet

### Introduction

ADO.NET provides you with the **Dataset** object for the caching of data on the client computer. This dataset is automatically disconnected from the data source but maintains the ability to later update the source based on changes made at the client.

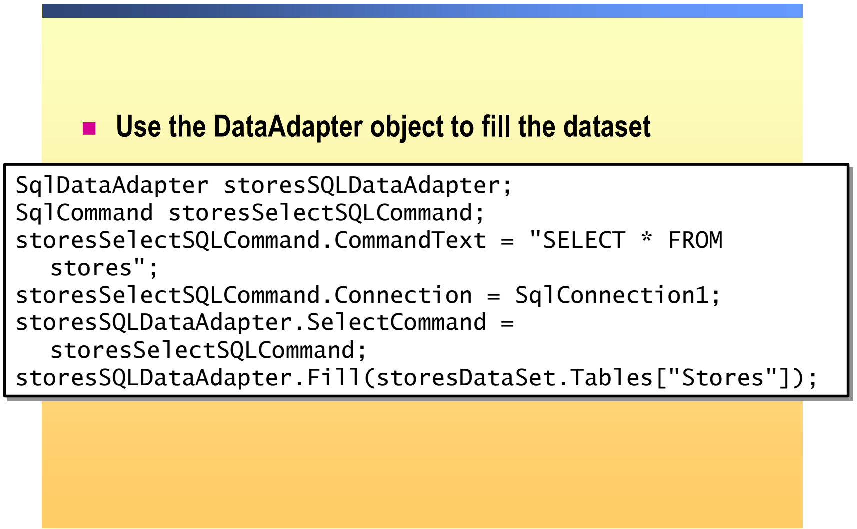
This lesson shows how to create datasets and populate tables in them. This lesson also shows how to edit these tables and propagate those changes to the data source.

### Lesson objectives

After completing this lesson, you will be able to:

- Populate a dataset with data from a data source.
- Update the data in a dataset.
- Update the data source by using a dataset.
- Create relationships between tables in a dataset.
- Create database schema on the client.
- Read XML data into a dataset.
- Write data from a dataset into an XML file.

# How to Populate a Dataset



- Use the DataAdapter object to fill the dataset

```
SqlDataAdapter storesSQLDataAdapter;
SqlCommand storesSelectSQLCommand;
storesSelectSQLCommand.CommandText = "SELECT * FROM
   stores";
storesSelectSQLCommand.Connection = SqlConnection1;
storesSQLDataAdapter.SelectCommand =
   storesSelectSQLCommand;
storesSQLDataAdapter.Fill(storesDataSet.Tables["Stores"]);
```

### Introduction

A dataset is an in-memory representation of data and does not actually contain any data until you fill it by using a **DataAdapter** object. After you create the **DataAdapter** object, you use the **Fill** method, passing a **DataSet** and, optionally, the **DataTable** name as parameters.

### Populating a dataset

The **DataAdapter** uses the **SelectCommand** object to execute the query and returns the results to the **DataSet** or to the **DataTable** referenced as parameters. The following code programmatically creates and configures a **SQLDataAdapter** and a **SQLCommand** and then uses the **Fill** method to populate a table in a dataset:

```
SqlDataAdapter storesSQLDataAdapter;
SqlCommand storesSelectSQLCommand;
storesSelectSQLCommand.CommandText = "SELECT * FROM stores";
storesSelectSQLCommand.Connection = SqlConnection1;
storesSQLDataAdapter.SelectCommand = storesSelectSQLCommand;
storesSQLDataAdapter.Fill(storesDataSet.Tables["Stores"]);
```

### Passing parameters to SELECT statements

It is often necessary to pass parameters to a SQL statement. For example, when accessing rows from a data source, you use the **SELECT** statement, which uses a unique identifier to identify the rows to be accessed. The unique identifier is commonly the value of a primary key field. The **SELECT** statement uses parameters that contain the unique identifier, and the columns and values to be updated, as shown in the following SQL statement:

```
SELECT stor_id, ord_num, qty, ord_date, payterms, title_id
FROM sales WHERE (stor_id = @stor_id)
```

In the previous example, the **stor_id** field must be populated with a value from the **@stor_id** parameter for the SQL statement to return results.

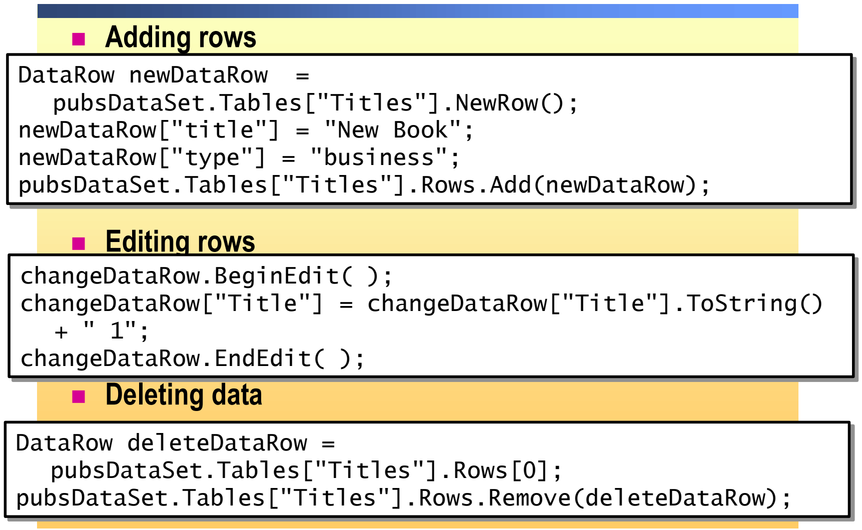
A **Parameter** object holds the input and output parameters of SQL statement and stored procedures. The **Parameters** collection of a **Command** object is where you add the required arguments of SQL statements or stored procedures. There are a variety of ways to add **Parameter** objects to the **Parameters** collection. The following code uses the **SelectCommand** property to access a **Command** object and then assigns the value of *stor_id* to the **Value** property of the **Parameter** object in the **Parameters** collection that is identified by *@stor_id*:

```
salesSqlDataAdapter.SelectCommand.Parameters["@stor_id"].Value
= stor_id;
```

After all the arguments for a SQL statement or stored procedure are defined in **Parameter** objects, you can call the **Fill** method of the **DataAdapter** to get the results.

```
salesSqlDataAdapter.Fill(StoreSalesDataSet1.sales);
```

# How to Update Data in a DataSet



- Adding rows

```
DataRow newDataRow  =
   pubsDataSet.Tables["Titles"].NewRow();
newDataRow["title"] = "New Book";
newDataRow["type"] = "business";
pubsDataSet.Tables["Titles"].Rows.Add(newDataRow);
```

- Editing rows

```
changeDataRow.BeginEdit( );
changeDataRow["Title"] = changeDataRow["Title"].ToString()
   + " 1";
changeDataRow.EndEdit( );
```

- Deleting data

```
DataRow deleteDataRow =
   pubsDataSet.Tables["Titles"].Rows[0];
pubsDataSet.Tables["Titles"].Rows.Remove(deleteDataRow);
```

### Introduction

After you have created a dataset of **DataTables**, you might want to add, update, and delete data. Any changes that you make to the data are stored in memory and later used to apply the changes to the data source.

### Procedure: Adding rows to a dataset

To add new rows to a dataset table:

1. Instantiate a **DataRow** object by using the **NewRow** method of the **DataTable**.
2. Populate the columns with data.
3. Call the **Add** method of the **DataRows** collection, passing the **DataRow** object.

   The following code shows how to add rows to a dataset:

```
DataRow newDataRow  =
pubsDataSet.Tables["Titles"].NewRow();
newDataRow["title"] = "New Book";
newDataRow["type"] = "business";
pubsDataSet.Tables["Titles"].Rows.Add(newDataRow);
```

### Procedure: Editing rows in a dataset

To edit existing rows in a dataset table:

1. Call the **BeginEdit** method of the row.
2. Change the data in the columns.
3. Call **EndEdit** or **CancelEdit** to accept or reject the changes.

   The following code shows how to edit data in an existing column:

   ```
   DataRow changeDataRow  =
   pubsDataSet.Tables["Titles"].Rows[0];
   changeDataRow.BeginEdit( );
   changeDataRow["Title"] = changeDataRow["Title"].ToString()
   + " 1";
   changeDataRow.EndEdit( );
   ```

### Procedure: Deleting data in a dataset

Use either of the following methods to delete a row:

- **Remove** method

  Call the **Remove** method of the **DataRows** collection. This permanently removes the row from the dataset.

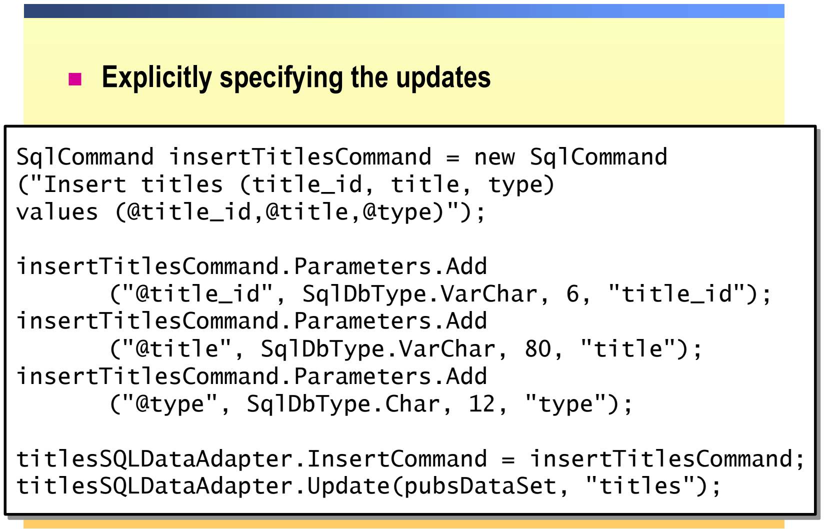
- **Delete** method

  Call the **Delete** method of the **DataRow** object. This only marks the row for deletion in the dataset, and calling **RejectChanges** will undo the deletion.

  The following code shows how to delete an existing row from a dataset:

  ```
  DataRow deleteDataRow =
  pubsDataSet.Tables["Titles"].Rows[0];
  pubsDataSet.Tables["Titles"].Rows.Remove(deleteDataRow);
  ```

# How to Update Data to a Data Source



- Explicitly specifying the updates

```
SqlCommand insertTitlesCommand = new SqlCommand
("Insert titles (title_id, title, type)
values (@title_id,@title,@type)");

insertTitlesCommand.Parameters.Add
      ("@title_id", SqlDbType.VarChar, 6, "title_id");
insertTitlesCommand.Parameters.Add
      ("@title", SqlDbType.VarChar, 80, "title");
insertTitlesCommand.Parameters.Add
      ("@type", SqlDbType.Char, 12, "type");

titlesSQLDataAdapter.InsertCommand = insertTitlesCommand;
titlesSQLDataAdapter.Update(pubsDataSet, "titles");
```

### Introduction

After you have updated the tables in your dataset, you will want to replicate those changes to the underlying data source. To do this, use the **Update** method of the **DataAdapter** object, which is the link between dataset and data source.

The **Update** method, like the **Fill** method, takes two parameters: the **DataSet** and the name of the **DataTable** in which the changes have been made. The **Update** method determines the changes to the data and executes the appropriate SQL command (Insert, Update, or Delete) against the source data.

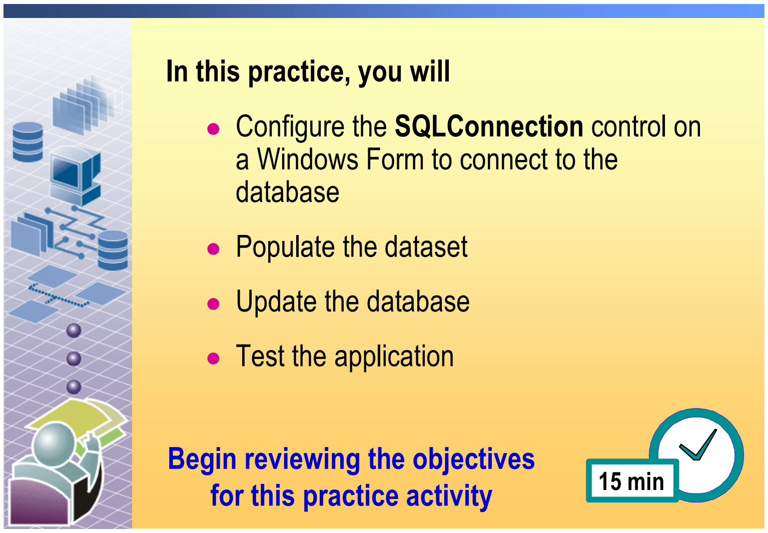
### Explicitly specifying the updates

You use the **InsertCommand**, **UpdateCommand**, and **DeleteCommand** properties of the **DataAdapter** to identify the changes occurring in your dataset. You specify each of these as an existing command object for an Insert, Update, or Delete SQL statement. For any variable columns in the statements, you use **SqlParameter** objects to identify the column, data type, size, and data to be inserted.

The following code shows how to use the **InsertCommand** property to add a row to the Titles table in the **pubs** database:

```
SqlCommand insertTitlesCommand = new SqlCommand
  ("Insert titles (title_id, title, type) values
  (@title_id,@title,@type)");

insertTitlesCommand.Parameters.Add
  ("@title_id", SqlDbType.VarChar, 6, "title_id");
insertTitlesCommand.Parameters.Add
  ("@title", SqlDbType.VarChar, 80, "title");
insertTitlesCommand.Parameters.Add
  ("@type", SqlDbType.Char, 12, "type");

titlesSQLDataAdapter.InsertCommand = insertTitlesCommand;
titlesSQLDataAdapter.Update(pubsDataSet, "titles");
```

# Practice: Populating and Updating DataSets



**In this practice, you will**

- Configure the **SQLConnection** control on a Windows Form to connect to the database
- Populate the dataset
- Update the database
- Test the application

Begin reviewing the objectives for this practice activity

15 min

**Introduction**

In this practice, you will populate a dataset and update a database with the changes made to the dataset. You will fill the dataset with the Stores and Sales tables of the pubs database. For the Stores table, you will write all of the code required to populate the dataset. For the Sales table, you will use the code generated by Data Adapter Configuration Wizard and a typed dataset. You will also update the database with any changes to the Sales table.

**Instructions**

### ► Open the practice project

1. Use Microsoft Windows® Explorer to browse to *install_folder*\Practices\Mod04\Mod04_02\Starter.

**Note** If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

2. Double-click the PopulatingAndUpdatingDataSets.sln solution file to open the project.

### ► Populate StoreSalesDataSet1 with data from the Stores table

1. Show TODO comments in the Task List.

   To show TODO comments, click the **View** menu, point to **Show Tasks**, and then click **All**.

2. Locate TODO: 1 in the Code Editor of Form1.
3. Under TODO: 1, declare and instantiate a **SqlDataAdapter** named **storesSQLDataAdapter**.
4. Declare and instantiate a **SqlCommand** named **storesSelectSQLCommand**.

5. Declare a **DataTable** named **storesTable** and initialize it with the String **Stores**. Your code should look like this:

```
System.Data.SqlClient.SqlDataAdapter storesSQLDataAdapter =
new System.Data.SqlClient.SqlDataAdapter();
System.Data.SqlClient.SqlCommand storesSelectSQLCommand  =
new System.Data.SqlClient.SqlCommand();
DataTable storesTable = new DataTable("Stores");
```

6. Locate TODO: 2 in the Code Editor of Form1.
7. Under TODO: 2, set the **CommandType** property of **storesSelectSQLCommand** to **CommandType.Text**.
8. Set the **CommandText** property of **storesSelectSQLCommand** to, "SELECT stor_id, stor_name FROM stores".
9. Set the **Connection** property of **storesSelectSQLCommand** to **SQLConnection1**.
10. Set the **SelectCommand** property of **storesSQLDataAdapter** to **storesSelectSQLCommand**. Your code should look like this:

```
storesSelectSQLCommand.CommandType = CommandType.Text;
storesSelectSQLCommand.CommandText = "SELECT stor_id,
stor_name FROM stores";
storesSelectSQLCommand.Connection = sqlConnection1;
storesSQLDataAdapter.SelectCommand =
storesSelectSQLCommand;
```

11. Locate TODO: 3 in the Code Editor of Form1.
12. Under TODO: 3, use the **Fill** method of **storesSQLDataAdapter** to populate **storesTable**.
13. Use the **Add** method to add **storesTable** to the **Tables** collection of the existing **StoreSalesDataSet** dataSet. Your code should look like this:

```
storesSQLDataAdapter.Fill(storesTable);
StoreSalesDataSet1.Tables.Add(storesTable);
```

### ► Use existing code generated by Data Adapter Configuration Wizard and a typed dataset to populate StoreSalesDataSet1 with data from the Sales table

The **CommandText** property of **SalesSQLSelectCommand** is set to the following:

```
"SELECT stor_id, ord_num, qty, ord_date, payterms, title_id
FROM sales WHERE (stor_id = @stor_id)"
```

You must assign the **@stor_ID** parameter a value to get the required results. You use the **Parameters** collection of the **SQLCommand** object to assign this value.

1. Locate TODO: 4 in the Code Editor of Form1.
2. Under TODO: 4, use the **SelectCommand** property of **SalesSqlDataAdapter** to access **SalesSQLSelectCommand** and set the **Value** property of the **@storeid** parameter in the **Parameters** collection to **storeID**. Your code should look like this:

   ```
   SalesSQLDataAdapter.SelectCommand.Parameters["@stor_id"].
   Value = storeID;
   ```
3. Locate TODO: 5 in the Code Editor of Form1.
4. Under TODO: 5, use the **Clear** method of **Sales** DataTable in **StoreSalesDataSet1** to clear the **Sales** table of any existing data. Your code should look like this:

   ```
   StoreSalesDataSet1.sales.Clear();
   ```
5. Locate TODO: 6 in the Code Editor of Form1.
6. Under TODO: 6, use the **Fill** method of **SalesSQLDataAdapter** to populate the **sales** property of **StoreSalesDataSet1**. Your code should look like this:

   ```
   SalesSqlDataAdapter.Fill(StoreSalesDataSet1.sales);
   ```

### ► Update the pubs database with the changes in StoreSalesDataSet1

1. Locate TODO: 7 in the Code Editor of Form1.
2. Under TODO: 7, use the **Update** method of **SalesSqlDataAdapter**, and pass **StoreSalesDataSet1** as an argument. Your code should look like this:

   ```
   SalesSqlDataAdapter.Update(StoreSalesDataSet1);
   ```

### ► Test the application

1. Press F5 to compile and run the application.
2. In the **Stores** list, click **News and Brews**.

**Tip** You may want to set breakpoints in your code to follow the execution path.

3. Click the bottom row of the DataGrid to create a new row, and enter the following fields.

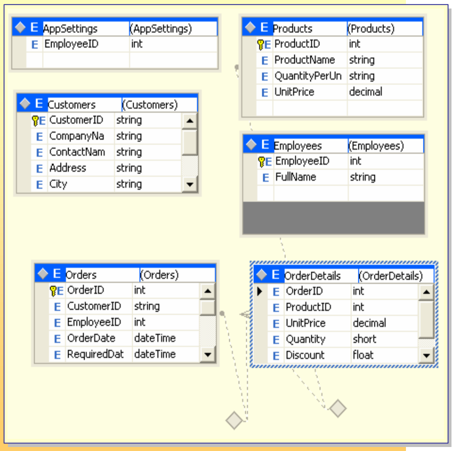
**Note** The date in the following table is in the mm/dd/yy format for the purposes of practicing the concepts learned in the lesson. However, when you create an application for an international audience, you should use a different method of capturing the date - like a pop-up calendar to select the date, or three individual list boxes: for the month, day, and year.

| Field | Value |
|---|---|
| stor_id | 7067 |
| ord_num | P2122 |
| qty | 12 |
| ord_date | 6/13/2002 |
| payterms | Net 60 |
| title_id | PC9999 |

4. Click **Update**.
5. Switch to the Visual Studio .NET IDE, and use Server Explorer to verify that the database was updated.
6. Switch back to the Store Orders application, set the **qty** field of the new row to **24**, and then click **Update**.
7. Switch to the Visual Studio .NET IDE, and use Server Explorer to verify that the database was updated.
8. To refresh the view, right-click the results, and then click **Run**.
9. Switch back to the Store Orders application, delete the new row, and then click **Update**.
10. Switch to the Visual Studio .NET IDE, and use Server Explorer to verify that the database was updated.

# How to Create Database Schema on the Client

- **XML Schema (.xsd) files enforce data integrity on the client**
- **Use the XML Designer to create and modify XML Schema files**
  1. Determine the schema design
  2. On the **Project** menu, click **Add New Item**
  3. Add the schema
  4. Create the schema



| E AppSettings | (AppSettings) |
|---|---|
| E EmployeeID | int |

| E Products | (Products) |
|---|---|
| E ProductID | int |
| E ProductName | string |
| E QuantityPerUn | string |
| E UnitPrice | decimal |

| E Customers | (Customers) |
|---|---|
| E CustomerID | string |
| E CompanyNa | string |
| E ContactNam | string |
| E Address | string |
| E City | string |

| E Employees | (Employees) |
|---|---|
| E EmployeeID | int |
| E FullName | string |

| E Orders | (Orders) |
|---|---|
| E OrderID | int |
| E CustomerID | string |
| E EmployeeID | int |
| E OrderDate | dateTime |
| E RequiredDat | dateTime |

| E OrderDetails | (OrderDetails) |
|---|---|
| E OrderID | int |
| E ProductID | int |
| E UnitPrice | decimal |
| E Quantity | short |
| E Discount | float |

****************************ILLEGAL FOR NON-TRAINER USE******************************

## Introduction

When a typed dataset is generated at design time, a dataset class and an associated XML Schema file is created in your project. XML Schema files define and validate the data being imported from XML streams or documents into typed datasets. The XML Schema file associated with the dataset can be loaded into the XML Designer providing a visual representation of the dataset structure that can be edited and viewed. You can create and edit table structures and create relations between tables by using the XML Designer.

In typed datasets, XML Schema (.xsd) files establish the relational structure of the dataset's tables and columns, the key columns, constraints, and relationships between tables. The relational structure information is used when generating a **DataSet** class. The table relation information is available within a dataset, and referential integrity can be maintained within the DataSet.

### Procedure: Creating an XML schema

To create an XML schema:

1. Determine the design needed for your schema.
2. On the **Project** menu, click **Add New Item**.
3. Do one of the following:
   - Add a schema. To add a schema, open the appropriate folder, and then double-click **XML Schema**.

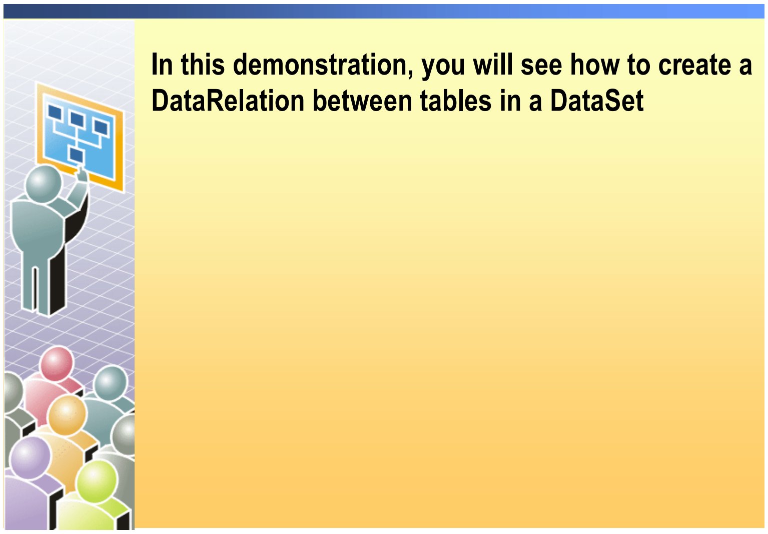
     An XML Schema (.xsd) file is added to your project.

     - Or -

   - Add a dataset. To add a dataset, open the appropriate folder, and then double-click **DataSet**.

     An XML Schema file and typed DataSet class file (.vb or .cs file) are added to your project.
4. To create the schema, add elements and attributes.

# Demonstration: Creating Database Schema by Using the XML Schema Designer



In this demonstration, you will see how to create a DataRelation between tables in a DataSet

### Introduction

In this demonstration, you will see the creation of a DataRelation in a dataset between the Suppliers and Products tables of the **Northwind** database. After the DataRelation is created, you will see how referential integrity is maintained on the client in the dataset.

### Instructions

► **Open the CreatingRelations.sln solution file**

- Open the CreatingRelations.sln solution file in Visual Studio .NET from *install_folder*\Democode\ Mod04\Mod04_01\Starter.

► **Test the behavior of the dataset before the relation is created**

1. Press F5 to compile and run the application.

   On the **Load** event of Form1, the **SuppliersSqlDataAdapter** and the **ProductsSqlDataAdapter** fill the **SuppliersProductsDataSet1** dataset.

2. Select the first row in the **Suppliers** DataGrid, and press DELETE.

   Because there is no relationship between the tables on the client dataset, the dataset does not maintain referential integrity.

3. Click the **Submit Data** button.

   The database is aware of the relationship between the **Suppliers** and **Products** tables and prevents the changes from being persisted.

4. Close the application.

### ► Create a relationship between the Suppliers and Products tables in the SuppliersProductsDataSet dataset

1. Open SuppliersProductsDataSet.xsd in Design view.
2. Drag a **Relation** control from the Toolbox to the Suppliers table.
3. In the **Edit Relation** dialog box, in the **Parent element** list, click **Suppliers**, and in the **Child element** list, click **Products**.

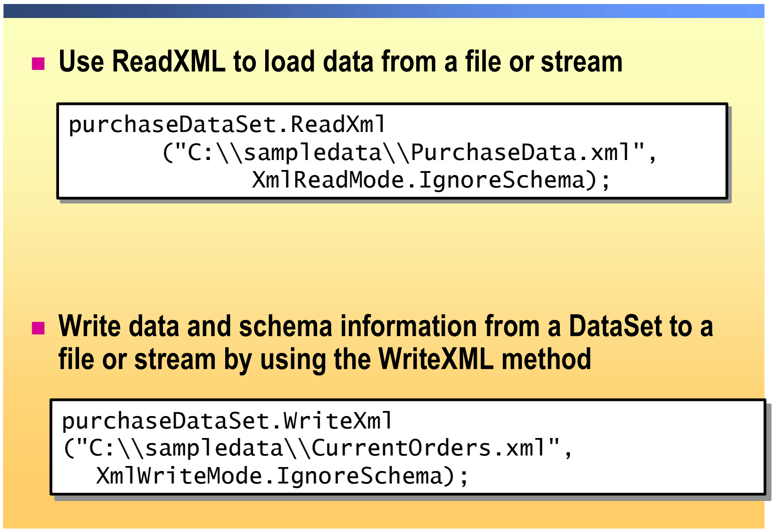
   Notice the **Update rule**, **Delete rule**, and **Accept/Reject rule** lists. You can control how a dataset manages DataRelations between tables when changes and updates occur. For example, if you set the **Delete rule** to **Cascading** (the default) the dataset will automatically delete all child elements when a parent is deleted to enforce referential integrity.
4. Click **OK**.
5. Close SuppliersProductsDataSet.xsd, and then click **Yes** to save the changes.

### ► Test the behavior of the dataset and notice that a relation exists

1. Press F5 to compile and run the application.
2. Notice that the first row of the Products DataGrid displays the product Chai, which has a SupplierID of 1.
3. Select the first row in the Suppliers DataGrid (where SupplierID = 1), and press DELETE.

Because the DataRelation exists, the dataset cascades the deletion of all the products that are associated with the Supplier parent and maintains referential integrity.

# How to Read and Write XML Data into a DataSet



- Use ReadXML to load data from a file or stream

```
purchaseDataSet.ReadXml
       ("C:\\sampledata\\PurchaseData.xml",
              XmlReadMode.IgnoreSchema);
```

- Write data and schema information from a DataSet to a file or stream by using the WriteXML method

```
purchaseDataSet.WriteXml
("C:\\sampledata\\CurrentOrders.xml",
   XmlWriteMode.IgnoreSchema);
```

### Introduction

You can use the **ReadXml** method of the **DataSet** object to load data from an XML file into a dataset. When you use this method, you can load data from XML files that contain only XML data or from files that contain XML data as well as an inline schema. You can write data and schema information from a dataset to a file or stream by using the **WriteXml** method of the **Dataset** object.

An *inline schema* is an XSD schema that appears at the beginning of the XML data file. This schema describes the XML information that appears after the schema in the XML file.

### Simplified syntax for reading XML data

The **ReadXml** method is overloaded and can be used to read from a stream object, an XML file, a **TextReader** subclass object, or an **XmlReader** subclass object, as shown in the following code:

```
Dataset.ReadXml(Stream | FileName | TextReader | XmlReader, {
ByVal mode as XmlReadMode })
```

Use the *XmlReadMode* parameter to specify what the XML file contains and what information should be loaded from the file. This parameter is optional. If no **XmlReadMode** value is supplied, the default value **Auto** is used.

## XmlReadMode parameter values

The following table shows the values for the *XmlReadMode* parameter of the **ReadXml** method of the **DataSet** object.

| XmlReadMode value | Description |
|---|---|
| ReadSchema | Reads any inline schema and then loads the schema and data:<br>■ If the dataset already contains a schema, any new tables that are defined by an inline schema are added to the dataset.<br>■ If the inline schema defines a table that is already in the dataset, an exception is thrown.<br>■ If the dataset does not contain a schema, and there is no inline schema, no data is read. |
| IgnoreSchema | Ignores any inline schema and loads data into the existing dataset. Any data that does not match the existing schema is discarded. |
| InferSchema | Ignores any inline schema and infers a new schema based on the structure of the XML data. If the dataset already defines a schema, tables are added to this schema.<br>The data is then loaded into the dataset. |
| DiffGram | Reads a DiffGram and adds the data to the current schema in the dataset. A DiffGram is an XML format that is used to identify current and original versions of data elements. |
| Fragment | Reads XML fragments and appends data to appropriate dataset tables. This setting is typically used to read XML data generated directly from SQL Server. |
| Auto | Examines the XML file and chooses the most appropriate option.<br>■ If the dataset contains a schema or the XML contains an inline schema, ReadSchema is used.<br>■ If the dataset does not contain a schema and the XML does not contain an inline schema, **InferSchema** is used.<br>For best performance, specify an **XmlReadMode** rather than **Auto**. |

### Example of loading a schema and data into a dataset

The following example first loads a schema into a new dataset by using the **ReadXmlSchema** method and then loads the data from an XML file by using the **ReadXml** method with the **IgnoreSchema** option of the *XmlReadMode* parameter:

```
private void ReadXmlDataOnly()
{
  try
  {
      DataSet purchaseDataSet = new DataSet();
      Console.WriteLine("Reading the Schema file");
      purchaseDataSet.ReadXmlSchema
          ("C:\\sampledata\\PurchaseData.xsd");
      Console.WriteLine("Loading the XML data file");
      purchaseDataSet.ReadXml
          ("C:\\sampledata\\PurchaseData.xml",
          XmlReadMode.IgnoreSchema);
      dataGrid1.DataSource = purchaseDataSet.Tables[0];
  }
  catch (Exception e)
  {
      Console.WriteLine("Exception: " + e.ToString());
  }
}
```

### Partial syntax for writing to an XML file

The following code shows partial syntax for the **WriteXml** method of the **DataSet** object:

```
Overloads Public Sub WriteXml (ByVal filename As String |
stream As Stream | writer as TextWriter | writer as XmlWriter,
{ByVal mode As XmlWriteMode})
```

### XmlWriteMode values

When you use the **WriteXml** method, you can specify an optional value for the *XmlWriteMode* parameter. This parameter specifies whether to generate a file that contains only XML data, XML data with an inline XSD schema, or a DiffGram.

The following table describes the different values for the *XmlWriteMode* parameter of the **WriteXml** method of the **DataSet** object.

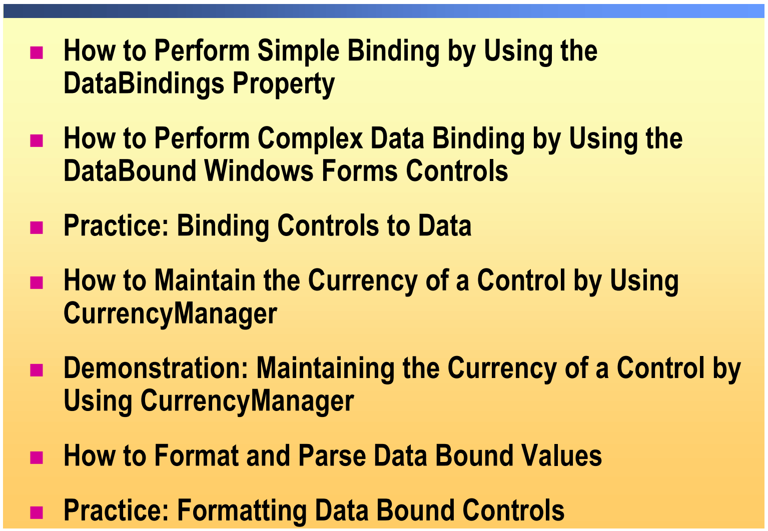
| XmlWriteMode value | What is generated |
|---|---|
| IgnoreSchema | An XML file containing the data from a dataset. No schema information is included. If the dataset is empty, no file is created. |
| WriteSchema | An XML file containing an inline schema and the data from a populated dataset. If the dataset contains only schema information, an inline schema is written to the output file. If the dataset does not include schema information, no file is created. |
| DiffGram | An XML file in the form of a DiffGram, containing both the original and current values for the data. |

### Example of writing XML data to a file

The following code saves the data stored in a dataset as an XML file but does not write any schema information:

```
private void SaveXMLDataOnly()
{
  try
  {
      DataSet purchaseDataSet = new DataSet();
      //Load an inline schema and data from an XML file
      purchaseDataSet.ReadXml
          ("C:\\sampledata\\PurchaseOrder.xml",
          XmlReadMode.ReadSchema);
      //Save the data portion of the DataSet to a file
      purchaseDataSet.WriteXml
("C:\\sampledata\\CurrentOrders.xml",
XmlWriteMode.IgnoreSchema);
  }
  catch (Exception exp)
  {
      Console.WriteLine("Exception: " + exp.ToString());
  }
}
```

# Lesson: Binding Data to Controls



- How to Perform Simple Binding by Using the DataBindings Property
- How to Perform Complex Data Binding by Using the DataBound Windows Forms Controls
- Practice: Binding Controls to Data
- How to Maintain the Currency of a Control by Using CurrencyManager
- Demonstration: Maintaining the Currency of a Control by Using CurrencyManager
- How to Format and Parse Data Bound Values
- Practice: Formatting Data Bound Controls

**Introduction**

In Windows Forms, you can bind to not just traditional data sources but also to almost any structure that contains data. You can bind to an array of values that you calculate at run time, read from a file, or derive from the values of other controls. In addition, you can bind any property of any control to the data source. In traditional data binding, you typically bind the display property (for example, the **Text** property of a **TextBox** control) to the data source. With the .NET Framework, you also have the option of binding other properties.

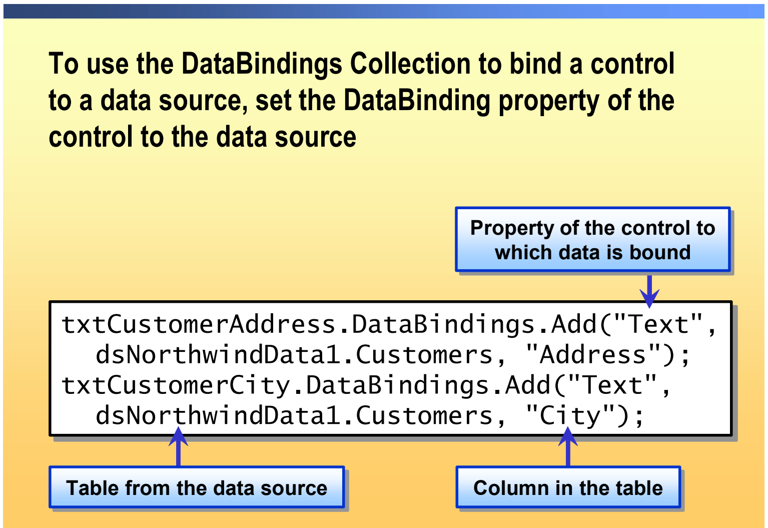
Windows Forms can take advantage of two types of data binding: simple binding and complex binding. This lesson covers how to bind data to controls by using simple binding and complex binding.

**Lesson objectives**

After completing this lesson, you will be able to:

- Describe the **ControlBindings** collection.
- Perform simple binding by using the **DataBindings** property.
- Perform complex binding by using data bound Windows Forms controls.
- Describe the **CurrencyManager** object.
- Maintain the currency of an object by using the **BindingContext** object.
- Format data-bound values.

# How to Perform Simple Binding by Using the DataBindings Property



**To use the DataBindings Collection to bind a control to a data source, set the DataBinding property of the control to the data source**

Property of the control to which data is bound

```
txtCustomerAddress.DataBindings.Add("Text",
  dsNorthwindData1.Customers, "Address");
txtCustomerCity.DataBindings.Add("Text",
  dsNorthwindData1.Customers, "City");
```

Table from the data source

Column in the table

### Introduction

In Windows Forms, you can bind to not just traditional data sources but also to almost any structure that contains data. For example, you can bind to arrays, collections, properties of other controls, and ADO.NET objects such as data views and table columns.

Simple data binding is accomplished by adding **Binding** objects to a **ControlBindingsCollection**. The **Binding** class is used to create and maintain a simple binding between the property of a control and the property of an object. Any object that inherits from the **Control** class maintains a list of the **Binding** objects in the **ControlBindingsCollection**. The **DataBindings** property exposes the **ControlBindingsCollection** and, like any collection, supports methods to Add, Remove, and Clear the objects (Bindings) in it.

### Procedure: Performing simple binding at design time

To perform a simple bind of a control:

1. In the form, select the control and display the Properties window.
2. Expand the **(DataBindings)** property.

   The properties most often bound are displayed in the **(DataBindings)** property list. For example, in most controls, the **Text** property is most frequently bound.
3. If the property that you want to bind is not one of the commonly bound properties, click the **Ellipsis** button (…) in the **Advanced** box to display the **Advanced Data Binding** dialog box with a complete list of properties for that control.
4. Click the list arrow for the property that you want to bind.

   A list of available data sources is displayed.

5. Expand the data source that you want to bind to until you find the single data element that you want.

   For example, if you are binding to a column value in a dataset's table, expand the name of the dataset, and then expand the table name to display column names.

6. Click the name of an element to bind to it.

### Procedure: Performing simple binding at run time

You can also create a **Binding** object programmatically at run time. To do so, use the **Add** method of the **DataBindings** collection. The method expects the following arguments:

- Name of the property of the control that will consume the data
- Data source
- Name of the field in the data source to bind to

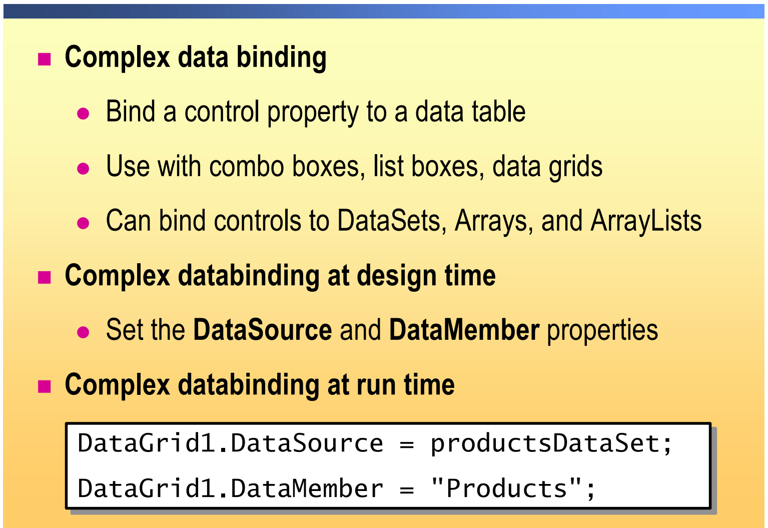
The following code binds the **Text** property of the **txtCustomer** text box to the **Address** column of the **Customers** tables in the dsNorthwind1 typed dataset:

```
txtCustomerAddress.DataBindings.Add("Text",
dsNorthwindData1.Customers, "Address");
```

A period-delimited navigation path is required when the data source is set to an object that contains multiple **DataTable** objects (such as a **DataSet** or **DataViewManager**). For example, the previous code could also be written as shown.

```
txtCustomerAddress.DataBindings.Add("Text", dsNorthwindData1,
"Customers.Address");
```

# How to Perform Complex Data Binding by Using the DataBound Windows Forms Controls



- **Complex data binding**
  - Bind a control property to a data table
  - Use with combo boxes, list boxes, data grids
  - Can bind controls to DataSets, Arrays, and ArrayLists
- **Complex databinding at design time**
  - Set the **DataSource** and **DataMember** properties
- **Complex databinding at run time**

```
DataGrid1.DataSource = productsDataSet;
DataGrid1.DataMember = "Products";
```

### Introduction

Instead of binding each data field to a separate control, you bind the complete data source to one control. This is called complex data binding. A very useful component for complex data binding is the **DataGrid** control.

Because a dataset can contain more than one DataTable, there are two relevant properties: **DataSource** and **DisplayMember**. **DataSource** refers to the dataset and **DisplayMember** refers to the contained DataTable. Alternatively, you can use the **SetDataBinding** method to set both values at once.

### Procedure: Performing complex data binding at design time

To bind a combo box, list box, or data grid to a data source at design time:

1. Add the control to the form.
2. Select the control.
3. Set the **DataSource** property to the data source, such as a dataset.
4. Set the **DisplayMember** property to the name of a column in the data source.

### Procedure: Performing complex data binding at run time

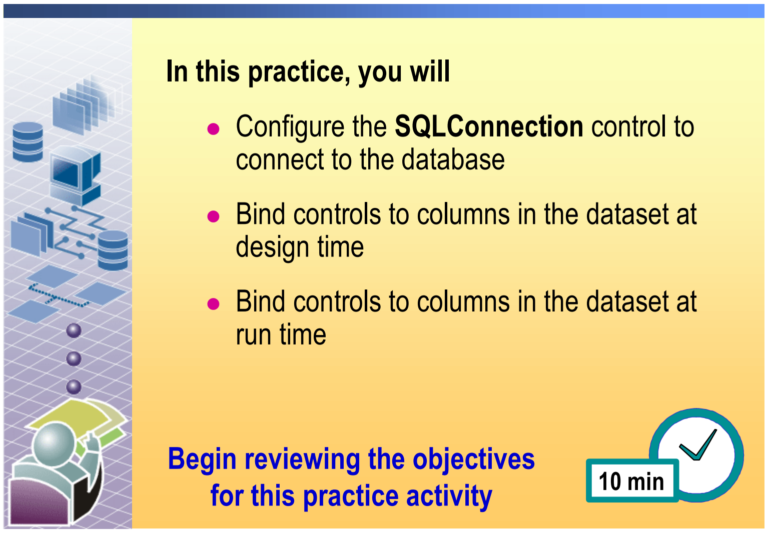
To bind a data-bound control to a data source at run time, add the following lines of code that set the **DataSource** and **DisplayMember** properties:

```
DataGrid1.DataSource = productsDataSet;
DataGrid1.DataMember = "Products";
```

If the **DataSource** property is not ambiguous, you can use it without defining the **DataMember**. The following code accomplishes the same results as the previous line of code:

```
DataGrid1.DataSource = productsDataSet.Tables["Products"];
```

# Practice: Binding Controls to Data



In this practice, you will

- Configure the **SQLConnection** control to connect to the database
- Bind controls to columns in the dataset at design time
- Bind controls to columns in the dataset at run time

Begin reviewing the objectives for this practice activity

### Introduction

In this practice, you will bind various controls to data in a dataset at design time and also programmatically at run time. You will begin by using simple binding to bind data to **Label** controls and then you will use complex data binding to bind data to a **ComboBox** and a DataGrid.

### Instructions

► **Open the practice project**

1. Use Windows Explorer to browse to *install_folder*\Practices\Mod04\Mod04_03\Starter.
2. Double-click the DataBinding.sln solution file to open the project.

► **Bind StateLabel control to the State column by using the Properties window**

1. Show TODO comments in the Task List.

   To show TODO comments, click the **View** menu, point to **Show Tasks**, and then click **All**.
2. Open Form1 in Design view.
3. In the Properties window, select **StateLabel** on Form1, and then expand **DataBindings**.
4. Select the **Text** property, click the **Text Property** arrow, expand **StoresSalesDataSet1**, expand **Sales**, and then click **State**.

### ► Bind CityLabel to the city column programmatically at run time

1. Locate TODO: 1 in the Code Editor of Form1.
2. Under TODO: 1, call the **Add** method of the **DataBindings** collection of **CityLabel**, and pass **Text** as the name of the property, **StoreSalesDataSet1**.**sales** as the name of the data source, and **city** as the name of the column. Your code should look like this:

```
CityLabel.DataBindings.Add("Text",
StoreSalesDataSet1.sales,"city");
```

### ► Bind DataGrid1 to the sales table by using the Properties window

1. Open Form1 in Design view.
2. Select **DataGrid1** on Form1 and use the Properties window to set the **DataSource** property to **StoreSalesDataSet1**.
3. Use the Properties window to set the **DataMember** to **sales**.

### ► Bind StoresComboBox to the stores table programmatically at run time

1. Locate TODO: 2 in the Code Editor of Form1.
2. Under TODO: 2, set the **DataSource** property of **StoresComboBox** to the **Stores** table in **StoreSalesDataSet1**.
3. Set the **DisplayMember** property of **StoresComboBox** to the **stor_name** column.
4. Set the **ValueMember** property of **StoresComboBox** to the **stor_id** column. Your code should look like this:

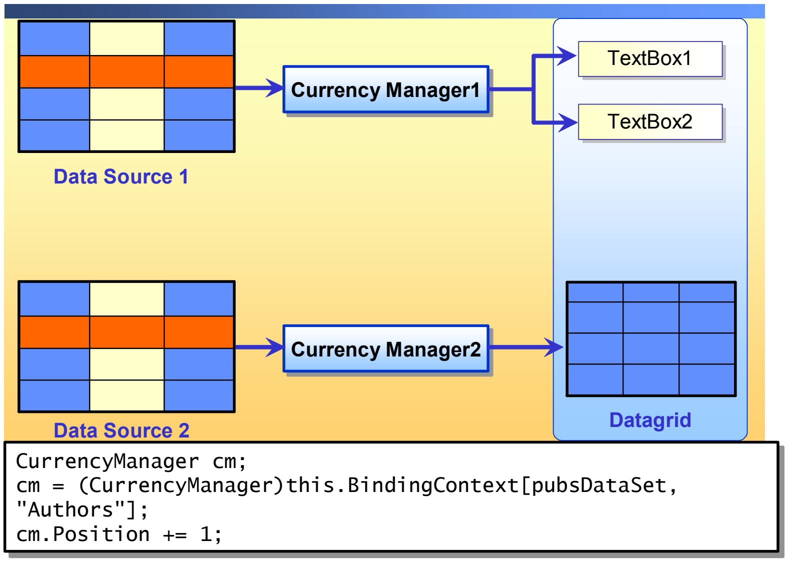
```
StoresComboBox.DataSource =
StoreSalesDataSet1.Tables["Stores"];
StoresComboBox.DisplayMember = "stor_name";
StoresComboBox.ValueMember = "stor_id";
```

**Note** The **ComboBox** control allows you to bind the **ValueMember** property to an alternate value that is not displayed in the **ComboBox**. This is often used when you want to display a user–friendly value to the user and bind to another field that is used programmatically, such as a primary key.

### ► Test the application

1. Press F5 to compile and run the application.
2. Select different stores in the **Stores** list to ensure the controls display the data bound values.

# How to Maintain the Currency of a Control by Using CurrencyManager



```
CurrencyManager cm;
cm = (CurrencyManager)this.BindingContext[pubsDataSet,
"Authors"];
cm.Position += 1;
```

## Introduction

Any data source that you bind to a Windows Forms control to will have an associated **CurrencyManager** object. **CurrencyManager** is an object that is used to keep data-bound controls synchronized with a data source. The **CurrencyManager** object is important because data sources, such as Tables in datasets, do not keep track of the currently selected row. You need an intermediary object that is aware of the currently selected position in a data source and can notify databound controls when the position changes. That intermediary object is **CurrencyManager**. The **CurrencyManager** object synchronizes data-bound controls with the data source by managing a collection of the bound data supplied by a data source.

For each data source associated with a Windows Form, the form maintains at least one **CurrencyManager**. There is a **CurrencyManager** object on the form for each discrete data source that you are binding to. If the controls on the form all bind to a single source (for example, if several **TextBox** controls are bound to the same data table), then they will share the same **CurrencyManager**. However, there are times when controls on the form will be bound to different sources. In that case, there are multiple **CurrencyManager** objects on the form, and each one tracks which record or data element is being used by the controls.

### CurrencyManager properties

The following table lists some of the properties of the **CurrencyManager** object that helps it track data.

| Property | Description |
|---|---|
| Bindings | Gets the collection of bindings being managed. |
| Count | Determines the last item in the list of rows. |
| Current | Contains the value of the current item in the data source. |
| List | Gets the data source (which implements IList) for this **CurrencyManager**. |
| Position | Specifies the position of the item in the underlying list. |

### Procedure: Maintaining the currency by using the CurrencyManager

Every Windows Form has a **BindingContext** object. The **BindingContext** object tracks all of the **CurrencyManager** objects on a form. Thus, any Windows Form with data-bound controls has at least one **BindingContext** object that tracks one (or more) **CurrencyManager** object (or objects) that track one data source (for each **CurrencyManager**).

For example, if you add a **TextBox** control to a form and bind it to a column of a table in a dataset, the control communicates with the **BindingContext** object for that form. The **BindingContext** object, in turn, communicates with the specific **CurrencyManager** object for that data association. If you query the **CurrencyManager's Position** property, it reports the current record for that **TextBox** control's binding.

You use the **BindingContext** of a form to access the various **CurrencyManagers**. To do so, you must specify the data source that the **CurrencyManger** is managing. For example, to access the **CurrenyManager** that manages the Authors table of **DataSet1** you would use the following code:

```
CurrencyManager cm;
cm = (CurrencyManager)this.BindingContext[pubsDataSet,
"Authors"];
```

If you use a container control, such as a **GroupBox**, **Panel**, or **TabControl**, to contain data-bound controls, you can create a **BindingContext** for just that container control and its controls. This allows each part of your form to be managed by its own **CurrencyManager** object.

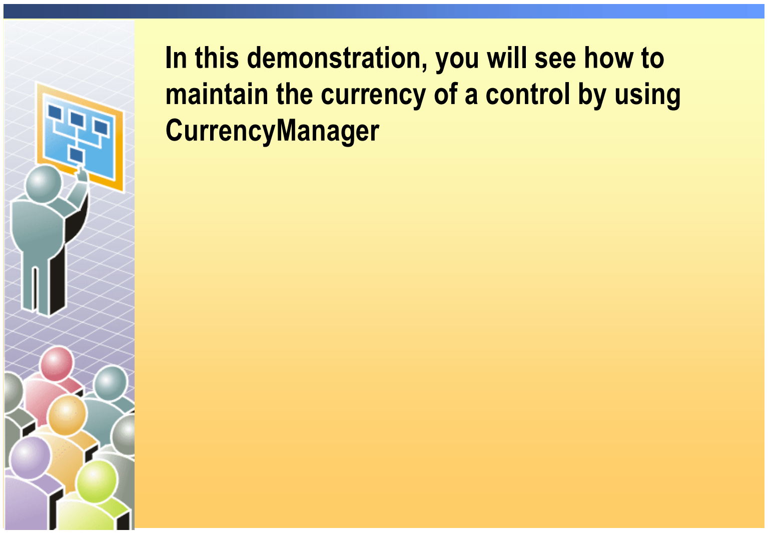
To determine the currency of a control, use the **Position** property of the **CurrencyManager** as shown.

```
cm.Position += 1;
```

You can also use the following code to increment **CurrencyManager**.

```
this.BindingContext[pubsDataSet, "Authors"].Position += 1
```

# Demonstration: Maintaining the Currency of a Control by Using CurrencyManager



In this demonstration, you will see how to maintain the currency of a control by using CurrencyManager

**Introduction**

In this demonstration, you will see how to maintain the currency of a control by using CurrencyManager.

**Instructions**

► **Create a Windows-based application project**

1. Open Visual Studio .NET.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **New Project** dialog box, click **Windows Application**, name it **BindingContextNavigation**, and then click **OK**.

► **Use Data Form Wizard to create a data bound form**

1. On the **File** menu, click **Add New Item**.
2. In the **Add New Item** dialog box, in the **Templates** section, click **Data Form Wizard**, and then click **Open**.
3. In the opening page of Data Form Wizard, click **Next**.
4. Specify the name of the new dataset as **OrdersDataSet**, and click **Next**.
5. Select a connection to the **Northwind** database in the database connection drop-down list. If a connection does not exist, create a new connection. Click **Next**.
6. Add the **Orders** and **Order Details** tables by selecting them in the **Available Items** tree, click the **Add (>)** button, and then click **Next**.
7. In the **Name** box, type **OrderOrderDetailsRelation**

8. Set the following settings.

| List | Setting |
|---|---|
| Parent table | Orders |
| Child table | Order Details |
| Keys (for both tables) | OrderID |

9. Click the **Add (>)** button, and then click **Next**.
10. Ensure that the **Orders** table is set as the **Master or single table** and **Order Details** is set as the **Detail table**, and then click **Next**.
11. Select **Single record in individual controls**, and then click **Finish**.

► **Display and discuss the code generated for the navigation buttons**

1. Locate the **Click** event of the **btnNavNext** button in the Code Editor of DataForm1, and discuss the following line of code:

```
this.BindingContext[objOrdersDataSet,"Orders"].Position =
(this.BindingContext[objOrdersDataSet,"Orders"].Position +
1);
```

2. Locate the **objOrdersDataSet_PositionChanged()** procedure, and discuss the following line of code:

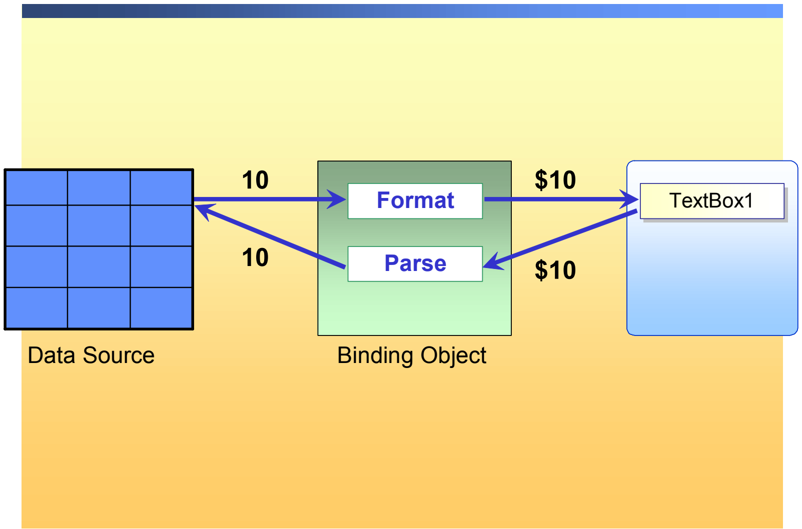
```
this.lblNavLocation.Text =
   ((((this.BindingContext[objOrdersDataSet,
   "Orders"].Position + 1)).ToString() + " of  ")  +
   this.BindingContext[objOrdersDataSet,
   "Orders"].Count.ToString());
```

3. Locate the **Click** event of the **btnLast** button in the Code Editor of DataForm1, and discuss the following line of code:

```
this.BindingContext[objOrdersDataSet,"Orders"].Position =
   (this.objOrdersDataSet.Tables["Orders"].Rows.Count - 1);
```

4. In Solution Explorer, right-click the **BindingContextNavigation** project, and then click **Properties**.
5. Add code to the **Load** event of **Form1** to display **DataForm1**.
6. Press F5 to compile and run the application, and then demonstrate the use of the navigation buttons.
7. Close the application.

# How to Format and Parse Data Bound Values



**Introduction**

Often the data retrieved from a data source may not match the data type accepted by a control in your application. The **Binding** object provides two methods—**Format** and **Parse**—to ensure that the data received and sent back to the data source is of the right data type.

**Format and Parse events**

The **Format** event occurs both when data is pushed from the data source into the control and when the data is pulled from the control into the data source. When the data is pushed from the data source into the control, the **Binding** object uses the **Format** event to put the formatted data into the control. When the data is pushed from the control into the data source, the **Binding** first parses the data by using the **Parse** event and then formats the data and sends it to the data source.

You can use the **Format** and **Parse** events to create custom formats for displaying data. For example, if the data in a table is of type **Decimal**, you can display the data in the local currency format by setting the **Value** property of the **ConvertEventArgs** object to the formatted value in the **Format** event. Consequently, you must remove the format from the displayed value in the **Parse** event.

The **Format** event occurs whenever the current value of the **BindingManagerBase** changes, which includes:

- The first time the property is bound.
- Any time the **Position** changes.
- Whenever the data-bound list is sorted or filtered, which is accomplished when a **DataView** supplies the list.

The **Parse** event occurs:

- After the **Validated** event of the **Control** object occurs.
- When the **EndCurrentEdit** method of the **BindingManagerBase** is called.
- When the current object of the **BindingManagerBase** changes (in other words, when the **Position** changes).

### Procedure: Using the Format and Parse events

The following code creates event delegates for the **Parse** and **Format** events of a **Binding** object and then uses event procedures to format the data between a **String** and a **Decimal**:

```
private void BindControl()
{
   Binding priceBinding = new
       Binding("Text",titlesDataSet1,"titles.price");
   priceBinding.Format += new
       ConvertEventHandler(this.FormatDecimalToString);
   priceBinding.Parse += new
       ConvertEventHandler(this.ParseStringToDecimal);
   PriceTextBox.DataBindings.Add(priceBinding);
}

private void FormatDecimalToString(object sender,
ConvertEventArgs convertArgs)
{
   convertArgs.Value =
       Convert.ToDecimal(convertArgs.Value).ToString("c");
}

private void ParseStringToDecimal(object sender,
ConvertEventArgs convertArgs)
{
   convertArgs.Value =
       Decimal.Parse(convertArgs.Value.ToString(),
       NumberStyles.Currency);
}
```

# Practice: Formatting Data Bound Controls

- **In this practice, you will**
  - Configure the **SQLConnection** control to connect to the database
  - Create an event procedure for the **Parse** event
  - Create an event delegate for the **Parse** event

Begin reviewing the objectives for this practice activity

### Introduction

In this practice, you will add event handlers to the **Format** and **Parse** events of a **Binding** object and then use the **ConvertEventArgs** class to apply appropriate formatting to the data bound value.

### Instructions

► **Open the practice project**

1. Use Windows Explorer to browse to *install_folder*\Practices\Mod04\Mod04_04\Starter.
2. Double-click the FormatParse.sln solution file to open the project.

► **Create an event procedure invoked by the Parse event that converts the Value of ConvertEventArgs from a String to a Decimal**

1. Show TODO comments in the Task List.

   To show TODO comments, click the **View** menu, point to **Show Tasks**, and then click **All**.
2. Locate TODO: 1 in the Code Editor of Form1.
3. Under TODO: 1, create a procedure called *ParseStringToDecimal* that accepts two arguments: *sender* as an **Object** and *convertArgs* as a **ConvertEventArgs**.
4. Assign the **Value** of *convertArgs* to a String named *stringValue*.

5. Call the **Parse** method of the **Decimal** object and pass the two arguments: *stringValue* as the string to be formatted and **NumberStyles.Currency** as the NumberStyles constant.

6. Assign the results of the **Parse** method to **Value** property of *convertArgs.* Your code should look like this:

   ```
   private void ParseStringToDecimal(object sender,
   ConvertEventArgs convertArgs)
   {
      convertArgs.Value =
          Decimal.Parse(convertArgs.Value.ToString(),
          NumberStyles.Currency);
   }
   ```

   The **Parse** method converts the String representation of a number in a specified style to its decimal equivalent by using the specified formatting style.

### ► Create an event delegate for the Parse event that references the FormatDecimalToString procedure

1. Locate TODO: 2 in the Code Editor of Form1.

2. Under TODO: 2, use **AddHandler** to create an event delegate for the **Parse** event of the **priceBinding** object that references the **ParseStringToDecimal** procedure. Your code should look like this:

   ```
   priceBinding.Parse += new
   ConvertEventHandler(this.ParseStringToDecimal);
   ```

3. The *priceBinding* object is a **Binding** object that is added to the **DataBindings** collection of **PriceTextBox**. Review the following code:

   ```
   Binding priceBinding = new
   Binding("Text",titlesDataSet1,"titles.price");
   PriceTextBox.DataBindings.Add(priceBinding);
   ```

### ► Test the application

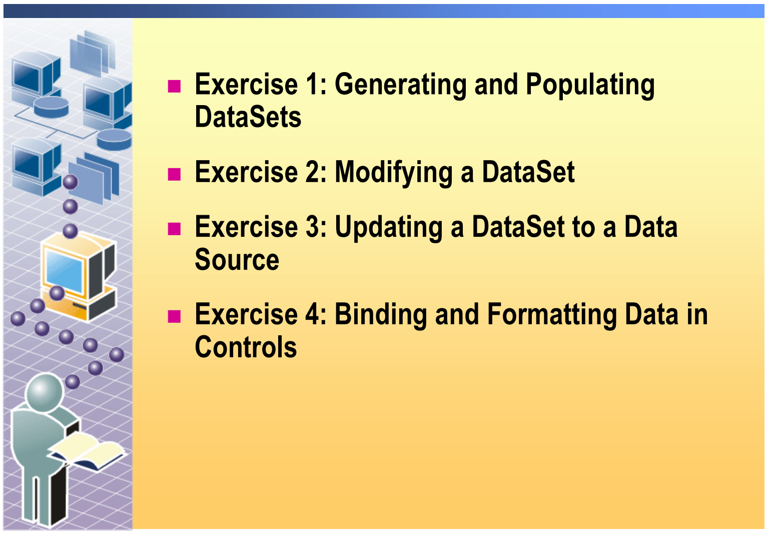
1. Press F5 to compile and run the application.

2. Select different titles in the **Book Titles** list, and review how the **Price** text box is updated and formatted.

3. Use the **Price** text box to change the price of a book, and then click the **Debug.Write Value** button.

4. Display and review the results in the Output window of the Visual Studio.NET IDE.

# Lab 4.1: Accessing Data by Using ADO.NET



- Exercise 1: Generating and Populating DataSets
- Exercise 2: Modifying a DataSet
- Exercise 3: Updating a DataSet to a Data Source
- Exercise 4: Binding and Formatting Data in Controls

**Objectives**

After completing this lab, you will have demonstrated your ability to:

- Add and configure ADO.NET objects by using Component Designer.
- Configure DataAdapters by using Data Adapter Configuration Wizard.
- Generate typed datasets.
- Populate a dataset from data in a database.
- Create relationships between tables in a dataset.
- Update data to a data source by using typed datasets.
- Bind data to controls.
- Format the data bound to controls.

**Note** This lab focuses on the concepts in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C#™ .NET).* As a result, this lab may not comply with Microsoft security recommendations.

**Prerequisites**

Before working on this lab, you must have:

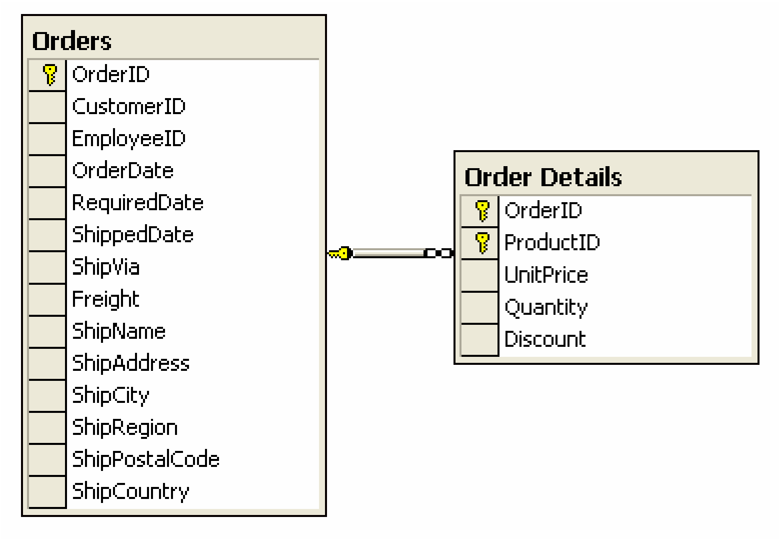
- The knowledge and skills to develop a simple Windows Forms application by using a Microsoft Visual Studio® .NET–compatible programming language.
- The knowledge and skills to use Data Adapter Configuration Wizard in the Visual Studio .NET IDE.
- The knowledge and skills to access and update a database from a Visual Studio .NET–based application.

- The knowledge and skills to create Relations between tables in a dataset in a Visual Studio .NET–based application.
- The knowledge and skills to bind data to controls in a Visual Studio .NET–based application.
- The knowledge and skills to format data bound to controls in a Visual Studio .NET–based application.

**Scenario**

You are an application developer at a trading company called Northwind Traders. The department that you work in is developing a Purchase Order application that will be used by the Northwind Traders sales force. You are responsible for completing data access code that will use ADO.NET objects to access the Northwind database. Much of the code was written by your colleagues, and you are responsible for creating the code that will populate and update the Order and Order Details tables. You are also responsible for writing the data binding code in the **OrderItemControl** control.

The Order and Order Details tables have a one-to-many relationship. To write the code that populates and updates these tables, you must understand the relationship between the tables and the fields involved. The following illustration represents the relations and fields in the Order and Order Details tables.



**Estimated time to complete this lab: 45 minutes**

# Exercise 1
# Generating and Populating DataSets

In this exercise, you will create a **SQLDataAdapter** by using Data Adapter Configuration Wizard, and populate a dataset with data from a database at run time. You will use Component Designer to add and configure ADO.NET objects. You can use Component Designer to add subcomponents to a class, configure them, and code their events.

## Scenario

You begin to build the data access features into the Purchase Order application. Your development team has built some of the preliminary data access code that you need to complete. Your responsibility is to create a **DataAdapter** that can access and update the Order table of the Northwind database.

The application uses employee information to load and create order entries. Each order is associated with a single employee. The Orders table includes an EmployeeID column that maps to the EmployeeID column in the Employees table. The **DataAdapter** that you create will use the value of the *employeeID* variable to determine which rows to return. You decide to use Data Adapter Configuration Wizard to configure the **SELECT** command of the **DataAdapter** and allow it to generate the **Update** command for you.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab04_1\Ex01\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab04_1\Ex01\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open Visual Studio .NET, and open the OrderApplication.sln file. To open the solution file, browse to *install_folder*\Labfiles\Lab04_1\Ex01\Starter\OrderApplication. | **a.** For more information about opening a project file, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. |
| **2.** Add a **SQLDataAdapter** control to **OrderApplicationData Class**. In Data Adapter Configuration Wizard, use the **NorthwindConnection** information, and create the following SQL Select statement:<br>SELECT * FROM Orders WHERE (EmployeeID = @EmployeeID)<br>• Ensure that the wizard created the **Insert**, **Update**, and **Delete** commands.<br>• Change the name of the new **SQLDataAdapter** to **OrdersDataAdapter**. | **a.** For more information about adding controls to Component Designer or by using Data Adapter Configuration Wizard, see the following resources:<br>• Lesson: Adding ADO.NET Objects to and Configuring ADO.NET Objects in a Windows Forms Application in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Adding ADO.NET Objects to and Configuring ADO.NET Objects in a Windows Forms Application in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Component Designer** or the phrase **Data Adapter Configuration Wizard**. |
| **3.** Use the Task List to locate the code section 'TODO: 1', and create a local variable to hold an instance of **NorthwindDataSet** and name it **tempDataSet**. | Additional information is not necessary for this task. |

| Tasks | Additional information |
|---|---|
| **4.** Use the Task List to locate the code section 'TODO: 2', and add **@employeeID** (the global employee ID) to the **Parameters** collection of the **SelectCommand** property of **OrdersDataAdapter** and **OrderDetailsDataAdapter**. | **a.** For more information about assigning values to the **Parameters** collection of **Command** objects, see the following resources:<br>• Lesson: Accessing and Modifying Data by Using DataSets in Module 4, "Using Data in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Populating and Updating DataSets in Module 4, "Using Data in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET SDK documentation. Search by using the phrase **Using Parameters with a DataAdapter**. |
| **5.** Use the Task List to locate the code section TODO: 3', and call the **Fill** method of **OrdersDataAdapter** and **OrderDetailsDataAdapter** to fill the **Orders** and **OrderDetails** tables of **tempDataSet**. | **a.** For more information about filling **DataSets** with data by using a **DataAdapter**, see the following resources:<br>• Lesson: Accessing and Modifying Data by Using DataSets in Module 4, "Using Data in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Populating and Updating DataSets in Module 4, "Using Data in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The.NET Framework SDK documentation. For additional information about assigning values to the **Parameters** collection of **Command** objects, search by using the phrase **Populating a DataSet from a DataAdapter**. |
| **6.** Place a breakpoint at the beginning of the **RefreshLocalData** procedure. Compile and run the application. From the **Data** menu, click **Refresh Data**, and step through each line of code. Locate and review the contents of the NorthwindData.XML file, which is in the most recent folder version folder of C:\Documents and Settings\All Users\ Application Data\ Northwind Traders\ PurchaseOrder\. | **a.** For more information about building and debugging your applications, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Default and Custom Builds** and **Using the Debugger**. |

# Exercise 2
# Modifying a DataSet

In this exercise, you will review the code that creates data tables objects, insert new data rows, and append the information to an existing dataset.

## Scenario

The Orders table uses an auto incrementing column named **OrderID** to generate primary keys when new rows are created in the database. This value is also generated by datasets when a new row is inserted in the Orders table. The Purchase Order application uses a global dataset named **pendingOrdersData** to store all the orders until they are submitted to the database. Each order is associated with one or more order details. The Orders table includes an OrderID column that maps to the OrderID column of the Order Details table.

One of your colleagues has written a procedure called **SaveOrders** that creates new rows in the Orders table, captures the value of the OrderID generated by the dataset, and uses it to create rows in the Order Details table. After all of the rows in the Order Details table are created, the procedure persists the order information in an XML file named PendingOrders.XML. You decide to review the code more closely to understand how it works.

There are solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab04_1\Ex02\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open Visual Studio .NET, and open the OrderApplication.sln file. To open the solution file, browse to *install_folder*\Labfiles\Lab04_1\Ex02\Solution\OrderApplication. | **a.** For more information about opening a project file, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. |
| **2.** Place a breakpoint at the beginning of the **SaveOrders** procedure in MainForm. Compile and run the application. On the toolbar, click the **Refresh** button, and choose an employee name. To create different order items, click the **New Order Item** button and then click the **Save Order** button. Step over each line of code by pressing F10 and review the flow of the code.<br>When reviewing the code, look for the following things:<br>• The **SaveOrders** procedure creates a new row in the Orders table and assigns the columns to values from the controls of **MainForm**. When the row is added to the table, the auto incrementing OrderID generated by the **DataSet** is stored in a variable named *clientOrderID*.<br>• The **SaveOrders** procedure then creates rows in the Order Details table and uses the value of *clientOrderID* to populate the OrderID column for each row inserted into the Order Details table. | **a.** For more information about building and debugging your applications, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Default and Custom Builds** and **Using the Debugger**. |

# Exercise 3
# Updating a DataSet to a Data Source

In this exercise, you will create a relationship between two tables to create a typed dataset and then update the changes from the dataset to a DataSource.

## Scenario

In the Northwind database, each order is associated with one or more order details. The Orders table includes an OrderID column that maps to the OrderID column of the Order Details table. When you create a new row in the Orders table, the **SaveOrders** procedure ensures that all child rows in the Order Details table have the same OrderID as the parent in the Orders table. When you update the dataset to a data source, you must first update the Orders table and then the Order Details table. When you update a row in the Orders table, however, a new OrderID will be generated by the database that might not match the child values in the Order Details table. This causes an error when you attempt to update the Order Details table.

One way to resolve this problem is to write code that uses **DataSet** events to capture the new OrderID generated by the database and assign it to all child rows in the Order Details table. However, other options are available when you use typed datasets and DataAdapters for each table in a database. Instead of writing code for **DataSet** events, you can use the **Refresh the DataSet** option in Data Adapter Configuration Wizard. With this option enabled, the wizard adds a **SELECT** statement after the **Insert** and **Update** statements to retrieve the Identity column generated by the database.

When you create a relation between tables by using XML Designer, you can choose how table relationships are handled when the parent table is modified. For example, if you set the **Delete Rule** to **Cascade** (the default), and you delete rows in a parent table, all the related rows in the child table are also deleted. When you create a relation between the Orders and Order Details tables, it is important that you set the value of the **Update Rule** option to **Cascade** (the default) so that any changes made to a parent table are cascaded to the child tables.

When you update a row in the Orders table, the DataAdapter returns the OrderID value generated by the database and changes the value in the Orders table of the dataset. With the cascading update rule, all the child rows in the Order Details table of the dataset will automatically be changed to the new OrderID of the parent. Now when you update the Order Details table to the database it will have the same OrderID as the parent.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab04_1\Ex03\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab04_1\Ex03\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open Visual Studio .NET, and open the OrderApplication.sln file. To open the solution file, browse to *install_folder*\Labfiles\Lab04_1\Ex03\Starter\OrderApplication. | **a.** For more information about opening a project file, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. |
| **2.** Use XML Designer to create a relation between the Orders and Order Details tables in the NorthwindDataSet.xsd file. | **a.** For more information about using XML Designer, see the following resources:<br>• Lesson: Accessing and Modifying Data by Using DataSets in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Demonstration: Creating Database Schema by Using the XML Schema Designer in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **XML Designer**. |
| **3.** Use the Task List to locate the code section 'TODO: 1' in the code view of OrderApplicationDataClass.cs, and update the database by using the **Update** method of **OrdersDataAdapter** and **OrderDetailsDataAdapter**.<br>The Cascading Update rule enforced by the relation between the two tables should automatically retrieve the OrderID generated by the server. When an Order row is created, it is assigned to related rows in the Order Details table on the client computer. | **a.** For more information about updating a data source, see the following resources:<br>• Lesson: Accessing and Modifying Data by Using DataSets in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Populating and Updating DataSets in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Updating the Database with a DataAdapter and the DataSet**. |

| Tasks | Detailed steps |
|---|---|
| **4.** Compile and run the application . On the toolbar, click the **Refresh** button, and choose an employee name. To create an order, click the **New Order Item** button. To save the order, click the **Save Order** button, and then click the **Submit** button on the Toolbar. Use Server Explorer to verify that the order was saved to the database. | **a.** For more information about building and debugging your applications and using Server Explorer, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Default and Custom Builds, Using the Debugger**, and **Server Explorer Window**. |

# Exercise 4
# Binding and Formatting Data in Controls

In this exercise, you will bind data to controls and format the data by using the events of the **Binding** class.

## Scenario

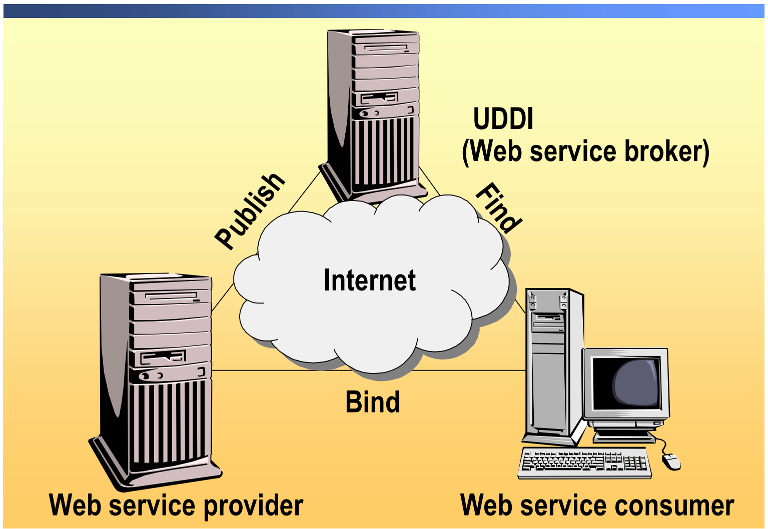
The **OrderItemControl** control is under development and requires code to bind constituent controls to data from the Products table. A method within the control called **GetProductData** has been created, and it accepts a parameter named *productsTable* that contains data from the Products table. You need to bind data from this table to the constituent controls. You also need to convert the values in the Price column to currency when you bind them to UnitPriceTextBox control.

There are starter and solution files associated with this exercise. Browse to the *install_folder*\Labfiles\Lab04_1\Ex04\Starter folder to find the starter files, and browse to the *install_folder*\Labfiles\Lab04_1\Ex04\Solution folder to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Detailed steps |
|---|---|
| **1.** Open Visual Studio .NET, and open the OrderApplication.sln file. To open the solution file, browse to *install_folder*\Labfiles\Lab04_1\Ex04\Starter\OrderApplication. | **a.** For more information about opening a project file, see the following resources<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. |
| **2.** Use the Task List to locate the code section 'TODO: 1 in the code window of OrderItemControl.cs' , and assign the **DataSource** of **ProductNameComboBox** to the DataTable passed to the GetProductData procedure. Assign the **DisplayMember** and **ValueMember** of **ProductNameComboBox** to the ProductName and ProductID columns, respectively. | **a.** For more information about binding controls to data, see the following resources:<br>• Lesson: Binding Data to Controls in Module 4, "Using Data in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*<br>• Practice: Binding Controls to Data in Module 4, "Using Data in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*<br>• The.NET Framework SDK documentation. For additional information about binding controls to data, search by using the phrase **Data Binding and Windows Forms**. |

| Tasks | Detailed steps |
|---|---|
| **3.** Use the Task List to locate the code section 'TODO: 2', and declare a **Binding** object, and use it to bind the **Text** property of **UnitPriceTextBox** to the UnitPrice column of the DataTable passed to the **GetProductData** procedure. | **a.** For more information about binding data to controls, see the following resources:<br>• Lesson: Binding Data to Controls in Module 4, "Using Data in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Binding Controls to Data in Module 4, "Using Data in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The.NET Framework SDK documentation. For additional information about binding controls to data, search by using the phrase **Data Binding and Windows Forms**. |
| **4.** Use the Task List to locate the code section 'TODO: 3', and create an event procedure named **DecimalToCurrency** that converts the **ConvertEventArgs** event argument to a currency format. | **a.** For more information about creating event procedures for **Binding** objects, see the following resources:<br>• Lesson: Binding Data to Controls in Module 4, "Using Data in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Formatting Data Bound Controls in Module 4, "Using Data in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The.NET Framework SDK documentation. For additional information about binding controls to data, search by using the phrase **Data Binding and Windows Forms**. |
| **5.** Use the Task List to locate the code section 'TODO: 4', and create an event handler for the **Format** event of the **Binding** object created in step 4. | **a.** For more information about creating event handlers, see the following resources:<br>• Lesson: Creating an Event Handler for a Control in Module 2, "Working with Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Creating an Event Handler for a Control in Module 2, "Working with Controls," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET SDK documentation. For additional information about binding controls to data, search by using the phrase **Creating Event Handlers at Run Time for Windows Forms**. |
| **6.** Compile and run the application. On the toolbar, click the **Refresh** button, and choose an employee name. To populate the **OrderItemControl** control, click the **New Order Item** button. Navigate through the products by using the **ProductNameComboBox** control. | **a.** For more information about building and debugging your applications, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Default and Custom Builds** and **Using the Debugger**. |

# Lesson: Overview of XML Web Services



**Introduction**

XML Web services enable the exchange of data and the remote invocation of application logic by using XML messaging to move data through firewalls and between heterogeneous systems.

XML Web services are accessible from just about any kind of application, including other XML Web services, Web applications, applications for the Windows operating system, and console applications. The only requirement is that the client must be able to send, receive, and process messages to and from the XML Web service.

An XML Web service can be used either internally by a single application or exposed externally over the Internet for use by any number of applications by using a service broker, which is a node that hosts a Universal Description, Discovery and Integration (UDDI) registry. Because it is accessible through a standard interface, an XML Web service allows completely different systems to work together as a single web of computation.

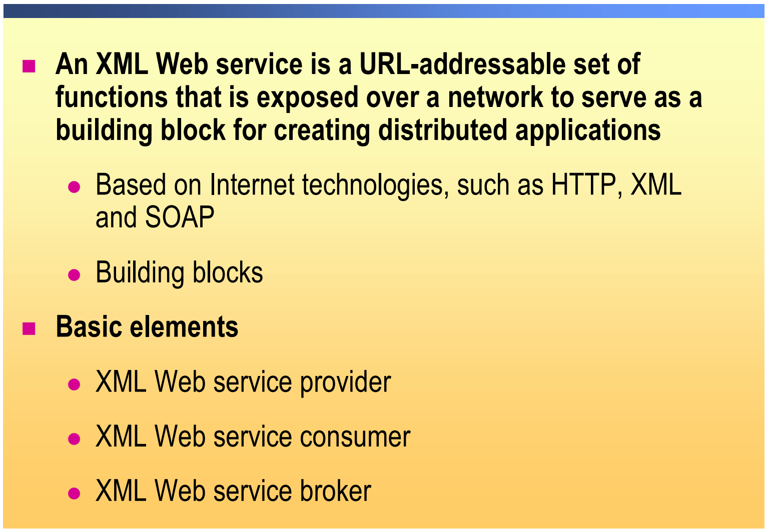
This lesson presents an overview of the XML Web services programming model and shows how to create applications that use XML Web services.

**Lesson objectives**

After completing this lesson, you will be able to:

- Describe the XML Web services model.
- Describe the roles of HTTP, Simple Object Access Protocol (SOAP), and XML in the Web services model.

# What Are XML Web Services?



- **An XML Web service is a URL-addressable set of functions that is exposed over a network to serve as a building block for creating distributed applications**
  - Based on Internet technologies, such as HTTP, XML and SOAP
  - Building blocks
- **Basic elements**
  - XML Web service provider
  - XML Web service consumer
  - XML Web service broker

**Introduction**

In many cases, applications must access remote data or functionality. One way to make this happen is to call XML Web services.

**Definition**

An XML Web service is an URL-addressable set of functions that is exposed over a network to serve as a building block for creating distributed applications.

**Example of an XML Web service**

An example of an XML Web service is Microsoft Passport. Passport provides authentication services and all of its functionality is accessible through HTTP requests.

The foundations for XML Web services are HTTP, XML, and Simple Object Access Protocol (SOAP, a lightweight HTTP- and XML-based protocol used for information exchange). The development of these technologies is governed by the World Wide Web Consortium (W3C).

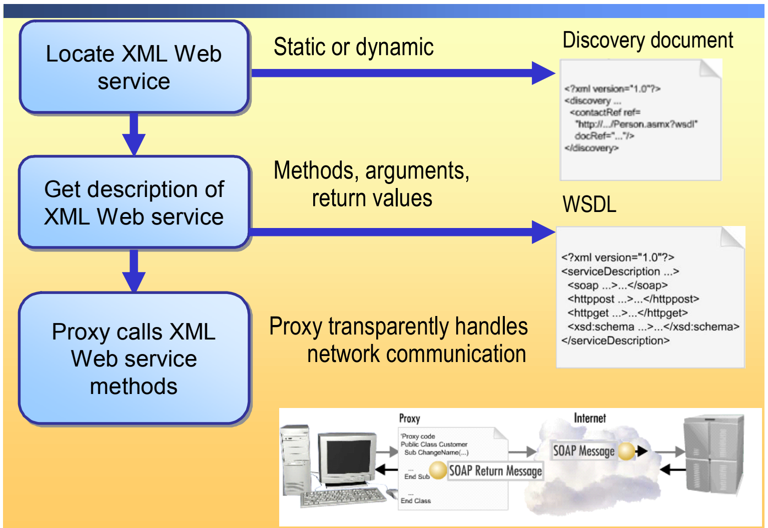
Like components, XML Web services are black boxes. They encapsulate the implementation and provide an interface for communicating with the Web service. Therefore, you can use XML Web services as building blocks for applications.

**Basic elements of the XML Web service model**

The basic elements of the XML Web service model are:

- The XML Web service broker, which is a network node hosting a global registry of available Web services, much like a telephone directory.
- The XML Web service provider, which is a network node hosting a Web service.
- The XML Web service consumer, which is a network node hosting any client that can communicate by using HTTP. The clients include browsers, console applications, and traditional graphical user interface (GUI) applications.

# How Do XML Web Services Work?



Locate XML Web service

Static or dynamic

Discovery document

```
<?xml version="1.0"?>
<discovery ...
  <contactRef ref=
    "http://.../Person.asmx?wsdl"
    docRef="..."/>
</discovery>
```

Get description of XML Web service

Methods, arguments, return values

WSDL

```
<?xml version="1.0"?>
<serviceDescription ...>
  <soap ...>...</soap>
  <httppost ...>...</httppost>
  <httpget ...>...</httpget>
  <xsd:schema ...>...</xsd:schema>
</serviceDescription>
```

Proxy calls XML Web service methods

Proxy transparently handles network communication

```
'Proxy code
Public Class Customer
  Sub ChangeName(...)
  
   ...
  End Sub
  
   ...
End Class
```

****************************ILLEGAL FOR NON-TRAINER USE*****************************

**Introduction**

One of the core characteristics of an XML Web service is the high degree of abstraction that exists between the implementation and consumption of a service. By using XML-based messaging as the mechanism by which the service is created and accessed, both the XML Web service client and the XML Web service provider are freed from needing any knowledge of each other beyond inputs, outputs, and locations.

**Discovering the location of the XML Web service**

You can discover the location of an XML Web service either statically or dynamically:

- If you know the location of the XML Web service you want to call, you explicitly provide it in your application. This is called *static discovery*.
- If you do not know the location of XML Web service you want to call, then you must use dynamic discovery methods.

For more information about static and dynamic discovery of XML Web services, see the lesson titled Creating a Simple XML Web Services Client in Module 4, "Using Data in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C#™ .NET)*.

**XML Web service description**

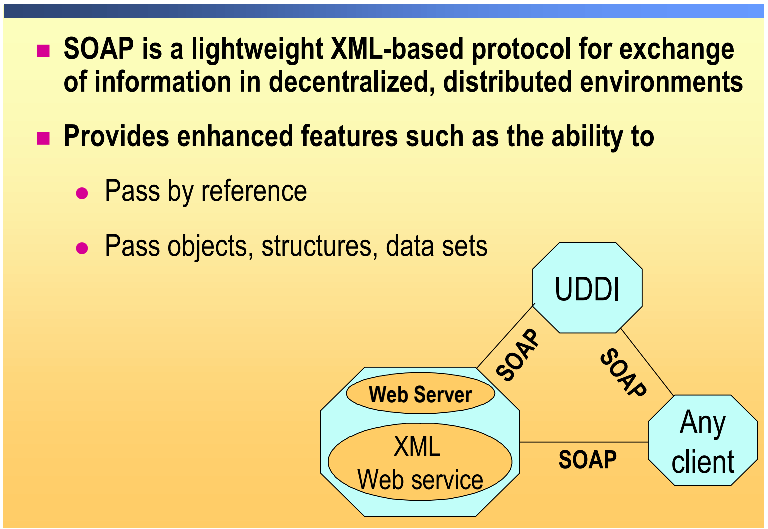
An XML Web service can be used either internally by a single application or exposed externally over the Internet for use by any number of applications. Because it is accessible through a standard interface, an XML Web service allows heterogeneous systems to work together as a single web of computation.

**Proxy calls XML Web service methods**

An XML Web service consumer must be able to construct the messages that are sent to a Web service and parse the messages that are received from a Web service. Manually writing the code to construct and parse the messages is time-consuming and prone to error. It is better to encapsulate this code in a class that can be reused. A class like this is called a *proxy class*.

When you reference an XML Web service from a consumer application, the .NET Framework generates a proxy for you.

# What is SOAP?



- SOAP is a lightweight XML-based protocol for exchange of information in decentralized, distributed environments
- Provides enhanced features such as the ability to
  - Pass by reference
  - Pass objects, structures, data sets

****************************ILLEGAL FOR NON-TRAINER USE******************************

**Introduction**

XML Web services use SOAP as the protocol for network messages.

**Definition**

Simple Object Access Protocol (SOAP) is a lightweight XML-based protocol for exchange of information in decentralized, distributed environments.

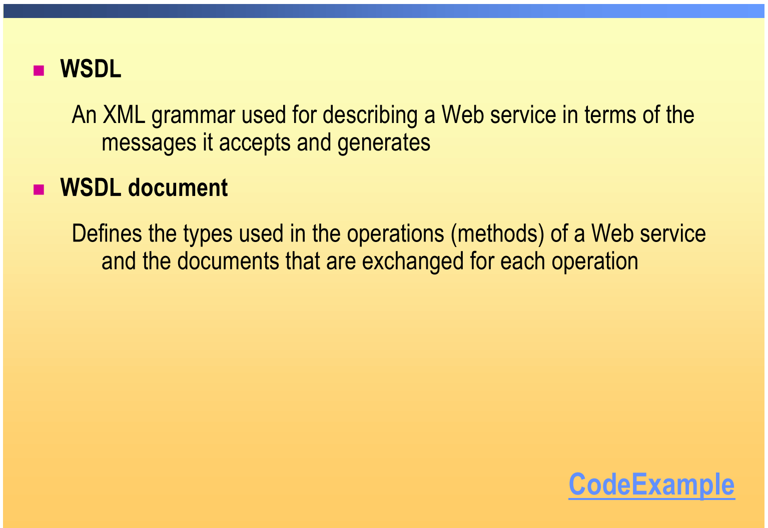
Soap messages consist of two main parts:

- An envelope for handling extensibility and modularity.
- An encoding mechanism for representing types in an envelope.

**The advantages of using SOAP**

By using SOAP as an underlying protocol for XML, messages can pass data by reference and contain complex structures such as objects, structures, and data sets.

# What is WSDL?



- **WSDL**

  An XML grammar used for describing a Web service in terms of the messages it accepts and generates

- **WSDL document**

  Defines the types used in the operations (methods) of a Web service and the documents that are exchanged for each operation

CodeExample

**Introduction**

To be able to use or consume a Web service, you must first know how to interact with it.

**Definition**

Web Services Description Language (WSDL) is an XML grammar that is used for describing a Web service in terms of the messages it accepts and generates. In other words, a WSDL file acts as a contract between a Web service consumer (client) and a Web service.

A WSDL document defines the types used in the operations (methods) of a Web service and the documents that are exchanged for each operation. Then, these definitions are associated with a network protocol and grouped into messages to define an endpoint (the network location of a Web service). WSDL can describe endpoints and their operations without specifying the message formats or the network protocols to which an endpoint is bound.

**Structure of a WSDL document**

As a Web service consumer, it is important that you are familiar with WSDL to understand the contract defined in a WSDL document.

A WSDL document is just a list of definitions. In a WSDL file, the root element is named **definitions**. This element contains five primary child elements that define the Web service. The following table describes the five elements that appear in the **definitions** element in a WSDL file in the order specified.

| Element | Description |
|---|---|
| types | Defines the various data types used to exchange messages. |
| message | Describes the messages to be communicated. |
| portType | Identifies a set of operations and the messages involved with each of those operations. |
| binding | Specifies the protocol details for various service operations and describes how to map the abstract content of these messages into a concrete format. |
| service | Groups a set of related ports together. |

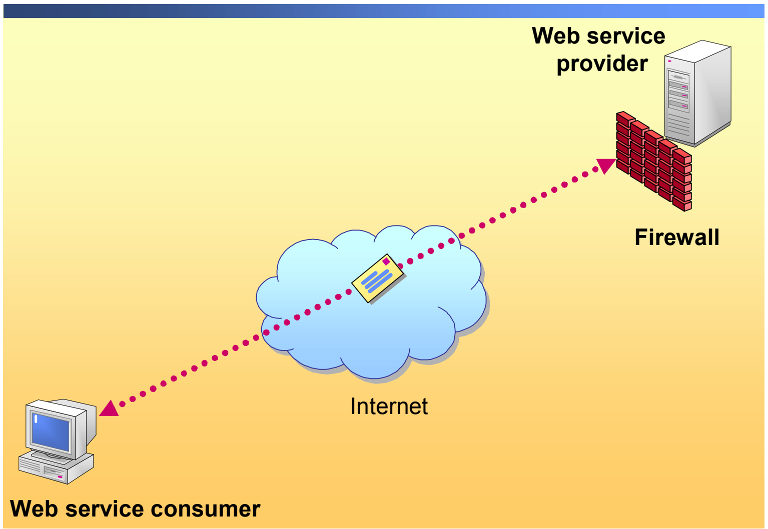
**Example**

The following code is an example of some of the contents of the WSDL document for the ExpenseReportWebService XML Web service that your client application will call in Lab 4.2: Calling an XML Web Service in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*:

```
<?xml version="1.0" encoding="utf-8" ?>
- <definitions
xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:s="http://www.w3.org/2001/XMLSchema"
xmlns:s0="http://localhost/ExpenseReportWebService"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
targetNamespace="http://localhost/ExpenseReportWebService"
xmlns="http://schemas.xmlsoap.org/wsdl/">
- <types>
- <s:schema elementFormDefault="qualified"
targetNamespace="http://localhost/ExpenseReportWebService">
<s:import namespace="http://www.w3.org/2001/XMLSchema" />
- <s:element name="AuthenticateUser">
- <s:complexType>
- <s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="username"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="password"
type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="passwordToken"
type="s:string" />
</s:sequence>
</s:complexType>
</s:element>
 ...
```

Usually this information is transparent to the developer. The XML Web service description is just used by the .NET Framework when you add a reference to an XML Web service to your client application. You will see a more user-friendly display of the details of the methods, arguments, and return values associated with an XML Web service that you reference in the lesson titled Creating a Simple XML Web Services Client in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET*).

# Lesson: Creating a Simple XML Web Services Client



**Introduction**

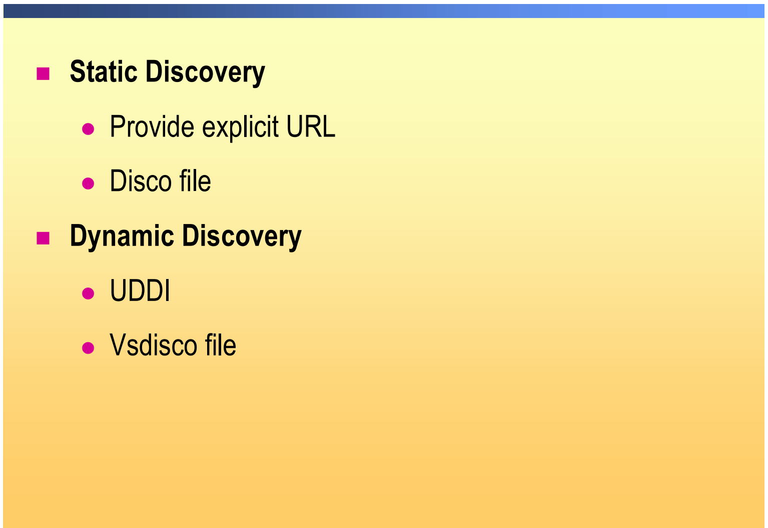
This lesson shows how to create and test a client application that uses XML Web services.

**Note** For more information about how to make asynchronous calls to XML Web services to avoid blocking the user interface, see Module 7, “Asynchronous Programming,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.

**Lesson objective**

After completing this lesson, you will be able to create and test a simple XML Web service client application.

# How to Locate an XML Web Service



- **Static Discovery**
  - Provide explicit URL
  - Disco file
- **Dynamic Discovery**
  - UDDI
  - Vsdisco file

### Introduction

The process by which you locate a Web service and its descriptions and learn how to interact with it is known as Web service discovery.

A Web service provider can make discovery information available to Web service consumer developers. It can do this by either statically or dynamically generating a document that contains a link to the WSDL document for all the Web services hosted by the provider.

### Static discovery

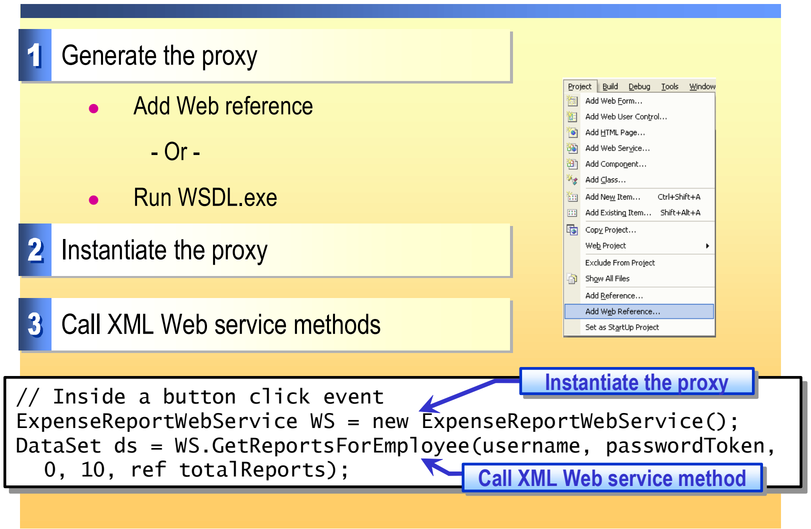
Static discovery is possible when the location of a discovery document (usually with the extension .disco) is already known. Static discovery involves providing an explicit URL for an XML Web service or retrieving the discovery document by providing an explicit URL and interpreting its contents.

### Dynamic discovery

Dynamic discovery takes place when all that is known to the consumer is the endpoint of the Web service provider. In this situation, there is no static list of .disco files at the endpoint. Instead, the list of available Web services and the associated service contracts must be dynamically retrieved from a UDDI directory by the consumer.

**Note** A static discovery document usually has the extension .disco and a dynamic discovery document has the extension .vsdisco.

# How to Access an XML Web Service



1 Generate the proxy

- Add Web reference

  - Or -

- Run WSDL.exe

2 Instantiate the proxy

3 Call XML Web service methods

```
// Inside a button click event
ExpenseReportWebService WS = new ExpenseReportWebService();
DataSet ds = WS.GetReportsForEmployee(username, passwordToken,
  0, 10, ref totalReports);
```

## Introduction

Much of the work of calling an XML Web service is done for you by the .NET Framework, but you must add a few lines of code to your application to get the .NET Framework to do the work on your behalf. The following is an overview of the main steps that you must take in your application to call an XML Web service:

1. Generate the proxy that will call the XML Web service methods.

   You can either add a Web reference to your Visual Studio .NET project or call the command line utility WSDL.exe.

2. Instantiate the proxy.

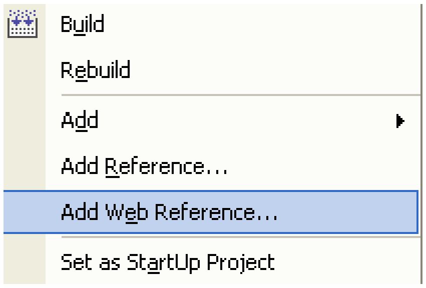
   Based on the description of the XML Web service, instantiate the proxy object that you use to call the XML Web service methods.

3. Call the XML Web service methods that are required by your application.

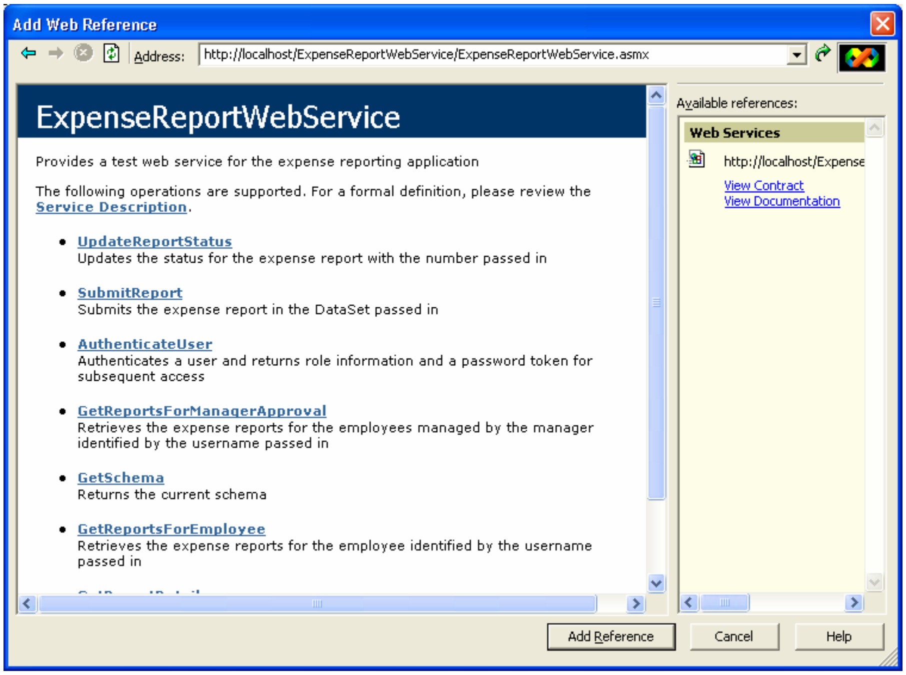
   Based on the description of the XML Web service methods, call them appropriately.

## Procedure: Generating the proxy

In Visual Studio .NET, you can generate a proxy by adding a Web Reference to your client application project.



The following is the documentation for an XML Web service—a user-friendly summary of the WSDL document that contains a short description of the methods available. When you click the **Service Description** hyperlink, you will see the WSDL document.



ExpenseReportWebService

Provides a test web service for the expense reporting application

The following operations are supported. For a formal definition, please review the Service Description.

- UpdateReportStatus
  Updates the status for the expense report with the number passed in
- SubmitReport
  Submits the expense report in the DataSet passed in
- AuthenticateUser
  Authenticates a user and returns role information and a password token for subsequent access
- GetReportsForManagerApproval
  Retrieves the expense reports for the employees managed by the manager identified by the username passed in
- GetSchema
  Returns the current schema
- GetReportsForEmployee
  Retrieves the expense reports for the employee identified by the username passed in

Alternatively, you can use WSDL to generate a proxy. The following code generates a proxy by using the default language (C#) and default protocol (SOAP).

```
wsdl http://www.contoso.com/ExpenseReportWebService/
ExpenseReportWebService.asmx?wsdl
```

### Procedure: Instantiating the proxy

As shown in the code on the slide, the following statement generates the proxy to call the ExpenseReportWebService WS XML Web service:

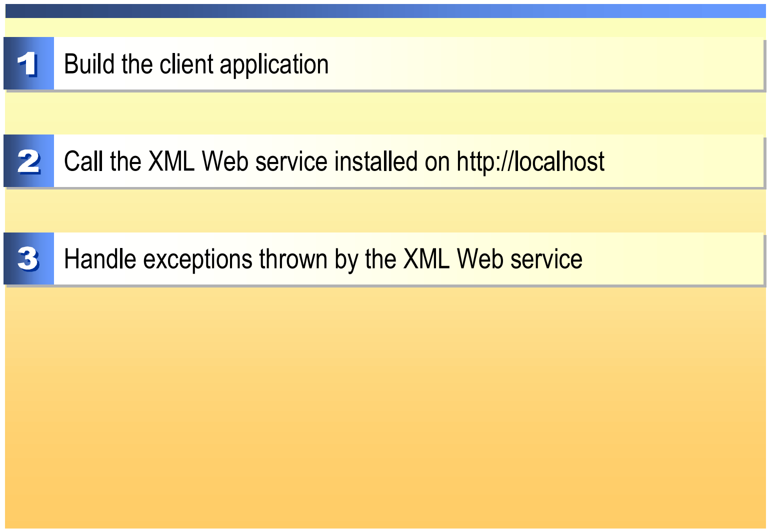
```
ExpenseReportWebService WS = new ExpenseReportWebService();
```

### Procedure: Calling the XML Web service methods

As shown in the code on the slide, you can call the methods of the XML Web service, as appropriate, to get the data that your application needs:

```
DataSet ds = WS.GetReportsForEmployee(username, passwordToken,
   0, 10, ref totalReports);
```

# How to Test an XML Web Services Client



1. Build the client application
2. Call the XML Web service installed on http://localhost
3. Handle exceptions thrown by the XML Web service

## Introduction

After you have developed an XML Web services consumer application, you can test it on your own computer by using a URL that begins with http://localhost. To do this, the XML Web service that you are calling must be installed on your computer. For Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET*), you have ExpenseReportWebService installed to use in the practice called Calling an XML Web Service and in Lab 4.2: Calling an XML Web Service in Module 4, "Using Data in Windows Forms Applications."

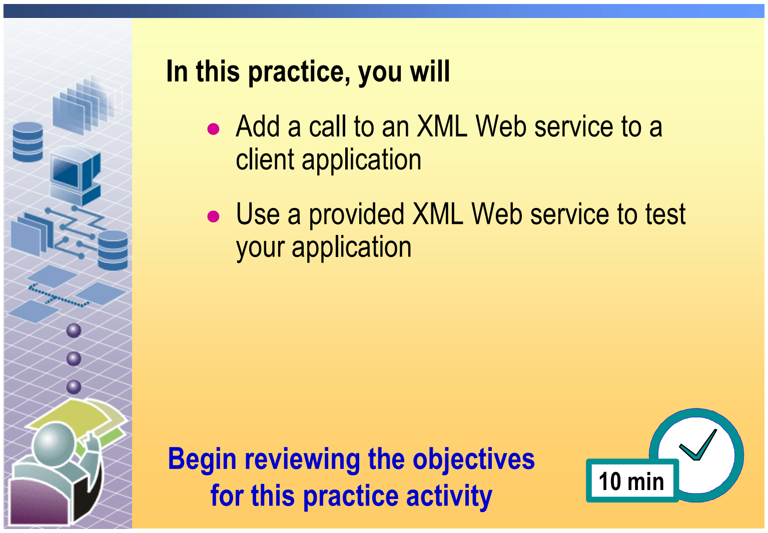
## Calling the XML Web service

If you experience an error when you try to call the XML Web service, you can place a breakpoint in your application at the point where you call a method of the XML Web service and then step through the code to get more details about the problems that are occurring.

## Exceptions

Any exception thrown in the XML Web service call will be returned as a **SoapException**. The **Message** property of the exception contains a text representation of the exception. You can parse this property to provide customized error handling or display it to the user.

# Practice: Calling an XML Web Service



**In this practice, you will**

- Add a call to an XML Web service to a client application
- Use a provided XML Web service to test your application

Begin reviewing the objectives for this practice activity

10 min

**Introduction**

In this practice, you will add a call to an XML Web service to a client application and then use a provided XML Web service to test your application.

**Instructions**

► **Add a call to the Web service**

1. Open the WebServiceClient project in *install_folder*\Practices\Mod04_05\Starter.
2. Open the MainForm form, and view its code.
3. To add a Web reference to the Expense Report Web service, in Solution Explorer, right-click **WebServiceClient**, and then click **Add Web Reference**.
4. In the **Add Web Reference** dialog box, in the **Address** box, type **http://localhost/ExpenseReportWebService/ExpenseReport WebService.asmx**, and then press ENTER.
5. After the Web reference information is loaded, click **Add Reference**.
6. Show TODO comments in the Task List.

   To show TODO comments, click the **View** menu, point to **Show Tasks**, and then click **All**.
7. In the **MainForm** source file, find the first TODO comment near the top of the file. Add the **using** statement for the **XML Web service** namespace.

```
using WebServiceClient.localhost;
```

8. Find the next TODO comment, which instructs you instantiate the XML Web service proxy class. Add the code to instantiate the proxy.

```
WS = new ExpenseReportWebService();
```

9. Find the last TODO comment, which instructs you to add a call to the **GetReportsForEmployee** method of the Web service. Add the code to make the call to the method.

```
WS.GetReportsForEmployee(null, null, 0, 10,
    ref TotalReports);
```

10. Build the project.

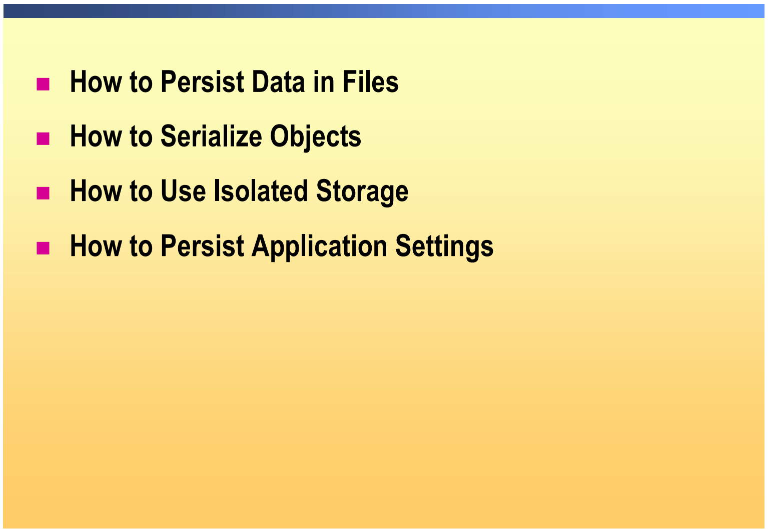
### ► Test the call to the XML Web service

1. Run the application.
2. Click **Call Web Service**.

   After the Web service call completes, a text message will appear in the application.

   If time permits, set a breakpoint in the **CallBtn_Click** method and step through the creation of the Web service proxy and the call to the XML Web service method.

# Lesson: Persisting Data



- How to Persist Data in Files
- How to Serialize Objects
- How to Use Isolated Storage
- How to Persist Application Settings

**Introduction**

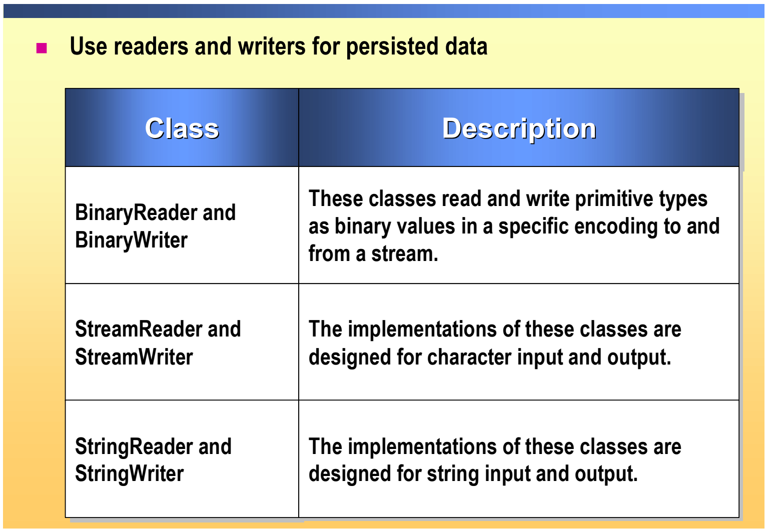
Many applications must persist, or store, data. This lesson presents techniques and locations for persisting data.

**Lesson objectives**

After completing this lesson, you will be able to:

- Persist data to and read data from files.
- Serialize objects.
- Use isolated storage.
- Persist application settings.

# How to Persist Data in Files



- Use readers and writers for persisted data

| Class | Description |
|---|---|
| BinaryReader and BinaryWriter | These classes read and write primitive types as binary values in a specific encoding to and from a stream. |
| StreamReader and StreamWriter | The implementations of these classes are designed for character input and output. |
| StringReader and StringWriter | The implementations of these classes are designed for string input and output. |

## Introduction

Many applications must persist, or store, data. Basic file input/output (I/O) functionality, found in the **System.IO** root, provides the ability to access, store, and manipulate data that is stored in hierarchical file systems that have files that are referenced by using unique paths.

When an application stores data in a file, you must carefully choose the file name and storage location to minimize the possibility that the storage location will be known to another application and, therefore, vulnerable to corruption. Isolated storage provides the means to manage downloaded Web application files to minimize storage conflicts.

## Reading and writing files

The abstract **Stream** class is designed for byte input and output. You can use the reader and writer classes to input and output to streams and strings that use other types.

The following table describes some commonly used **Reader** and **Writer** classes.

| Class | Description |
|---|---|
| **BinaryReader** and **BinaryWriter** | These classes read and write primitive types as binary values in a specific encoding to and from a stream. |
| **StreamReader** and **StreamWriter** | The implementations of these classes are designed for character input and output. |
| **StringReader** and **StringWriter** | The implementations of these classes are designed for string input and output. |

A reader or writer is attached to a stream so that the required types can be read or written to easily.

The following code shows how to write data of type **Integer** to and read from a new, empty file stream that is named Test.data. After creating the data file in the current directory, the **BinaryWriter** class writes the integers 0 through 10 to Test.data. Then, the **BinaryReader** class reads the file and displays the file's content to the console.

```
using System;
using System.IO;

class MyStream {
   private const string FILE_NAME = "Test.data";
   public static void Main(String[] args) {
      // Create the new, empty data file.
      if (File.Exists(FILE_NAME)) {
         Console.WriteLine("{0} already exists!", FILE_NAME);
         return;
      }
      FileStream fs = new FileStream(FILE_NAME,
         FileMode.CreateNew);
      // Create the writer for data.
      BinaryWriter w = new BinaryWriter(fs);
      // Write data to Test.data.
      for (int i = 0; i < 11; i++) {
         w.Write( (int) i);
      }
      w.Close();
      fs.Close();
      // Create the reader for data.
      fs = new FileStream(FILE_NAME, FileMode.Open,
          FileAccess.Read);
      BinaryReader r = new BinaryReader(fs);
      // Read data from Test.data.
      for (int i = 0; i < 11; i++) {
         Console.WriteLine(r.ReadInt32());
         w.Close();
      }
   }
}
```

In the following example, the code defines a string and converts it to an array of characters, which can then be read as required by using the appropriate **StringReader.Read** method:

```
using System;
using System.IO;

public class CharsFromStr
{
   public static void Main(String[] args) {
     // Create a string to read characters from.
     String str = "Some number of characters";
     // Size the array to hold all the characters of the
     // string, so that they are all accessible.
     char[] b = new char[24];
     // Create a StringReader and attach it to the string.
     StringReader sr = new StringReader(str);
     // Read 13 characters from the array that holds
     // the string, starting from the first array member.
     sr.Read(b, 0, 13);
     // Display the output.
     Console.WriteLine(b);
     // Close the StringReader.
     sr.Close();
   }
}
```

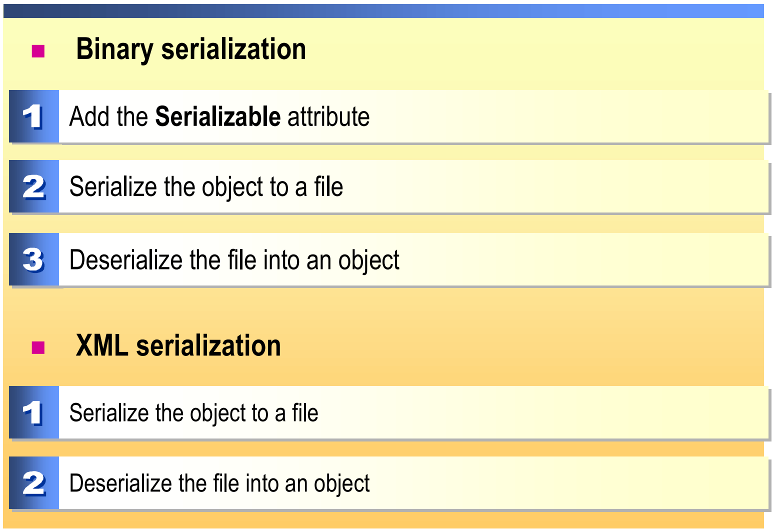
The preceding example produces the following output:

```
Some number o
```

### System.Text.Encoding

Internally, the common language run time represents all characters as Unicode. However, Unicode can be inefficient when transferring characters over a network or when persisting in a file. To improve efficiency, the .NET Framework class library provides several types that are derived from the **System.Text.Encoding** abstract base class. These classes know how to encode and decode Unicode characters to ASCII, UTF-7, UTF-8, Unicode, and other arbitrary code pages. When you construct a **BinaryWriter** or **StreamWriter**, you can choose any of these encodings. The default encoding is UTF-8.

# How to Serialize Objects



- **Binary serialization**

1 Add the **Serializable** attribute

2 Serialize the object to a file

3 Deserialize the file into an object

- **XML serialization**

1 Serialize the object to a file

2 Deserialize the file into an object

**Introduction**

*Serialization* is the process of converting the state of an object into a form that can be persisted or transported. The complement of serialization is *deserialization*, which converts a stream into an object. Together, these processes allow data to be easily stored and transferred. The .NET Framework features two serializing technologies: binary serialization and XML serialization.

**Binary serialization**

Binary serialization preserves type fidelity, which is useful for preserving the state of an object between different invocations of an application. For example, you can share an object between different applications by serializing it to the clipboard. You can serialize an object to a stream, disk, or memory; over the network; and so forth.

### Procedure: Saving data by using binary serialization

There are three main steps to saving data by using binary serialization.

1. Add the **Serializable** attribute to the class to be serialized.

```
 [Serializable]
public class MyObject {
  public int n1 = 0;
  public int n2 = 0;
  public String str = null;
}
```

2. Serialize the object to a stream by using a **BinaryFormatter** object.

```
MyObject obj = new MyObject();
obj.n1 = 1;
obj.n2 = 24;
obj.str = "Some String";
IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("MyFile.bin",
   FileMode.Create, FileAccess.Write, FileShare.None);
formatter.Serialize(stream, obj);
stream.Close();
```

3. When your application needs the persisted data, deserialize the file to an object, by using a **BinaryFormatter** object.

```
IFormatter formatter = new BinaryFormatter();
Stream stream = new FileStream("MyFile.bin", FileMode.Open,
   FileAccess.Read, FileShare.Read);
MyObject obj = (MyObject) formatter.Deserialize(stream);
stream.Close();
```

For an object to be successfully serialized, every object it refers to must also be serializable. If the serialization engine encounters any object that is not serializable during serialization, it will throw a SerializationException.

By default, binary serialization stores all the members of an object, both public and private. The .NET Framework provides attributes to control which members are serialized; you can use these if you want to override the default behavior. In addition, you can implement the ISerializable interface if you need complete control of the binary serialization process.

Because binary serialization stores all the members of an object by default, versioning can be tricky. If you add or remove members from an object, serialized representations of that object may not deserialize correctly. Careful use of attributes to prevent the serialization of non-essential members may avoid this problem, but you may have to implement the ISerializable interface to allow backward compatibility if you make many changes to your objects.

For more information about binary serialization, see the Binary Serialization section in the .NET Framework software development kit (SDK) documentation.

### XML serialization

XML serialization serializes only public properties and fields and does not preserve type fidelity. This is useful when you want to provide or consume data without restricting the application that uses the data. Because XML is an open standard, it is an attractive choice for sharing data across the Web.

### Procedure: Saving data by using XML serialization

There are two main steps to saving data by using XML serialization.

1. Serialize the object to a file.

```
public class OrderForm {
   public DateTime OrderDate;
}

OrderForm myObject = new OrderForm();
// Insert code to set properties and fields of the object.
XmlSerializer mySerializer = new
   XmlSerializer(typeof(OrderForm));
// To write to a file, create a StreamWriter object.
StreamWriter myWriter = new StreamWriter("myFileName.xml");
mySerializer.Serialize(myWriter, myObject);
```

2. When your application needs the persisted data, deserialize the object from a file.

```
OrderForm myObject;
// Construct an instance of the XmlSerializer with the type
// of object that is being deserialized.
XmlSerializer mySerializer = new
   XmlSerializer(typeof(OrderForm));
// To read the file, create a FileStream object.
FileStream myFileStream = new FileStream("myFileName.xml",
   FileMode.Open);
// Call the Deserialize method and cast to the object type.
myObject = (OrderForm)
   mySerializer.Deserialize(myFileStream);
```

The following items can be serialized using the **XmLSerializer** class.

- Public read/write properties and fields of public classes
- Classes that implement **ICollection** or **IEnumerable**

**Note** Only collections are serialized, not public properties.

- **XmlElement** objects
- **XmlNode** objects
- **DataSet** objects

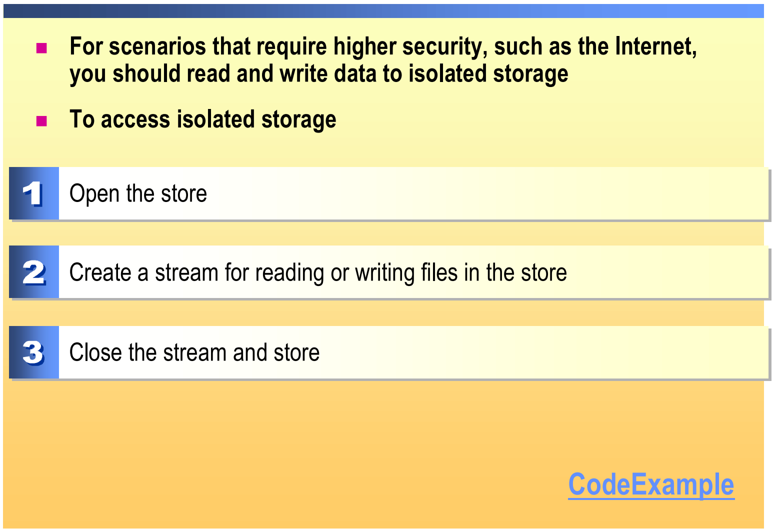
As with binary serialization, the .NET Framework provides attributes and other techniques to control XML serialization more finely. In addition, XML serialization is used by the .NET Framework to create SOAP messages for XML Web service calls. For more information about XML serialization, see the XML Serialization section of the .NET Framework SDK documentation.

### Comparison of binary and XML serialization

Generally speaking, binary serialization is best for storing private state and data used by one application or assembly. It produces compact files that are fast to save and load. XML serialization produces larger files and has more overhead but is better for cases when data will be shared across systems, such as with XML Web services.

| Binary serialization | XML serialization |
|---|---|
| Stores all members, both public and private, by default. | Only stores public members by default. |
| Usually produces very compact serialized representations of objects, with little overhead. | Usually produces larger serialized representations of objects, with more overhead. |
| Makes interoperation difficult, because object state is stored in a proprietary format. | Interoperation is easier, because it uses open standards such as XML and SOAP. |

# How to Use Isolated Storage



- For scenarios that require higher security, such as the Internet, you should read and write data to isolated storage
- To access isolated storage

1 Open the store

2 Create a stream for reading or writing files in the store

3 Close the stream and store

CodeExample

## Introduction

For some applications, such as downloaded Web applications and code that may come from sources that are not trusted, the basic file system does not provide the necessary isolation and safety. *Isolated storage* is a data storage mechanism that provides isolation and safety by defining standardized ways of associating code with saved data.

Administrators can use tools that are designed to manipulate isolated storage to configure file storage space, set security policies, and delete unused data. With isolated storage, code no longer must invent unique paths to specify safe locations in the file system while data is protected from unauthorized access. There is no need for hard coding of information that indicates where an application's storage area is located. With isolated storage, partially trusted applications can store data in a manner that is controlled by the computer's security policy. Security policies rarely grant permission to access the file system by using standard I/O mechanisms. However, by default, code that runs from a local computer, a local network, or the Internet is granted the right to use isolated storage. Web applications can also use isolated storage with roaming user profiles, thereby allowing user's isolated stores to roam with their profile.

The namespace **System.IO.IsolatedStorage** contains the **IsolatedStorageFile** and **IsolatedStorageFileStream** classes, which applications can use to access the files and directories in their isolated storage area.

### Procedure: Accessing isolated storage

There are three main steps to accessing isolated storage:

1. Open the store.

   ```
   IsolatedStorageFile isoStore = null;
   isoStore = IsolatedStorageFile.GetUserStoreForDomain();
   ```

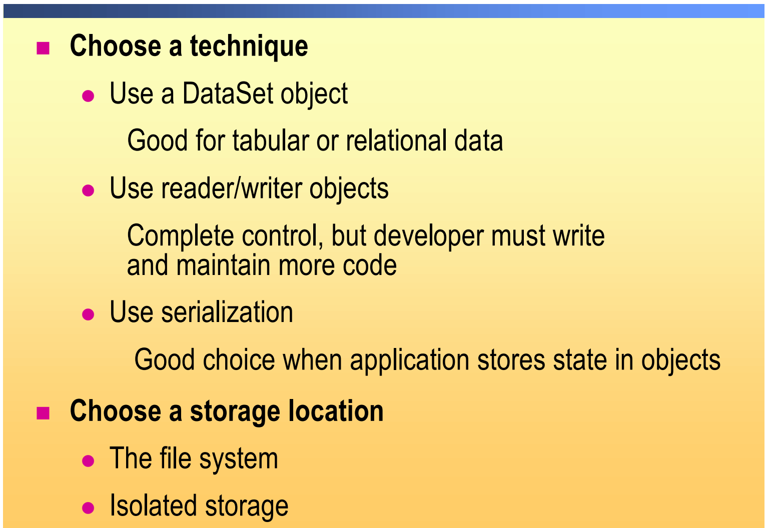
2. Create a stream for reading or writing files in the store.

   ```
   IsolatedStorageFileStream dataFile = null;
   dataFile = new IsolatedStorageFileStream("myfile.txt",
      FileMode.Open, isoStore);
   // code to read from or write to file goes here
   ```

3. Close the stream and store.

   ```
   dataFile.Close();
   isoStore.Close();
   ```

# How to Persist Application Settings



- Choose a technique
  - Use a DataSet object

    Good for tabular or relational data
  - Use reader/writer objects

    Complete control, but developer must write and maintain more code
  - Use serialization

    Good choice when application stores state in objects
- Choose a storage location
  - The file system
  - Isolated storage

### Introduction

There are many techniques that you can use to save user data in Windows Forms applications. Your application may use one or all of the following techniques, depending on its particular requirements.

### Storing user data in a DataSet object

If you have just a few properties to save, the **DataSet** object is a good candidate for saving user data. Data sets are easy to configure, access, and persist. Data sets are also obvious choices for user data in the form of rows and tables. For example, the expense report sample application stores saved reports and **DataSet** objects in isolated storage. Also, the order entry sample application uses a **DataSet** object to store some user setting, such as whether to play sounds when orders are entered.

### Saving user data to a file manually

If you want complete control over saving user data, or if you need to save data to a specific file format, manually saving data in a file is the right choice. With this technique, you can use **StreamReader** and **StreamWriter** objects or **BinaryReader** and **BinaryWriter** objects to store data. The main disadvantage of this approach is that you must write and maintain all the code to save your data.

### Saving user data by using serialization

Serialization is a good choice when your state is stored in a few objects, and you can use the serialization mechanisms in the .NET Framework to persist these objects. Depending on your particular needs, you can use binary serialization or XML serialization to persist your objects. Serialization provides very useful default behavior, but it also allows you a lot of flexibility if you need to customize its behavior.

### Location of saved user data

In addition to the storage technique, you must also choose the location to which you will save user data. In addition to the file system, the .NET Framework provides isolated storage. Using isolated storage is a good choice when you want a simple virtual file system that is tied to the identity of your code and the user. You must use the file system when you need to write files that are used by other applications or read files generated by other applications.

# Lab 4.2: Calling an XML Web Service

Exercise 1: Calling an XML Web Service

**Objectives**

After completing this lab, you will have demonstrated your ability to:

- Add a Web reference.
- Create an XML Web service proxy object.
- Call methods in an XML Web service.

**Note** This lab focuses on the concepts in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).* As a result, this lab may not comply with Microsoft security recommendations.

**Prerequisites**

Before working on this lab, you must have:

- The knowledge and skills to develop a simple Windows Forms application by using a Visual Studio .NET–compatible programming language.
- The knowledge and skills to debug an application by using Visual Studio .NET.
- The knowledge and skills to call XML Web services.

**Scenario**

Your organization, Contoso, Ltd., provides access to expense report information by using a private XML Web service. You are writing a Windows Forms client application that uses and updates expense report information. Your application will use your organization’s XML Web service to accomplish this task.

**Lab setup**

There are starter and solution files associated with this lab. Browse to *install_folder*\Labfiles\Lab04_2\Ex01\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab04_2\Ex01\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

**Estimated time to complete this lab: 15 minutes**

# Exercise 1
# Calling an XML Web Service

In this exercise, you will call some methods of an XML Web Service.

| Tasks | Additional information |
|---|---|
| **1.** Open the ExpenseReport project in Visual Studio .NET. Browse to *install_folder*\Labfiles\Lab04_2\Ex01\Starter to find the project files. | **a.** For more information about opening a project file and starting an application, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles** only check box, then search by using the phrase **Open Project Dialog Box**. For additional information about starting an application from in the Designer, in Index, search by using the phrase **Debugging Windows Applications**. |
| **2.** Add a Web reference to the test XML Web service on your local computer. The URL for this XML Web service is:<br>`http://localhost/`<br>`ExpenseReportWebService/`<br>`ExpenseReportWebService.asmx` | **a.** For more information about how to add a Web reference, see the following resources:<br>• Practice: Calling an XML Web Service in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Lesson: Creating a Simple XML Web Services Client in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. |
| **3.** Open the ExpenseReportUtilities file. Find the TODO comment toward the bottom of the file. Add the code to create the XML Web service proxy object and store it in the **WSInternal** member of the class. | **a.** For more information about how to create an XML Web service proxy object, see the following resources:<br>• Practice: Calling an XML Web Service in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Lesson: Creating a Simple XML Web Services Client in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. |

<table>
<tr><th>Tasks</th><th>Additional information</th></tr>
<tr><td>
4. Open the ExpenseReportList form, and view its code. Find the two TODO comments in the constructor. Add the code to make the XML Web service calls after the comments. In each case, you will retrieve the XML Web service proxy from the static (shared) **Utilities** object, by using the following syntax:

```
Utilities.WS
```

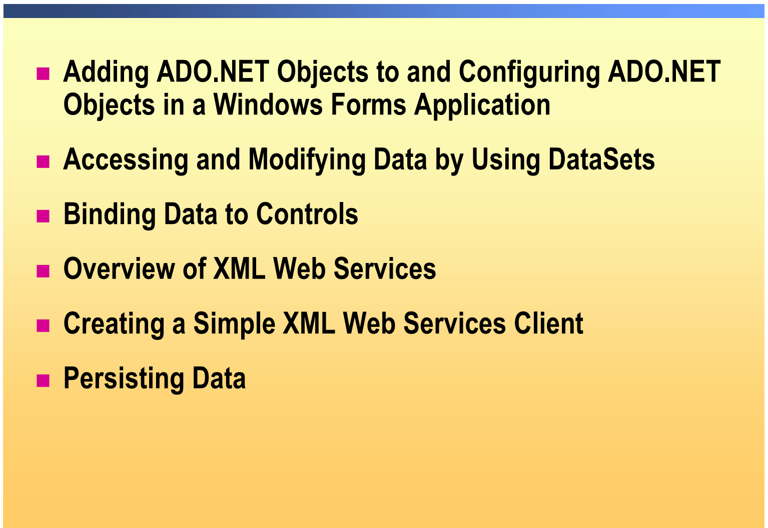
a. The first method call will be to the **GetReportsForEmployee** method. Store the return value from this call in the **ExpRepDataSet** data member, and pass the following parameters.

<table>
<tr><th>Parameter</th><th>Value</th></tr>
<tr><td>username</td><td>UCred.UserNameString</td></tr>
<tr><td>passwordToken</td><td>UCred.PasswordTokenString</td></tr>
<tr><td>reportIndex</td><td>RecordCursor</td></tr>
<tr><td>numberReports</td><td>10</td></tr>
<tr><td>totalNumberReports</td><td>ref TotalNumRecords</td></tr>
</table>

b. The second method call will be to the **GetReportsForManagerApproval** method. Again, store the return value from this call in the **ExpRepDataSet** data member, and pass the same parameters as in step 4a.
</td><td>
a. For more information about how to call a method of an XML Web service, see the following resources:

- Practice: Calling an XML Web Service in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- Lesson: Creating a Simple XML Web Services Client in Module 4, “Using Data in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
</td></tr>
<tr><td>
5. Build and run the application. Click the **View Submitted Reports** button and make sure the list of expense reports is populated.
</td><td>
Additional information is not necessary for this task.
</td></tr>
</table>

# Review



- Adding ADO.NET Objects to and Configuring ADO.NET Objects in a Windows Forms Application
- Accessing and Modifying Data by Using DataSets
- Binding Data to Controls
- Overview of XML Web Services
- Creating a Simple XML Web Services Client
- Persisting Data

1. What is the difference between a connection object and a command object?

   **Applications use Connection objects to connect to databases.**

   **You can use Command objects to access data directly in the database in a connected environment. Command objects use SQL statements or stored procedures to retrieve data. Commands travel across connections, and result sets are returned in the form of streams that can be read by DataReaders or pushed into DataSet objects.**

2. What is the difference between a dataset and a DataAdapter?

   **Datasets store data in a disconnected cache. The structure of a dataset is similar to that of a relational database; it exposes a hierarchical object model of tables, rows, and columns.**

   **A DataAdapter object serves as a bridge between a dataset and a data source for retrieving and saving data. The DataAdapter class represents a set of database commands and a database connection that you use to fill a dataset and update the data source.**

3. How do you access tables and columns in a typed and untyped dataset?

   **Typed dataset**

```
PubsDataSet.Titles;
```

   **Untyped dataset**

```
PubsDataSet.Tables(“Titles”);
```

4. How do you generate a new Database schema?

   **On the Project menu, click Add New Item.**

   **To add a schema, open the appropriate folder, and then double-click XML Schema.**

   **An XML Schema (.xsd) file is added to your project.**

5. What are the functions of **ReadXML** and **WriteXML** methods?

   **You can use the ReadXml method of the DataSet object to load data from an XML file into a dataset. When you use this method, you can load data from XML files that contain only XML data or from files that contain XML data as well as an inline schema.**

   **You can write data and schema information from a dataset to a file or stream by using the WriteXml method of the Dataset object.**

6. What is **CurrencyManager**?

   **The CurrencyManager object is used to keep data-bound controls synchronized with a data source. The CurrencyManger is important because data sources, such as tables in datasets, do not track the currently selected row. You need an intermediary object that is aware of the currently selected position in a data source and can notify databound controls when the position changes. That intermediary object is a CurrencyManager object.**

7. What are the main steps that you take in your application to call an XML Web service?

   **The main steps that you take in your application to call an XML Web service are to:**

   - **Generate the proxy that will call the XML Web service methods.**
   - **Instantiate the proxy.**
   - **Call the XML Web service methods that are required by your application.**

8. Describe the two ways to generate an XML Web service proxy.

   **Add a Web Reference in Visual Studio .NET and use the WSDL.exe utility.**

9. List three techniques for persisting application settings.

   **Three techniques for persisting application settings include using:**

   - **A DataSet object**
   - **Reader/writer objects**
   - **Serialization**

msdn® training

# Module 5: Interoperating with Managed Objects

**Contents**

Microsoft®

# Instructor Notes

**Presentation:**
**60 minutes**

**Lab:**
**30 minutes**

This module provides students with knowledge of how to use COM components in Microsoft® .NET Framework-based applications. The module also covers how to call Microsoft Win32® APIs from a .NET-based application.

After completing this module, students will be able to:

- Use .NET and COM components in a Microsoft .NET Framework Windows Forms application.
- Call Microsoft Win32 APIs from a Windows Forms application.

**Required materials**

To teach this module, you need the Microsoft PowerPoint® file 2555A_05.ppt.

**Preparation tasks**

To prepare for this module:

- Read all of the materials for this module.
- Complete the practices, demonstrations, and lab.

# How to Teach This Module

This section contains information that will help you to teach this module.

- Lab 5.1: Interoperating with COM and Calling Win32 APIs is based on the Purchase Order application in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C#™ .NET)* and is intended to simulate a real-world environment in which students will demonstrate what they learned during the lecture and practice portions of the module. The lab does not provide step-by-step detailed instructions; instead, the students are given tasks to complete in the left column and a list of resources that they can use (if they need help) in the right column. Students get the hands-on experience that they need by completing the practice activities at the end of each lesson.

## Lesson: Using .NET and COM Components in a Windows Forms Application

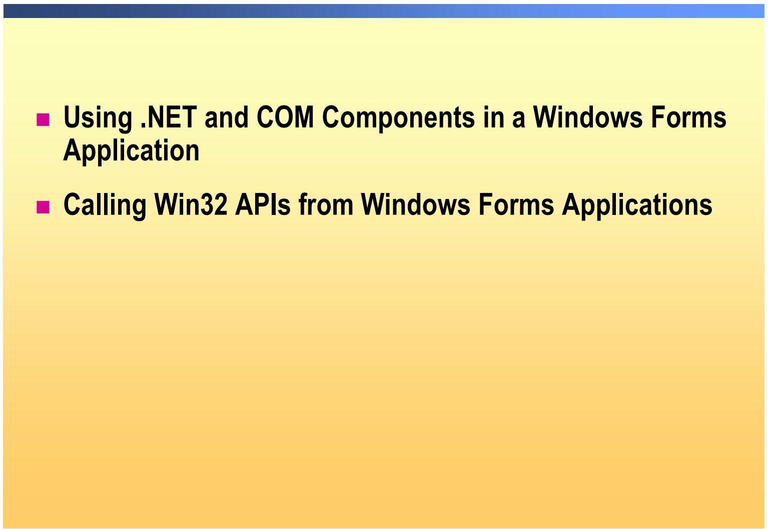
This section describes the instructional methods for teaching this lesson.

- Tell students that this lesson covers only how to call COM components from a .NET-based application. It does not cover using .NET objects from a COM client. For more information about calling a .NET object from a COM client, refer them to the Course 2571A*, Application Upgrade and Interoperability Using Visual Studio .NET.*

## Lab: Interoperating with COM and Calling Win32 APIs

- Make sure that you have demonstrated the two lab applications—the Expense Report application and the Purchase Order application—in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C#  .NET)* before students begin the lab. For information about how to demonstrate the lab scenarios, see the Introduction module in Course 2555A, *Developing .NET Windows Applications (Visual C# .NET).*
- Practice exercises will enable students to successfully complete the lab exercises. Therefore, make sure that students have completed all practice exercises before they begin the lab.

# Overview



- Using .NET and COM Components in a Windows Forms Application
- Calling Win32 APIs from Windows Forms Applications

### Introduction

With the advent of the Microsoft® .NET Framework, there is need to upgrade existing software systems and for developers to adapt to the new platform. To provide a convenient and easy migration path for existing systems, the .NET Framework offers a broad interoperable functionality between managed code and existing COM components.

Two of the primary goals of .NET Interoperation are to ensure that:

1. Installation of the .NET Framework does not affect existing COM applications.

   Because .NET and COM can *coexist*, you can selectively upgrade COM components to .NET without affecting other COM or .NET-based applications. Therefore, you may or may not choose to upgrade COM components to .NET.

2. The .NET Framework and COM components can communicate with each other; they can cooperate.

   Because .NET and COM can *cooperate*, you can build .NET-based applications that implement functionality exposed by COM objects and COM applications that implement functionality exposed by .NET objects.

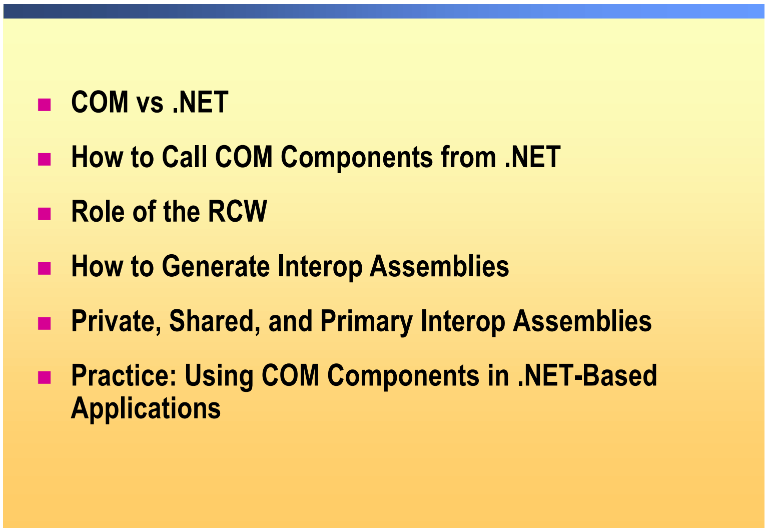
This module introduces you to the concepts and mechanics of the interoperation of .NET managed code with COM unmanaged code and platform services.

### Objectives

After completing this module, you will be able to:

- Use .NET and COM components in a Microsoft .NET Framework Windows Forms application.
- Call Microsoft Win32® APIs from a Windows Forms application.

# Lesson: Using .NET and COM Components in a Windows Forms Application



- COM vs .NET
- How to Call COM Components from .NET
- Role of the RCW
- How to Generate Interop Assemblies
- Private, Shared, and Primary Interop Assemblies
- Practice: Using COM Components in .NET-Based Applications

**Introduction**

The Microsoft .NET Framework provides interoperability services to use existing applications, smooth the migration path, and simplify the interaction between managed and unmanaged code. Interop Services conceal from both clients and servers the differences between the managed world and the unmanaged world.

The common language runtime provides two mechanisms for interoperating with unmanaged code:

- Platform Invocation Services

  Platform invoke is a service that allows managed code to call unmanaged functions that are implemented in dynamic-link libraries (DLL). It locates and invokes an exported function and marshals its arguments (integers, strings, arrays, structures, and so on) across the interoperation boundary as needed.

- COM Interop

  COM Interop is a bi-directional service that provides a bridge between the.NET Framework and COM. It allows .NET clients to call COM components and COM clients to call .NET Framework components.

  COM Interop maintains programming model consistency in both COM and .NET. It resolves the inconsistencies between the two models and makes the difference transparent to both clients and servers.

In this lesson, you will learn how to call COM components from the .NET Framework.

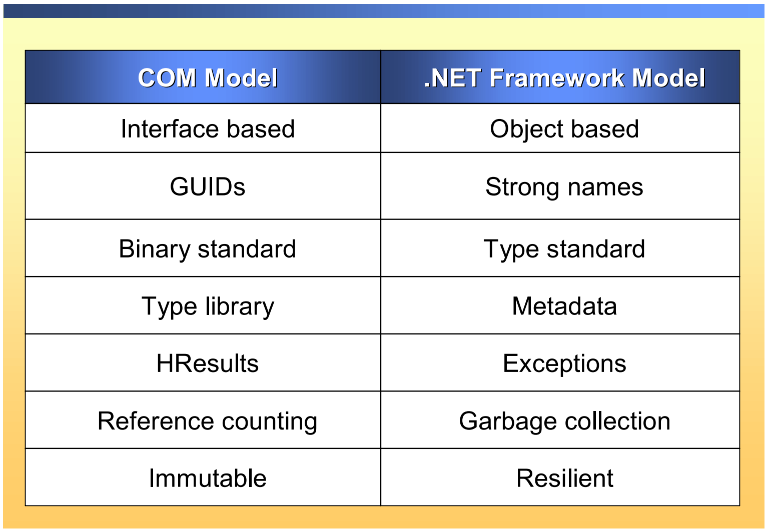
**Lesson objectives**

After completing this lesson, you will be able to:

- Identify the differences between COM and .NET programming models.
- Explain how to call COM components from the .NET Framework.
- Describe the role of the runtime callable wrapper (RCW) in interoperating.
- Generate interop assemblies.
- Describe private, shared, and primary interop assemblies.

**Note** This module does not cover using .NET objects from a COM client by using the COM callable wrapper (CCW). For more information about the CCW, see “COM Callable Wrapper” in the .NET Framework software development kit (SDK) documentation.

# COM vs .NET



| COM Model | .NET Framework Model |
|---|---|
| Interface based | Object based |
| GUIDs | Strong names |
| Binary standard | Type standard |
| Type library | Metadata |
| HResults | Exceptions |
| Reference counting | Garbage collection |
| Immutable | Resilient |

### Introduction

COM differs from the .NET Framework object model in several ways.

### Differences between COM and .NET

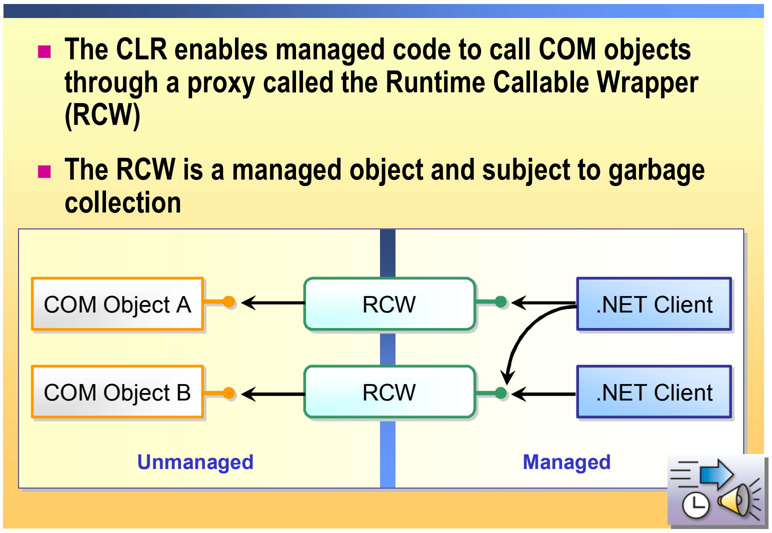
The differences between COM and .NET are summarized in the following table.

| Characteristic | COM | .NET Framework |
|---|---|---|
| Coding model | *Interface based*. COM objects always communicate through interfaces. | *Object based*. .NET Framework objects can pass data directly to each other, without implementing interfaces. |
| Identity | *Globally unique identifiers (GUIDs).* GUIDs identify a specific unmanaged type and provide no location information for that type. | *Strong names*. Strong names consist of a unique assembly name in addition to a type name. Because the assembly name uniquely identifies the type, you can reuse a type name across multiple assemblies. An assembly also introduces publisher key, version, and location information to a managed type. |
| Type compatibility | *Binary standard*. The internal binary layout of classes must comply with COM rules. | *Type standard.* The .NET common type system specification establishes a framework that enables cross-language integration, type safety, and so on. |
| Type definition | *Type library*. Type information is stored only for public types. Moreover, a type library is optional. | *Metadata*. Type information is stored as metadata and is mandatory for all types. Metadata is embedded inside assemblies. |

**(*continued*)**

| **Characteristic** | **COM** | **.NET Framework** |
|---|---|---|
| Type safety | *Not type-safe*. Unmanaged compilers do not provide type checking on pointer types. Therefore, the code becomes susceptible to potentially harmful activity. | *Optionally safe*. Managed code requires a higher level of trust. Therefore, although you can use pointers in managed code, the code has restrictions because of its unsafe behavior. |
| Error-handling mechanism | *HRESULTs*. A COM method returns an HRESULT to indicate whether the call succeeded or failed. | *Exceptions*. Managed code incorporates exceptions. |
| Object-lifetime management | *Reference counting*. Clients of COM objects manage object lifetime by means of reference counting. | *Garbage collection*. The common language runtime manages the lifetime of objects by means of garbage collection. |
| Versioning | *Immutable*. COM interfaces are immutable. In other words, if you change an interface, you must rename it with a new GUID. | *Resilient*. Managed types can evolve, retaining the same name. |

# How to Call COM Components from .NET



- The CLR enables managed code to call COM objects through a proxy called the Runtime Callable Wrapper (RCW)
- The RCW is a managed object and subject to garbage collection

**Introduction**

Many organizations cannot afford to upgrade to .NET overnight. A lot of money has been invested in the existing systems and users need to preserve their investment. In addition, some critical business applications cannot be changed. Therefore, Interop Services enable developers to use existing code or migrate their existing systems to .NET in multiple stages.

**Process**

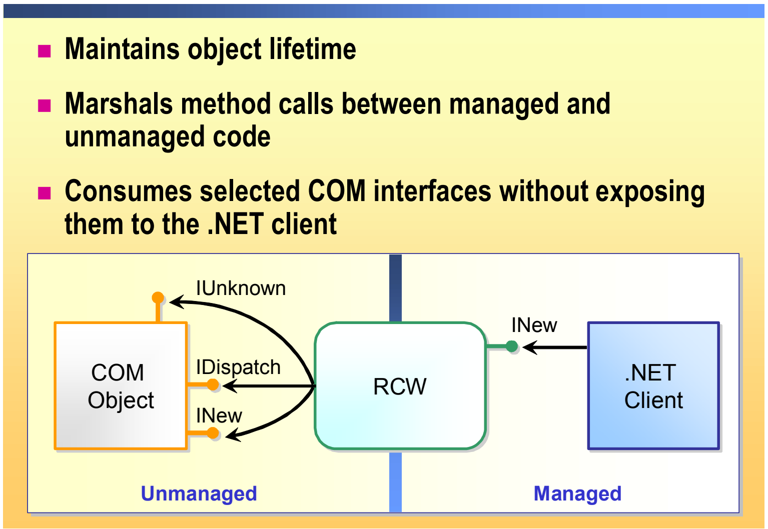
When a .NET client calls a COM component, a series of events take place.

1. When a .NET Client loads a COM object, a runtime callable wrapper (RCW) is created.

   The runtime creates exactly one RCW for each COM object, regardless of the number of references that exist on that object. This ensures that a single object identity is shared between the RCW and the COM object.

2. Using metadata derived from a type library, the runtime creates both the COM object being called and a wrapper for that object.
3. Each RCW maintains a cache of interface pointers on the COM object it wraps.
4. The runtime performs garbage collection on the RCW.

# Role of the RCW



- Maintains object lifetime
- Marshals method calls between managed and unmanaged code
- Consumes selected COM interfaces without exposing them to the .NET client

### Introduction

The primary goal of the RCW is to hide the differences between the managed and unmanaged programming models.

### Role of the RCW

The RCW is an important part of the interoperation model. It performs several roles.

1. Maintains object lifetime.

   The RCW is responsible for keeping the COM object alive as long as the RCW is being used. The runtime performs garbage collection on the RCW like any other managed object, but the RCW internally holds a reference to the COM object it wraps.

2. Marshals method calls between managed and unmanaged code.

   The RCW marshals method calls between managed and unmanaged code on behalf of the wrapped object. This involves several tasks, such as:

   a. Implicitly converting parameters passed between the two environments to the correct type.

      For example, when you pass a parameter of the type **System.String** to a COM function, the RCW automatically converts the parameter to the correct BSTR type expected by the COM component. If the COM function returns a BSTR parameter to the .NET client, the RCW automatically converts the parameter to **System.String**.

   b. Translating **HRESULT** errors generated by COM objects to .NET exceptions.

      For example, the RCW translates the **HRESULT** error **COR_E_UNAUTHORIZEDACCESS** to **UnauthorizedAccessException** exception in .NET.

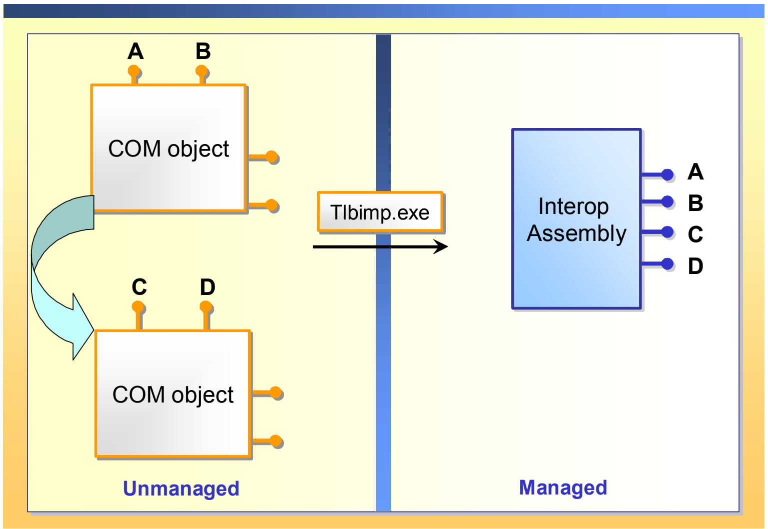
**Note** There are some limitations to using the RCW. For example, Success HRESULTS are not returned from unmanaged code. Variable length arrays also pose challenges. For more information about these limitations and how to fix them, refer to the "Interop Marshaling" section of the .NET Framework SDK.

3. Consumes selected COM interfaces without exposing them to the .NET client.

   An important role of the RCW is to make a .NET client act as if it is using a .NET object and to make a COM object act as if it is being used by a COM client. To accomplish this, the RCW consumes selected COM interfaces without exposing them to the managed client. For example, the COM **IUnknown** interface exposes methods for reference counting and querying interfaces on all COM objects. The RCW consumes this interface from the COM object but does not expose it to .NET clients. Instead, the RCW uses this interface internally to function as a COM object.

4. Allows developers to write code that treats COM objects wrapped by the RCW like any other object.

# How to Generate Interop Assemblies



**Introduction**

.NET-based applications can bind to COM classes in an early or late bound fashion. Early binding requires complete type information for the COM class at compile-time and requires the underlying COM class to support early binding as well. COM type definitions usually reside in a type library. To use COM types in managed code, you must generate metadata from type libraries.

COM type libraries can be standalone .tlb files or they can be embedded in the resource section of a .dll or .exe file. Other sources of type-library information are .olb and .ocx files.

After locating the type library containing the implementation of your target COM type, you can generate an assembly containing type metadata. The metadata can be generated in various ways. Usually it is generated from an existing type library that describes the types with a tool called Tlbimp.exe (Type Library Importer).

After the metadata is generated, it is saved as a typical .NET assembly. The assemblies are commonly called *interop assemblies* because they contain metadata for COM types but no code.

Just like any other assembly, you can examine an interop assembly with tools like Ildasm.exe to browse the types defined in the assembly.

There are a variety of tools that allow you to generate interop assemblies. The most common include the Microsoft Visual Studio® .NET IDE and the Tlbimp.exe utility.

**Procedure: Generate interop assemblies**

To add a reference to a type library from the Visual Studio IDE:

1. On the **Project** menu, click **References**.
2. Click the **COM** tab.
3. In the **Available References** list, double-click the type library.
4. Click **OK** to run TLBIMP and add the assembly to the hidden Bin directory of the project.

### Procedure: Add a reference to a type library by using the TLBIMP tool

You can build metadata by using the TLBIMP tool. The TLBIMP provides command-line switches to adjust metadata in the resulting interop file, imports types from an existing type library, and generates an interop assembly and a namespace. The TLBIMP provides a finer degree of control when creating interop assemblies.

Use the following syntax to generate an interop assembly by using the TLBIMP tool:

**Tlbimp TlbFile [/out**: *name*] **[/reference**: *file*] [/**silent**] [/**verbose**] [/**strictref**] [/**unsafe**][ [/**primary**]][/**publickey**: *file*] [/**keyfile**: *file*][/**keycontainer**: *name*]

TLBIMP uses the command line switches described in the following table.

| Command Line Switch | Description |
|---|---|
| **/out:***filename* | Specifies the name of the output file, assembly, and namespace in which to write the metadata definitions. |
| **/keyfile:***filename* | Options for supplying information to generate a strongly named assembly. |
| **/publickey:***filename* | |
| **/keycontainer:***containername* | |
| **/reference:***filename* | Specifies the assembly file to use to resolve references to types defined outside the current type library. If you do not specify the **/reference** option, Tlbimp.exe automatically recursively imports any external type library that the type library being imported references. |
| **/primary** | Produces a primary interop assembly for the specified type library. Information is added to the assembly indicating that the publisher of the type library produced the assembly. |

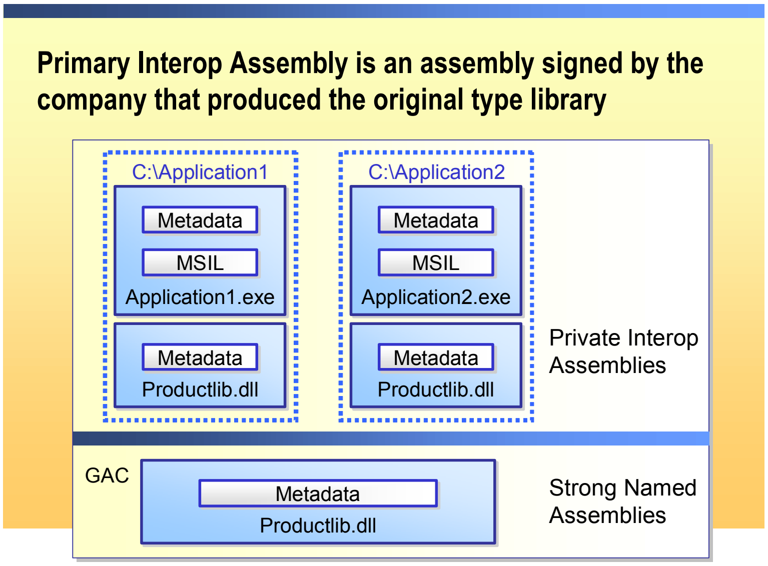
### Example

The following command line creates an interop assembly by using the Com1.tlb COM type library.

```
tlbimp COM1.tlb /keyfile:myPublicKeyFile.sn
/reference:COM2InteropAssembly.dll
/out:COM1InteropAssembly.dll
```

The example uses the myPublicKeyFile.sn file to assign a strong name to the assembly by using a public key. The COM1 component calls members of another COM class called COM2. However, if you have already created an interop assembly for COM2 called COM2InteropAssembly.dll, this will be a duplication of work. To prevent the TLBIMP tool from generating an interop assembly that contains information for both COM1 and COM2, add a reference to the existing COM2InteropAssembly.dll. This will ensure that metadata for COM2 will not be generated in the COM1InteropAssembly.

# Private, Shared, and Primary Interop Assemblies



Primary Interop Assembly is an assembly signed by the company that produced the original type library

**Introduction**

When metadata is generated from a type library, the resulting assembly can be private and exposed only in the scope of an application or it can be shared by being placed in the global assembly cache.

**Private interop assemblies**

To install an assembly for private use, both the application .exe and the assembly containing imported COM types must be installed in the same directory structure. The illustration shows an unsigned assembly to be used privately by Application1.exe and Application2.exe, which reside in separate application directories. The interop assembly, which is called ProductLib.dll in this example, is installed twice.

**Shared assemblies**

Assemblies shared by multiple applications should be installed in a centralized repository called the global assembly cache. .NET clients can access the same copy of the interop assembly, which is signed and installed in the global assembly cache. A shared assembly has a strong name made up of the assembly name, version, culture, and the public key of the assembly publisher. TLBIMP includes the **/keyfile**, **/publickey**, and **/keycontainer** options for supplying information to generate strongly named assembly.

**Primary interop assemblies**

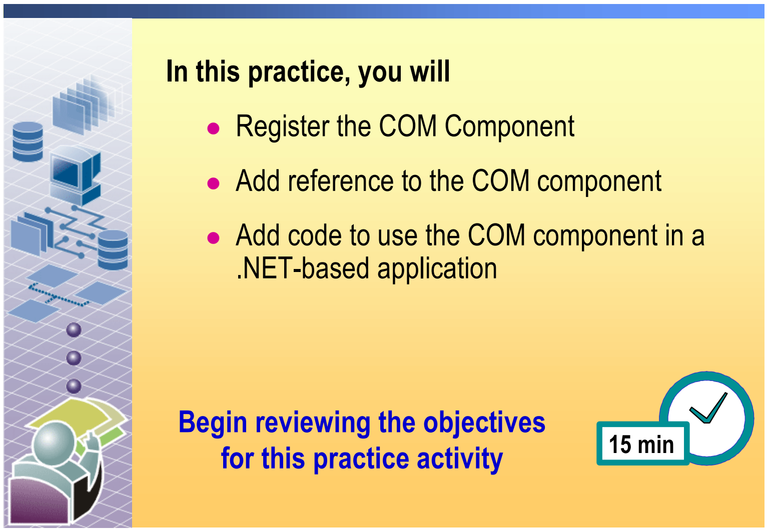
A shared interop assembly signed by the company that produced the original type library is called a *primary interop assembly*. An assembly signed by any other company or individual is called an *alternate interop assembly*.

Primary interop assemblies (PIA) are provided by the same publisher as the type library that they describe and provide the official definitions of the types defined with that type library. Primary interop assemblies are always signed by their publisher to ensure uniqueness. A primary interop assembly is created from a type library by running TLBIMP with the **/primary** switch.

PIAs are important because they provide unique type identity. They distinguish the official type definitions from all other definitions provided by other interop assemblies, thereby ensuring type compatibility between applications that share the types defined in a PIA.

When available, always use the primary interop assembly published by the author of the COM component you intend to incorporate in your managed code. Types in the primary interop assembly are imported for you and are ready to activate and call from managed code. Avoid using interop assemblies that are not primary interop assemblies.

# Practice: Using COM Components in .NET-Based Applications



**In this practice, you will**

- Register the COM Component
- Add reference to the COM component
- Add code to use the COM component in a .NET-based application

Begin reviewing the objectives for this practice activity

15 min

*****************************ILLEGAL FOR NON-TRAINER USE******************************

**Introduction**

In this practice, you will use two COM components in your .NET-based application, **Masked Edit ActiveX** and **NorthwindData_COM**.

The **Masked Edit ActiveX** control generally behaves as a standard text box control with enhancements for optional masked input and formatted output. The **Mask** property of the **Masked Edit** control allows developers to define the formatting of text in the control and the **ClipText** property provides access to the unformatted text.

The **NorthwindData_COM** component uses Microsoft ADO 2.6 to connect to the Northwind database. The component contains a class called **RemainingInventoryClass** that exposes the **ShowUnitsInStock** method. The **ShowUnitsInStock** method returns a live count of the number of units remaining in inventory for a product.

**Instructions**

### ► Register the NorthwindData_COM component

1. To open the Command Prompt window, click **Start**, point to **All Programs**, point to **Accessories**, and then click **Command Prompt**.
2. In the Command Prompt window, type the following and then press ENTER:

```
Regsvr32 “C:\Program
Files\Msdntrain\2555\Practices\Mod05\Mod05_01\Starter\North
windData_COM.dll"
```

**Note** The path in step 2 will change if the starter files for the practices and demonstrations are installed in a different location on your computer.

**Tip** You can also drag the NorthwindData_COM.dll file from Microsoft Windows Explorer to the **Run** dialog box or to the Command window, append **regsrv32** to the beginning of the file name, and then press ENTER.

A message box confirms a successful registration.

### ► Add a reference to the NorthwindData_COM component

1. Open Visual Studio .NET.
2. On the **File** menu, point to **New**, and then click **Project**.
3. In the **New Project** dialog box, click **Windows Application**, in the **Name** box, type **ComInterop** and then click **OK**.
4. On the **Project** menu, click **Add Reference**.
5. Click the **COM** tab, double-click the **NorthwindData_COM** component, and then click **OK**.

### ► Add the Masked Edit control to the Toolbox

1. On the **Tools** menu, click **Customize Toolbox**.
2. Select **Microsoft Masked Edit Control, version 6.0**, and then click **OK**.

   Confirm that the **MaskEdBox** control is added to the Toolbox in the Visual Studio IDE.

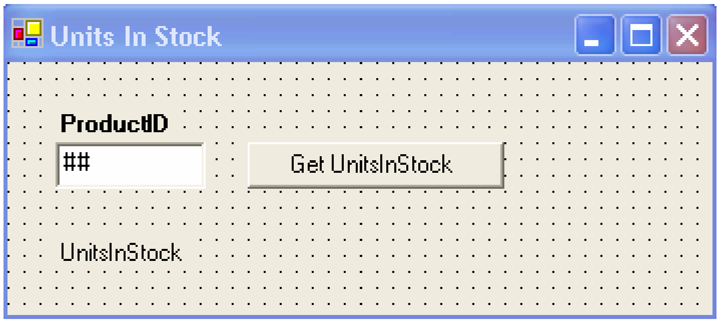
### ▶ Configure the controls on the form

1. Add the following controls to the form.

| Control | Name | Text |
|---|---|---|
| Label | ProductIDLabel | ProductID |
| MaskEdBox | COMMaskEdBox | |
| Button | GetUnitsInStockButton | Get UnitsInStock |
| Label | UnitsInStockLabel | UnitsInStock |

2. Set the **Mask** property of **COMMaskEdBox** to ##.

   Your form should look like the following illustration.



### ▶ Add code to use the COM objects in your .NET-based application

1. Create a procedure for the **Click** event of **GetUnitsInStockButton**.
2. Create a String variable named **serverName** and set it equal to **(local\MOC)**.
3. Create a Short variable called *productID* and assign it the value of the **ClipText** property of the **COMMaskedEdit** control. Convert the **ClipText** value to Short (Int16) before assigning it to *productID*.
4. Create a variable named *comObject* to hold an instance of the **NorhwindData_COM.RemainingInventoryClass** component.
5. Call the **ShowUnitsInStock** method of **comObject** and pass *productID*, by reference, as the first argument. Pass **serverName**, by reference, as the second argument. Assign the string returned from the **ShowUnitsInStock** method to the **Text** property of **UnitsInStockLabel**. Your code should look like this:

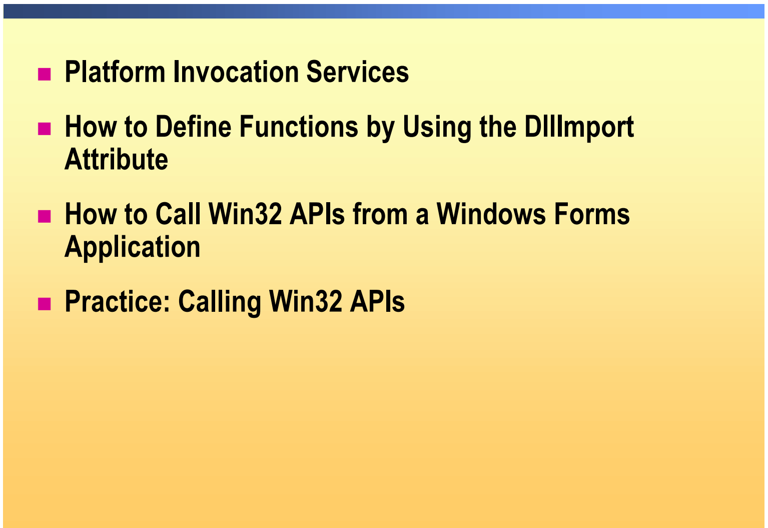
```
private void GetUnitsInStockButton_Click(object sender,
System.EventArgs e)
{
   serverName = "(local)\\MOC";
   short prodID = Convert.ToInt16(COMMaskEdBox.ClipText);
   NorthwindData_COM.RemainingInventoryClass comObject =
new NorthwindData_COM.RemainingInventoryClass();
   UnitsInStockLabel.Text = comObject.ShowUnitsInStock(ref
prodID, ref serverName);
}
```

► **Test the application**

1. Press F5 to compile and run the application.
2. Enter a number between **1** and **77**, and then click **GetUnitsInStock**.

   Confirm that the label reflects the number of items in stock.

# Lesson: Calling Win32 APIs from Windows Forms Applications



- Platform Invocation Services
- How to Define Functions by Using the DllImport Attribute
- How to Call Win32 APIs from a Windows Forms Application
- Practice: Calling Win32 APIs

**Introduction**

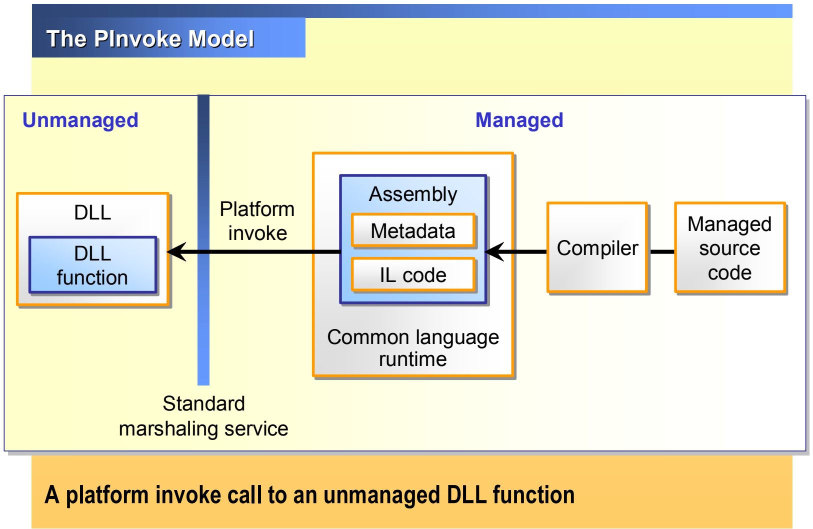
Windows APIs are DLLs that are part of the Microsoft Windows® operating system. You use them to perform tasks when it is difficult to write equivalent procedures of your own. The advantage of using Windows APIs in your code is that they can save development time because they contain hundreds of useful functions that are already written and waiting to be used. The disadvantage is that Windows APIs can be difficult to work with and unforgiving when things go wrong.

Windows DLLs represent a special category of interoperability. Windows APIs do not use managed code, do not have built-in type libraries, and use data types that are different from those used with Visual Studio .NET. Because of these differences, and because Windows APIs are not COM objects, interoperability with Windows APIs and the .NET Platform is performed by using platform invoke. Platform invoke is a service that enables managed code to call unmanaged functions implemented in DLLs. You can use platform invoke in C# by applying the **DllImport** attribute to an empty procedure. This lesson covers how to call Win32 APIs from Windows Forms applications by using platform invoke.

After completing this lesson, you will be able to:

- Explain the role of the Platform Invocation Services in calling unmanaged code from managed code.
- Call Win32 APIs from a Windows Forms application by using the **Declare** statement and the **DllImport** attribute.

# Platform Invocation Services



The PInvoke Model

A platform invoke call to an unmanaged DLL function

### Introduction

Platform Invocation Services is a mechanism by which managed code calls unmanaged code.

### Definition

Platform invoke allows managed code to call unmanaged functions that are implemented in a DLL. Platform invoke takes care of finding and invoking the correct function, as well as marshaling its managed arguments to and from their unmanaged counterparts (integers, strings, and so on). Platform invoke and COM Interop share much of the marshaling code inside the .NET Framework.

You can use the **DllImport** attribute to declare functions in DLLs.

### Role of platform invoke in calling Win32 APIs

When platform invoke calls an unmanaged function, it performs the following set of actions.

1. Locates the DLL containing the function.

   To locate a function in a DLL, platform invoke must have the following information.

   a. The name of the DLL file, such as User32.dll, GDI32.dll, Kernel32.dll.

   b. The name of the function or ordinal number, such as **MessageBox** or exported function number #**452**.

   c. The character set that will be used—ANSI or Unicode.

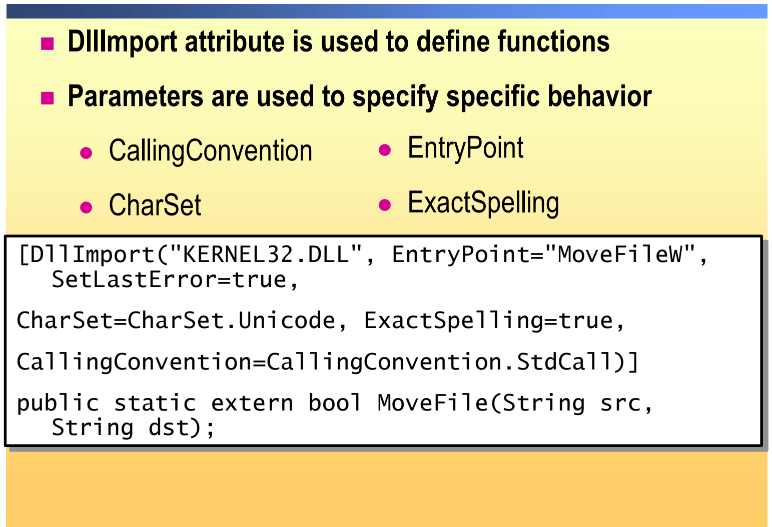
      Win32 API contains two versions of each function: a single-Byte character ANSI version and a double-Byte character Unicode version. For instance, MessageBoxA is the ANSI entry point for the **MessageBox** function and MessageBoxW is the Unicode version.

      The character set is controlled by the value of the **CharSet** field of **DllImportAttribute**, which will be discussed later in more detail.

2. Loads the DLL into memory.

3. Locates the address of the function in memory and pushes its arguments onto the stack, marshaling data as required.
4. Transfers control to the unmanaged function.
5. Returns exceptions generated by the unmanaged function to the managed caller.

# How to Define Functions by Using the DllImport Attribute



- DllImport attribute is used to define functions
- Parameters are used to specify specific behavior
  - CallingConvention
  - CharSet
  - EntryPoint
  - ExactSpelling

```
[DllImport("KERNEL32.DLL", EntryPoint="MoveFileW",
   SetLastError=true,
CharSet=CharSet.Unicode, ExactSpelling=true,
CallingConvention=CallingConvention.StdCall)]
public static extern bool MoveFile(String src,
   String dst);
```

### Introduction

The **DllImport** attribute is used to specify the DLL location that contains the implementation of an external method.

### DllImport parameters

**DllImport** has the following named parameters that specify the specific behavior of the attribute.

- *CallingConvention*

  The *CallingConvention* parameter indicates the calling convention for the entry point. If no *CallingConvention* is specified, a default of CallingConvention.Winapi is used.

- *CharSet*

  The *CharSet* parameter indicates the character set used in the entry point. If no *CharSet* is specified, a default of CharSet.Auto is used.

- *EntryPoint*

  The *EntryPoint* parameter gives the name of the entry point in the DLL. If no *EntryPoint* is specified, the name of the method is used.

- *ExactSpelling*

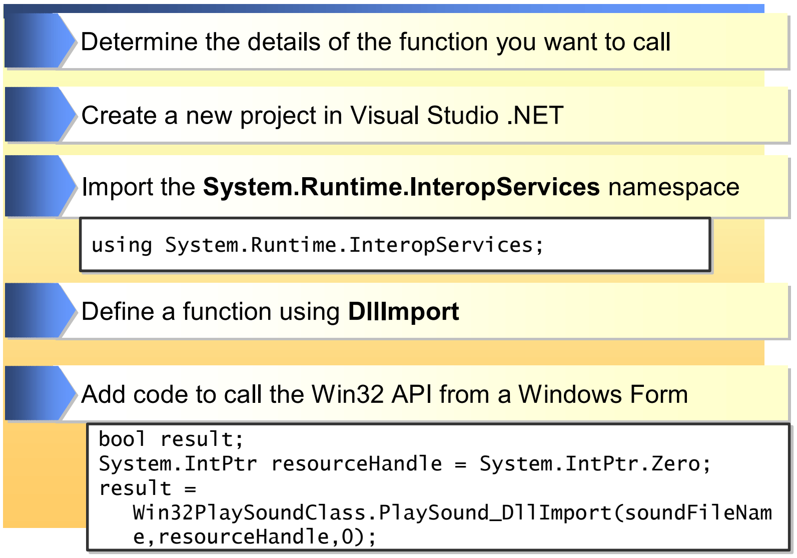
  The *ExactSpelling* parameter indicates whether *EntryPoint* must exactly match the spelling of the indicated entry point. If no *ExactSpelling* is specified, a default of **False** is used.

### Procedure

Apply the **DllImport** attribute to the empty function. The first parameter is the name and location of the DLL containing the function that you are calling. You do not need to specify the path for files located in the Windows system directories. The second parameter is a named argument that specifies the name of the function in the Windows API.

```
[DllImport("KERNEL32.DLL", EntryPoint="MoveFileW",
SetLastError=true,
CharSet=CharSet.Unicode, ExactSpelling=true,
CallingConvention=CallingConvention.StdCall)]
public static extern bool MoveFile(String src, String dst);
```

# How to Call Win32 APIs from a Windows Forms Application



Determine the details of the function you want to call

Create a new project in Visual Studio .NET

Import the **System.Runtime.InteropServices** namespace

```
using System.Runtime.InteropServices;
```

Define a function using **DllImport**

Add code to call the Win32 API from a Windows Form

```
bool result;
System.IntPtr resourceHandle = System.IntPtr.Zero;
result =
   Win32PlaySoundClass.PlaySound_DllImport(soundFileNam
   e,resourceHandle,0);
```

**Introduction**

So far, you have learned the components that are involved in calling a Win32 API from a .NET Framework-based application. Now you will look at the entire procedure of calling a Win32 API from a .NET Windows Forms application.

**Procedure**

To call a Win32 API:

1. Determine the name of the function that you want to call, its arguments, argument types, return value, and the name and location of the DLL that contains it.
2. Create a new class in Visual Studio .NET.
3. Import the **System.Runtime.InteropServices** namespace.

```
using System.Runtime.InteropServices;
```

4. Define a function by using **DllImport**.

   Add the following **DllImport** attribute either to the declaration section of the startup form for your project, or to the declaration section of the class or module where you want to use the DLL.

   ```
   [DllImport("winmm.dll", EntryPoint="PlaySound")]
   public static extern bool PlaySound_DllImport(string
   pszSound,IntPtr hmod ,int fdwSound );
   ```

5. Add code to call the Win32 API from a Windows Form.

   ```
   bool result;
   System.IntPtr resourceHandle = System.IntPtr.Zero;
   result =
   Win32PlaySoundClass.PlaySound_DllImport(soundFileName,
   resourceHandle,0);
   ```

**Note** Although calling an unmanaged DLL function from managed code is almost identical to calling a managed function, there are some differences when passing data structures and using callback functions. For more information about making advanced calls through platform invoke, see Course 2571A*, Application Upgrade and Interoperability Using Visual Studio .NET.*

# Practice: Calling Win32 APIs

In this practice, you will call a Win32 API from a .NET-based application using the DllImport attribute

Begin reviewing the objectives for this practice activity

### Introduction

In this practice, you will create and test a class that calls the Win32 PlaySound API.

The PlaySound API is used to play sound files such as .wav files and is located in the Winmm.dll file. It has the following signature:

```
BOOL PlaySound(
  LPCSTR pszSound,
  HMODULE hmod,
  DWORD fdwSound
);
```

Notice that the arguments of the API use unmanaged C++ variable types. You must understand these variables types to convert them to an equivalent .NET type.

For more information and a list of unmanaged data types and their managed equivalents, see the “Platform Invoke Data Types” section of the .NET Framework SDK.

The **PlaySound** function takes three arguments and returns a Boolean value indicating whether or not the function was able to play the sound file. The arguments include:

- **pszSound**

  A string that defines the name of the sound file.

- **hmod**

  Handle to the executable file that contains the resource to be loaded. This parameter must be **NULL** unless **SND_RESOURCE** is specified in **fdwSound**.

- **fdwSound**

  Flags that are used to determine how the sound is played. Examples of flags include whether the sound should play asynchronously or whether it should loop.

### Instructions

**► Open the practice project**

1. Using Windows Explorer, navigate to *install_folder*\Practices\Mod05\Mod05_02\Starter.

> **Note** If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

2. Double-click the PInvoke.sln solution file to open the project.

**► Add a class to the project and import the System.Runtime.InteropServices namespace**

1. On the **Project** menu, click **Add Class**.
2. Name the class **Win32PlaySoundClass.cs**, and then click **Open**.
3. Import the **System.Runtime.InteropServices** namespace by typing the following code above the **Win32PlaySoundClass**.

```
using System.Runtime.InteropServices;
```

**► Use the DllImportAttribute to call the PlaySound Win32 API**

1. Open the Win32PlaySoundClass file.
2. Inside the **Win32PlaySound** class, create a **DllImport** attribute that exposes the Winmm.dll and sets the **EntryPoint** parameter to **PlaySound**. Your code should look like the following.

```
[DllImport("winmm.dll", EntryPoint="PlaySound")]
```

3. Append a public static external function to the **DllImport** attribute named **PlaySound_DllImport** that returns a Boolean value. Create the arguments required by the **PlaySound** function by using the following table, which includes the argument names, the unmanaged type expected, and the .NET equivalent type.

| Argument Name | Unmanaged Type | Managed Equivalent |
|---|---|---|
| pszSound | LPCSTR | String |
| hmod | HMODULE | IntPtr |
| fdwSound | DWORD | Int |

Your code should look like the following.

```
[DllImport("winmm.dll", EntryPoint="PlaySound")]
public static extern bool PlaySound_DllImport(string
pszSound,IntPtr hmod ,int fdwSound );
```

► **Add code to test the PlaySound_DllImport function**

1. Open the PlaySoundForm in the Code Editor, and then locate the **PlayButton_Click** event procedure.
2. Declare a Boolean value named **result** to hold the results of the **PlaySound_Dll** function.
3. Create the *IntPtr* variable named *resourceHandle* and initialize it to **0**.
4. Call the **PlaySound_Dll** function of the **Win32PlaySound** class and assign the results to the **result** variable. Use the following table to fill the parameters.

| Parameter | Value |
|---|---|
| pszSound | soundFileName |
| hmod | resourceHandle |
| fdwSound | 0 |

Your code should look like the following.

```
bool result;
System.IntPtr resourceHandle = System.IntPtr.Zero;
result =
Win32PlaySoundClass.PlaySound_DllImport(soundFileName,
resourceHandle,0);
```

### ► Test the PlaySound_DllImport function

1. Press F5 to build and run the application.
2. On the **File** menu, click **Open**.
3. Click the **Windows XP Startup.wav** file, and then click **Open**.
4. Click the **Play** button to test the sound.
5. Close the application.

---

**Note** If there are no speakers available to test the sound, test the value of the **result** variable in the debug mode.

---

### ► Resolve the errors in the PlaySound_DllImport function of the Win32PlaySound_Error class

- Open the **Win32PlaySound_ErrorClass** file and review the **PlaySound_DllImport** function. There is an error in the function. You can attempt to test it to confirm. What is the error?

  **The function explicitly sets the ExactSpelling to True; however, the EntryPoint is set to PlaySound. The function PlaySound does not exist in Winmm.dll; only PlaySoundA (Ansi) and PlaySoundW (Unicode) exist. To correct this error, set ExactSpelling to False. Another way to correct this error is to explicitly set the EntryPoint to PlaySoundW.**

  ____________________________________________________________

  ____________________________________________________________

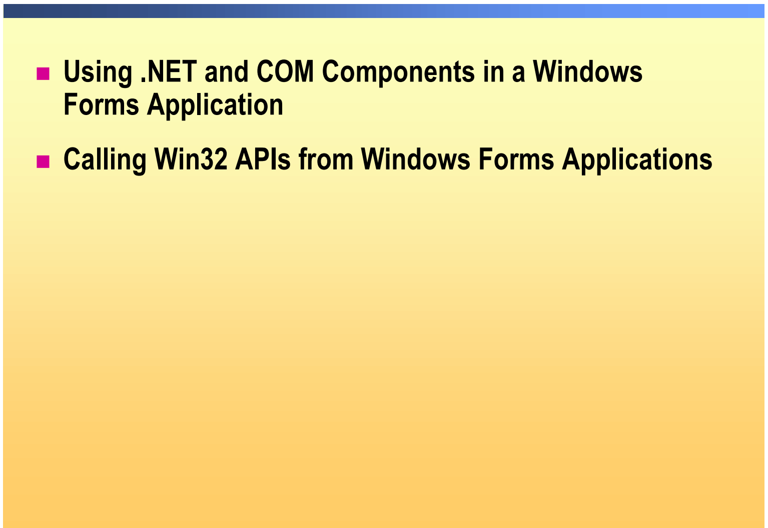
  ____________________________________________________________

---

**Note** You can also correct this error by setting the **EntryPoint** parameter to **PlaySoundA**; however, you must also set the **CharSet** parameter to **CharSet.Ansi**.

---

# Review



- Using .NET and COM Components in a Windows Forms Application
- Calling Win32 APIs from Windows Forms Applications

1. List some of the differences between COM and .NET components.

   **COM is based on binary standard. The internal binary layout of classes must comply with COM rules. .NET is based on a type standard. The common type system in .NET establishes a framework that enables cross-language integration, type safety, and high performance code execution.**

   **COM uses type libraries to store type information. In the .NET Framework, type information is stored as metadata and is mandatory for all types. Metadata is embedded inside assemblies.**

   **COM methods usually return an HRESULT, indicating that the call succeeded or failed. In .NET, managed code incorporates exceptions.**

   **Clients of COM objects manage object lifetime by reference counting. In .NET, the runtime manages the lifetime of objects through garbage collection.**

   **For identity, COM uses GUIDs. .NET uses strong names.**

2. How do you call COM components from a .NET-based application?

   **When a .NET Client loads a COM object, a RCW is created.**

   **Using metadata derived from a type library, the runtime creates both the COM object being called and a wrapper for that object.**

   **Each RCW maintains a cache of interface pointers on the COM object that it wraps.**

   **The runtime then performs garbage collection on the RCW.**

3. Describe the role of the RCW in interoperability.

   **Maintains object lifetime.**

   **Marshals method calls between managed and unmanaged code.**

   **Consumes selected COM interfaces without exposing them to the .NET client.**

   **Allows developers to write code that treats COM objects wrapped by the RCW like any other object.**

4. What are the two most common methods used for generating interop assemblies?

   **Visual Studio .NET IDE and TLBIMP tool.**

5. List the functions of the Platform invoke service.

   **Locates the DLL containing the function.**

   **Loads the DLL into memory.**

   **Locates the address of the function in memory and pushes its arguments onto the stack, marshaling data as required.**

   **Transfers control to the unmanaged function.**

   **Returns exceptions generated by the unmanaged function to the managed caller.**

6. How do you call a Win32 API from a .NET-based application?

   **1. Determine the name of the function you want to call, its arguments, argument types, return value, and the name and location of the DLL that contains it.**

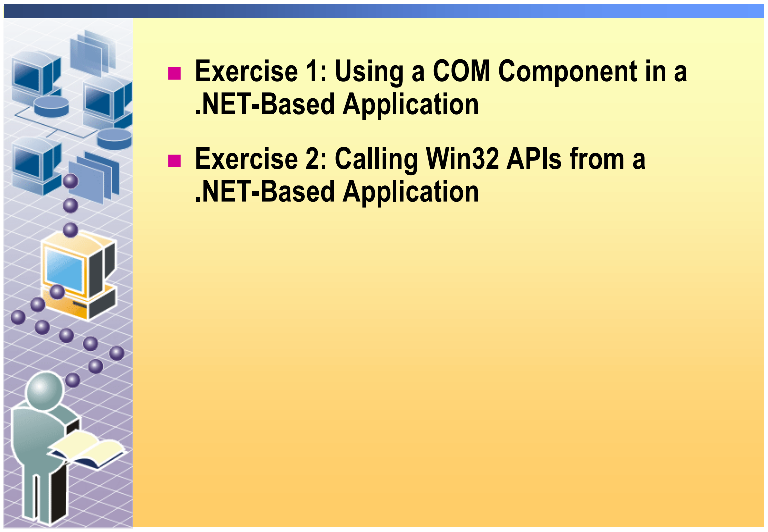
   **2. Create a new class in Visual Studio .NET.**

   **3. Import the System.Runtime.InteropServices namespace.**

   **4. Define a function by using DllImport.**

   **5. Add code to call the Win32 API from a Windows Form.**

# Lab 5.1: Interoperating with COM and Calling Win32 APIs



- Exercise 1: Using a COM Component in a .NET-Based Application
- Exercise 2: Calling Win32 APIs from a .NET-Based Application

**Objectives**

After completing this lab, you will have demonstrated your ability to:

- Create an **Interop** assembly.
- View assembly information with ILDASM.
- Invoke functions on a COM component from managed code.
- Declare functions that expose Win32 DLLs.
- Invoke Win32 DLLs from managed code.

---

**Note** This lab focuses on the concepts in Module 5, “Interoperating with Managed Objects,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. As a result, this lab may not comply with Microsoft security recommendations.

---

**Prerequisites**

Before working on this lab, you must have:

- The knowledge and skills to develop a simple Windows Forms application by using a Visual Studio .NET–compatible programming language.
- The knowledge and skills to create an **Interop** assembly from the Visual Studio .NET IDE.
- The knowledge and skills to invoke COM functions from a Visual Studio .NET–based application.
- The knowledge and skills to expose Win32 DLLs in a Visual Studio .NET-based application.
- The knowledge and skills to invoke Win32 DLLs in a Visual Studio .NET-based application.

**Scenario**

You are a developer in a trading company called Northwind Traders. The department that you work in is developing a purchase order application that will be used by the Northwind Traders sales force. As you develop the Purchase Order application, you realize that there are some required functionalities that are either too difficult to include in the first version of the application or are inaccessible from the .NET Framework. One of these functionalities is the ability to do a live lookup on the database to determine how many units of a specified product remain in the inventory. The other functionality is to include sound files that play when a user successfully completes an operation.

You realize that you can re-use existing COM components in your application. A COM component called NorthwindData_COM already exists, and it performs the required lookup task. In addition, NorthwindData_COM was installed with a previous application on all the client computers on which the Purchase Order application will be installed.

You also realize that you can expose and invoke the Win32 APIs that play media files and use them in the Purchase Order application.

**Estimated time to complete this lab: 30 minutes**

# Exercise 1
# Using a COM Component in a .NET-Based Application

In this exercise, you will create a reference to a COM object, view the **Interop** assembly by using ILDASM, and invoke the COM component from a Windows Forms application.

## Scenario

The sales staff of Northwind Traders has put in a request that the Purchase Order application be able to display the number of units in inventory for a specified product. Without knowing how much inventory is available, it is difficult for the sales staff to estimate the time it will take to deliver the order. By knowing the number of units in stock, they can anticipate late deliveries and warn the customer in advance.

You decide to use an existing COM component that was installed on the sales staff's laptops with a previous application. The existing COM component is called NorthwindData_COM, and it includes a class called **RemainingInventoryClass**. The **RemainingInventoryClass** class includes a method called **ShowUnitsInStock** that returns a string that identifies the product name and the number of units remaining in inventory.

All the functions that are available in NorthwindData_COM use ADO 2.6 and require a live connection to the database, so the salesperson must be connected to the database for the component to return the expected results. You intend to provide a disconnected feature that provides a similar function in the next version of the Purchase Order application.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab05_1\Ex01\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab05_1\Ex01\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open Visual Studio .NET, and open the PurchaseOrderApplication.sln file. To open the solution file, browse to *install_folder*\Labfiles\Lab05_1\Ex01\Starter\OrderApplication. | **a.** For more information about opening a project file, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. |
| **2.** Register the NorthwindData_COM.dll located in *install_folder*\Labfiles\Lab05_1\Ex01\Starter\ using the **REGSVR32** utility and add a reference to the component.<br>When you use the IDE to create a reference to a COM component, it uses TLBIMP to create an **Interop** assembly.<br>• If you already registered this component in Practice: Using COM Components in .NET Applications in Module 5, “Interoperating with Managed Objects,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*, then just add a reference to the NorthwindData_COM.dll component. | **a.** For more information about the adding a reference and creating an **Interop** assembly, see the following resources:<br>• Lesson: Using .NET and COM Components in a Windows Forms Application in Module 5, “Interoperating with Managed Objects,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Using COM Components in .NET-Based Applications in Module 5, “Interoperating with Managed Objects,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C#  .NET)*.<br>• The Visual Studio .NET Help documentation. For additional information about creating References, search by using the phrase **Adding and Removing References**.<br>• The.NET Framework SDK documentation. For additional information about creating **Interop** assemblies, search by using the phrase **Importing a Type Library as an Assembly**. |
| **3.** Run ILDASM, and view the information for Interop.NorthwindData_COM.dll. This DLL is located in *install_folder*\Labfiles\Lab05_1\Ex01\Starter\OrderApplication\Bin\Debug.<br>Interop.NorthwindData_COM.dll is the **Interop** assembly that was generated when you added a reference to the NorthwindData_COM.dll. | **a.** For more information about how to use ILDASM, see the following resource:<br>• The .NET Framework SDK documentation. Search by using the phrase **MSIL Disassembler (Ildasm.exe)**. |

| Tasks | Additional information |
|---|---|
| **4.** Open MainForm in the Code Editor. Use the Task List to locate the code section 'TODO 1. Create an instance of the **NorthwindData_COM RemainingInventoryClass** COM component. | **a.** For more information about using COM components in managed code, see the following resources:<br>• Lesson: Using .NET and COM Components in a Windows Forms Application in Module 5, “Interoperating with Managed Objects,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Using COM Components in .NET-Based Applications in Module 5, “Interoperating with Managed Objects,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. For additional information about using COM components in .NET code, search by using the phrase **Using COM Types in Managed Code**. |
| **5.** Use the Task List to locate the code section 'TODO 2. Create an instance of the **OrderItemControl** control and assign it the **SourceControl** property of the **ProductContextMenu** control. | **a.** For more information about using the **SourceControl** method of the **ContextMenu** class, see the following resource:<br>• The.NET Framework SDK documentation. Search by using the phrase **ContextMenu.SourceControl Property**. |
| **6.** Use the Task List to locate the 'TODO 3. Create a variable of the type **Short** and assign it the **OrderProductID** property of the **OrderItemControl**. | **a.** For more information about the **OrderItemControl**, see the following resource:<br>• Lab 3.1: Building Controls in Module 3, “Building Controls,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. |
| **7.** Use the Task List to locate the code section 'TODO 4. Create a **try**, **catch** block and call the **ShowUnitsInStock** method of the **RemainingInventoryClass** class, passing the Short variable containing the **ProductID** and the name of a SQL Server with the Northwind database installed. Display the results in a message box.<br>The name of the SQL Server uses syntax similar to *LONDON\MOC*. | **a.** For more information about creating a Try Catch block and using COM components in managed code, see the following resources:<br>• Lesson: Using .NET and COM Components in a Windows Forms Application in Module 5, “Interoperating with Managed Objects,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Using COM Components in .NET-Based Applications in Module 5, “Interoperating with Managed Objects,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The.NET Framework SDK documentation. For additional information about using Try Catch blocks, search by using the phrase **Using the Try/Catch Block to Catch Exceptions**.<br>• The Visual Studio .NET SDK documentation. For help with using COM components in managed code, search by using the phrase **Using COM Types in Managed Code**. |

| Tasks | Additional information |
|---|---|
| **8.** Compile and run the application. Add an **OrderItemControl** control by clicking the **New Order Item** button. Right-click the ComboBox within the **OrderItemControl**, and click **Show Inventory**. Repeat this for other products.<br>A message box appears that displays the current number of units in stock. | **a.** For more information about building and debugging your applications, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Default and Custom Builds** and **Using the Debugger**. |

# Exercise 2
# Calling Win32 APIs from a .NET-Based Application

In this exercise, you create a class that exposes a Win32 API. You will then invoke the Win32 API by using your class from a Windows Forms application. The **PlaySound** function takes three arguments and returns a Boolean value that indicates whether or not the function was able to play the sound file. The arguments include:

**pszSound**

A string that defines the name of the sound file.

**hmod**

Handle to the executable file that contains the resource to be loaded. This parameter must be **NULL** unless **SND_RESOURCE** is specified in **fdwSound**.

**fdwSound**

Flags that are used to determine how the sound is played. Examples of flags include whether the sound should play asynchronously or whether it should loop.

## Scenario

You decide to add sound effects to the Purchase Order application that will notify users when they successfully save an order and when they successfully update their orders to the database. However, the Win32 APIs that are responsible for playing media files are not exposed by the .NET Framework. You must create a class that exposes the winmm.dll and allows you to call the PlaySound API.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab05_1\Ex02\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab05_1\Ex02\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open Visual Studio .NET. Open the PurchaseOrderApplication.sln file. To open the solution file, browse to *install_folder*\Labfiles\Lab05_1\Ex02\Starter\OrderApplication. | **a.** For more information about opening a project file, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. |
| **2.** Add the Save.wav sound file located in *install_folder*\Labfiles\Lab05_1\Ex02\Starter to the Bin directory of the PurchaseOrderApplication project. | **a.** For more information about adding new items to a project, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Adding Projects and Items to the New Application**. |

| Tasks | Additional information |
|---|---|
| **3.** Add a new class to the PurchaseOrderApplication project and name it **Win32PlaySound.cs**. | **a.** For more information about adding new items to a project, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Adding Projects and Items to the New Application**. |
| **4.** Create a function that exposes the Win32 **PlaySound** function in the winmm.dll. | **a.** For more information about creating functions that expose WIN32 DLLs, see the following resources:<br>• Lesson: Calling Win32 APIs from Windows Forms Applications in Module 5, "Interoperating with Managed Objects," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Calling Win32 APIs in Module 5, "Interoperating with Managed Objects," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. For help with using COM components in managed code, search by using the phrase **Consuming Unmanaged DLL Functions**. |
| **5.** Open MainForm in the Code Editor. Use the Task List to locate the code section 'TODO 1. Use the **Win32PlaySound** class and call the **Win32 PlaySound** function if the **soundOn** Boolean value is **True**. Call the **PlaySound** function with "**Save.wav"** as the name of the media file, a **0** for the second argument, and a **0** for the third argument. | **a.** For more information about creating functions that expose WIN32 DLLs, see the following resources:<br>• Lesson: Calling Win32 APIs from Windows Forms Applications in Module 5, "Interoperating with Managed Objects," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Calling Win32 APIs in Module 5, "Interoperating with Managed Objects," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. For help with using COM components in managed code, search by using the phrase **Consuming Unmanaged DLL Functions**. |
| **6.** Compile and run the application. Click the **New Order Item** button, and then click the **Save Order** button to save an order.<br>You should hear a sound play. If sound is not available with your computer, test the Boolean value returned by the **PlaySound** function; this value indicates success or failure. | **a.** For more information about building and debugging your applications, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Default and Custom Builds** and **Using the Debugger**. |

msdn® training

# Module 6: Printing and Reporting in Windows Forms Applications

**Contents**

Microsoft®

# Instructor Notes

**Presentation:**
**90 minutes**

**Lab:**
**45 minutes**

This module provides students with an overview of how to print and create reports in Windows Forms applications, which are a part of the new Microsoft® .NET Framework.

After completing this module, students will be able to:

- Print documents in a Windows Forms application.
- Use the printing dialog boxes of Microsoft Visual Studio® .NET in a Windows Forms application.
- Use GDI+ to construct print document content.
- Create and format reports by using Crystal Reports.

**Required materials**

To teach this module, you need the Microsoft PowerPoint® file 2555A_06.ppt.

**Preparation tasks**

To prepare for this module:

- Read all of the materials for this module.
- Complete the demonstration, practices and lab.

# How to Teach This Module

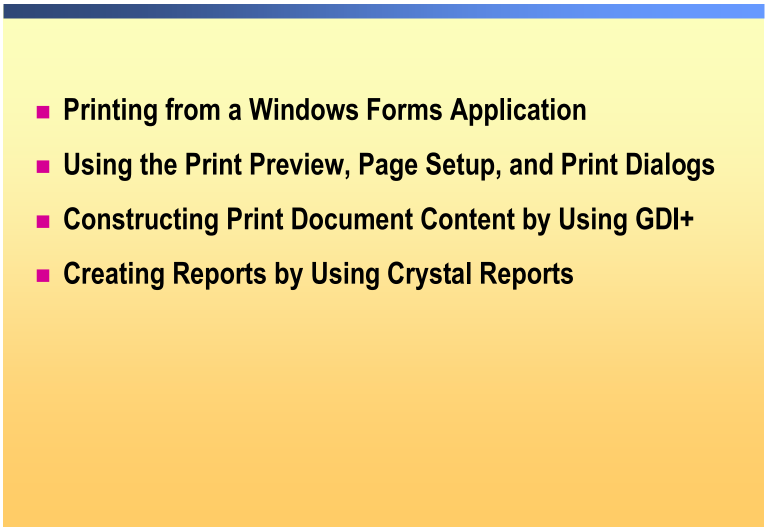
This section contains information that will help you to teach this module.

- The duration of this module is approximately 90 minutes. However, if you are running short of time, you should go through the slides quickly and then let students do the practices. If students can complete their practices, they will be able to meet the objectives of the module.
- Before you start this module, find out how many students use Crystal Reports. If this lesson is not important to your students, you can skip it.
- Lab 6.1: Printing Formatted Documents is based on the Purchase Order application in Course 2555A, *Developing Microsoft .NET Applications for Windows® (Visual C#™ .NET)* and is intended to simulate a real-world environment in which students will demonstrate what they learned during the lecture and practice portions of the module. The lab does not provide step-by-step detailed instructions; instead, the students are given tasks to complete in the left column and a list of resources that they can use (if they need help) in the right column. Students get the hands-on experience that they need by completing the practice activities at the end of each lesson.

## Lab 6: Printing Formatted Documents

- Make sure that you have demonstrated the two lab applications—the Expense Report application and the Purchase Order application—in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)* before students begin the lab. To see how to demonstrate the lab scenarios, see the Introduction module in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*

# Overview



- Printing from a Windows Forms Application
- Using the Print Preview, Page Setup, and Print Dialogs
- Constructing Print Document Content by Using GDI+
- Creating Reports by Using Crystal Reports

**Introduction**

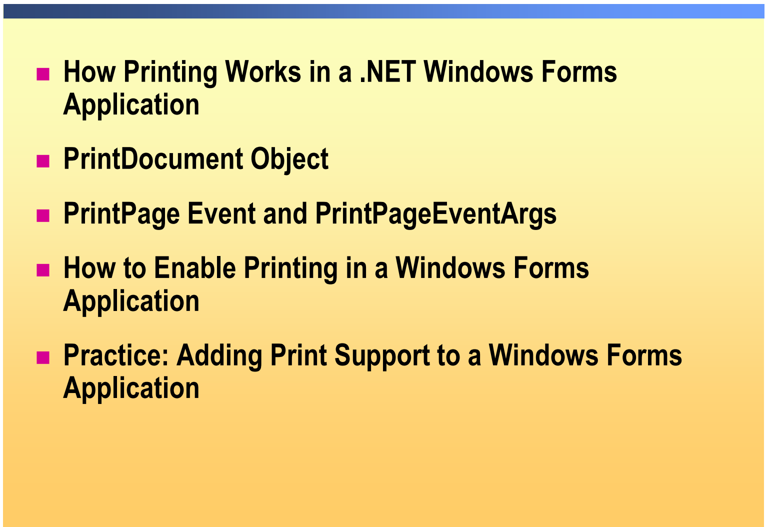
Any application developed for the Microsoft® Windows® operating system that deals with data should include printing and reporting features for the users. This module explores how to implement printing in a Windows Forms application and how to create reports in a Windows Forms application by using Crystal Reports for Microsoft Visual Studio® .NET.

**Objectives**

After completing this module, you will be able to:

- Print documents in a Windows Forms application.
- Use the Visual Studio .NET printing dialog boxes in a Windows Forms application.
- Use GDI+ to construct print document content.
- Create and format reports by using Crystal Reports.

# Lesson: Printing from a Windows Forms Application



- How Printing Works in a .NET Windows Forms Application
- PrintDocument Object
- PrintPage Event and PrintPageEventArgs
- How to Enable Printing in a Windows Forms Application
- Practice: Adding Print Support to a Windows Forms Application

**Introduction**

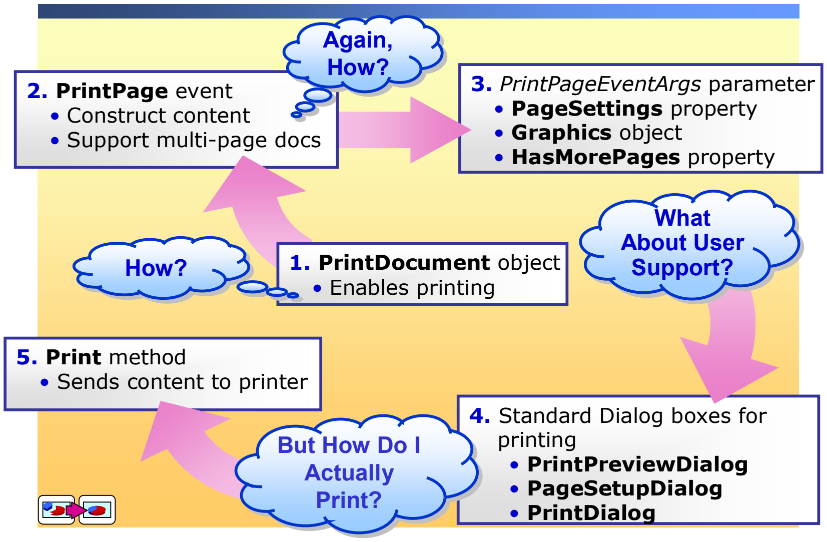
This lesson describes how to add basic print support to a Windows Forms application. Most applications developed for Windows include the ability to print some information. Visual Studio .NET provides components that simplify the print process in your Windows Forms application and provides you with control throughout the entire print process.

**Lesson objectives**

After completing this lesson, you will be able to:

- Add a **PrintDocument** control to a form and create a **PrintPage** event handler.
- Add programming logic to a **PrintPage** event procedure to construct page content.
- Add programming logic to a **PrintPage** event procedure that enables your application to print more than one page of content.

# How Printing Works in a .NET Windows Forms Application



**1. PrintDocument** object
- Enables printing

**2. PrintPage** event
- Construct content
- Support multi-page docs

**3.** *PrintPageEventArgs* parameter
- **PageSettings** property
- **Graphics** object
- **HasMorePages** property

**4.** Standard Dialog boxes for printing
- **PrintPreviewDialog**
- **PageSetupDialog**
- **PrintDialog**

**5. Print** method
- Sends content to printer

### Introduction

Application developers who use Windows Forms will use the **PrintDocument** class to support printing in their .NET Windows Forms applications. However, the **PrintDocument** object is not the only component that you will be using when printing documents in Windows Forms applications.

### Print procedure

1. Add the **PrintDocument** object.

   The **PrintDocument** object is central to printing in Windows Forms applications.

2. Create the **PrintPage** Event handler.

   The **PrintDocument** object uses the program logic that you create in the **PrintDocument.PrintPage** event to construct the content of the print document and indicate when additional pages must be generated.

3. Use the *PrintPageEventArgs* parameter.

   The **PrintPage** event uses the *PrintPageEventArgs* parameter that it receives to access and store information about the print document. Applications will generally use the **PageSettings** property to read information about document layout, construct page content by drawing text and graphics on the **Graphics** object, and then indicate when additional pages must be generated by setting the **HasMorePages** property.

**Note** More information about constructing print content by using the **Graphics** object supplied by the *PrintPageEventArgs* parameter is included in the lesson titled Constructing Print Document Content by Using GDI+ in Module 6, "Printing and Reporting in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C#™ .NET)*.

4. Use the standard print dialog boxes available in Visual Studio .NET.

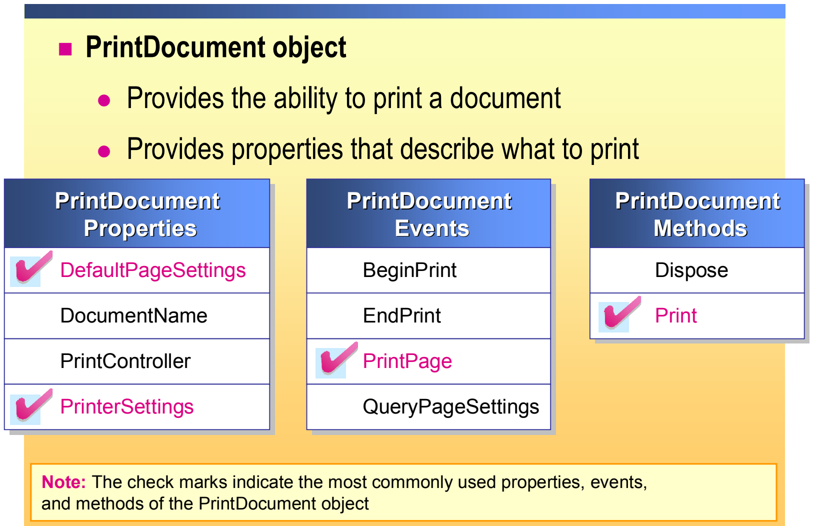
   User support can be added by using the three standard dialog boxes that are provided in the Design view of Toolbox. The standard dialog boxes provide an easy way of adding powerful end user support in your applications with a familiar user interface (UI).

   **Note** More information about **PrintPreviewDialog**, **PageSetupDialog**, and **PrintDialog** is included in the lesson titled Using the Print Preview, Page Setup, and Print Dialogs in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C#  .NET)*.

5. Print the document by using the **Print** method.

   The **PrintDocument.Print** method is used to enable actual document printing.

# PrintDocument Object



- PrintDocument object
  - Provides the ability to print a document
  - Provides properties that describe what to print

| PrintDocument Properties | PrintDocument Events | PrintDocument Methods |
|---|---|---|
| ✓ DefaultPageSettings | BeginPrint | Dispose |
| DocumentName | EndPrint | ✓ Print |
| PrintController | ✓ PrintPage | |
| ✓ PrinterSettings | QueryPageSettings | |

**Note:** The check marks indicate the most commonly used properties, events, and methods of the PrintDocument object

## Introduction

The **PrintDocument** object is central to printing in Windows Forms applications. You use **PrintDocument** to set properties that specify what to print and to provide the properties, events, and methods that are required to print documents. The **PrintDocument** object is added to a Form from the Toolbox at design time, but an instance of the **PrintDocument** class can be declared at run time as well.

## PrintDocument properties

The **DefaultPageSettings** property gets or sets page settings that are used as defaults for all pages to be printed.

The **PrinterSettings** property can be used to get or set the printer that prints the document, modify the properties of the selected printer, and modify settings associated with the print job such as the number of copies that will be printed.

The following table describes some additional properties of the **PrintDocument** object that you can use to customize a print job.

| Property | Description |
|---|---|
| **DocumentName** | Gets or sets the document name to display while printing the document. |
| **PrintController** | Gets or sets the print controller that guides the printing process. |

For more information and a complete list of the properties of the **PrintDocument** object, in the Visual Studio .NET Help documentation, search by using the phrase **PrintDocument members**.

### PrintDocument events

The **PrintPage** event is used to generate the content of the print document, and it is in the **PrintPage** event handler that you must include your own code to indicate when the document has additional pages of content to print.

The following table describes some additional events of the **PrintDocument** object that enable you to print output.

| Event | Description |
|---|---|
| **BeginPrint** | Occurs when the **Print** method is called and before the first page of the document prints. One example of when to use **BeginPrint** is when you want to notify the user about how many pages there are in a print job. |
| **EndPrint** | Occurs when the last page of the document has printed. One example of when **EndPrint** can be used is when you want to signal the user that the print job has completed. |
| **QueryPageSettings** | Occurs immediately before each **PrintPage** event. You can use **QueryPageSettings** when you want to use different **PageSettings** for one or more pages of a print document. |

For more information and a complete list of the events of the **PrintDocument** object, in the Visual Studio .NET Help documentation, search by using the phrase **PrintDocument members**.

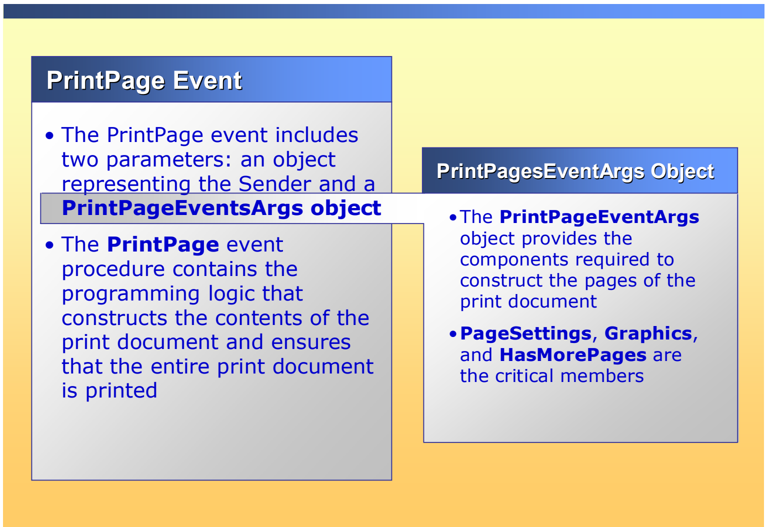
### PrintDocument methods

After you have established the printer and default page settings and constructed the contents of the print document, you will use the **Print** method to start the print process. The **Print** method sends the contents of the print document to the printer by passing the print device a **Graphics** object that acts as a container for the content. The **Graphics** object is discussed in more detail in the PrintPage Event and PrintPageEventArgs topic in this module.

The **Dispose** method releases the resources used by the PrintDocument component.

For more information and a complete list of the methods of the **PrintDocument** object, in the Visual Studio .NET Help documentation, search by using the phrase **PrintDocument members**.

# PrintPage Event and PrintPageEventArgs



**PrintPage Event**

- The PrintPage event includes two parameters: an object representing the Sender and a **PrintPageEventsArgs object**
- The **PrintPage** event procedure contains the programming logic that constructs the contents of the print document and ensures that the entire print document is printed

**PrintPagesEventArgs Object**

- The **PrintPageEventArgs** object provides the components required to construct the pages of the print document
- **PageSettings**, **Graphics**, and **HasMorePages** are the critical members

### Introduction

A **PrintDocument.PrintPage** event occurs for every page of a print document that is displayed or printed. You create a procedure to handle the **PrintPage** event and populate it with the code used to construct the content of the print document.

### PrintPage event procedure

The event procedure that handles the **PrintDocument.PrintPage** event has two parameters: the object representing the sender that fired the event (**PrintDocument**) and the **PrintPageEventArgs** object. The procedure that you create to handle the **PrintPage** event contains the programming logic required to construct the print document (using the components supplied by the **PrintPageEventArgs** object). The **PrintPage** event procedure also contains the programming logic that you develop to indicate when an additional page of content must be generated.

### Members of the *PrintPageEventArgs* parameter

The *PrintPageEventArgs* parameter contains all the data required to construct a page of the print document. The **PrintPageEventArgs** object has six member components, including the **Graphics** object that is used as the "paper" on which the contents of a page are drawn. The other two most critical members are the **HasMorePages** property that you must set to **True** when there are more pages to print, and the **PageSettings** property that can be used to read the page setting values for the current page.

The other three properties of the *PrintPageEventArgs* parameter are **Cancel**, **MarginBounds**, and **PrintBounds**. You can use the **Cancel** property to stop the construction of document pages before the end of the print document and the **PagesBounds** and **MarginBounds** properties to return the size of the current page and the size of the page inside the margins, respectively (both of which can be determined by using the **PageSettings** property).

### PrintPageEventArgs.PageSettings

The **PageSettings** property supplied by the *PrintPageEventArgs* parameter is a read-only property that is used to get page settings for the current page. **PageSettings** includes **Bounds** and **Margins** members that supply the same information found in the **PageBounds** and **MarginBounds** properties.

**Note** To modify page settings, you must handle the **QueryPageSettings** event of the **PrintDocument** object.

The following example demonstrates how to use the **PageSettings** and **MarginBounds** properties inside a **PrintPage** event handler to determine the location of the left margin and the number of text lines that will fit on a page for a given font.

```
…
LeftMargin = e.PageSettings.Margins.Left;
LinesPP = e.MarginBounds.Height/myFont.GetHeight(e.Graphics);
…
```

### PrintPageEventArgs.Graphics

The **Graphics** object supplied by the *PrintPageEventArgs* parameter acts as a canvas for each page of a print document. You use the GDI+ drawing methods of the **Graphics** object to draw the text and graphics contents of the page. For example, to draw text inside a **PrintPage** event handler, you could use the **Graphics.DrawString** method as follows:

```
…
  e.Graphics.DrawString(" text will go here.", myFont,
myBrush, X, Y);
…
```

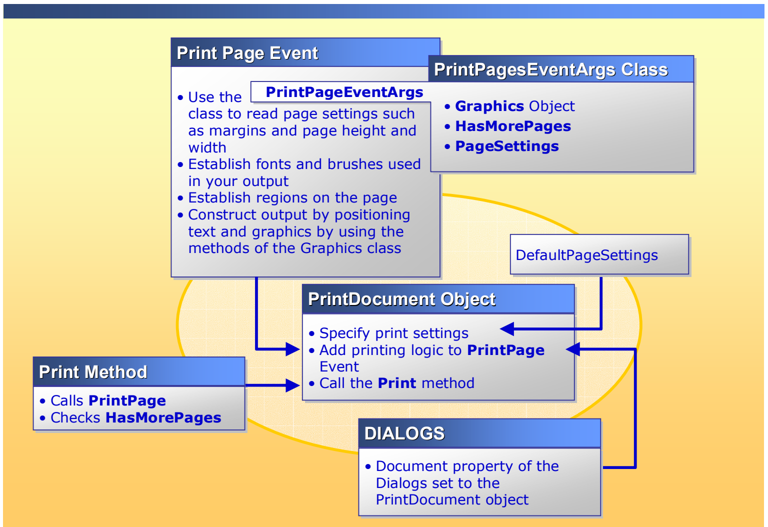
**Note** More information about constructing print content by using the **Graphics** object supplied by the *PrintPageEventArgs* parameter is included in the lesson titled Constructing Print Document Content by Using GDI+ in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C#  .NET).*

### PrintPageEventsArgs. HasMorePages

The **HasMorePages** property supplied by the *PrintPageEventArgs* parameter is a Boolean value that is **False** by default. You must manually set the value of the **HasMorePages** property to **True** to indicate that an additional page should be printed. The following example demonstrates how to set the **HasMorePages** property to **True** when the next line of text would occur below the bottom margin on the page.

```
// for each text line that will fit on the page,
//    read a new line from the document
while (CurrentLine < LinesPerPage)
{
  TextLine = StreamToPrint.ReadLine();
  if(TextLine == null)
  {
      break;
  }

  // set the vertical position on the page based
  //   on the current line number
  VerticalPosition = TopMargin +
      CurrentLine * MyFont.GetHeight(e.Graphics);

  // draw the text on the page
  e.Graphics.DrawString(TextLine,
                MyFont,
                MyBrush,
                HorizontalPosition,
                VerticalPosition);

  // increment the line number
  CurrentLine += 1;
}

// If more lines of text exist in the file,
//    print another page.
if (TextLine != null)
{
  e.HasMorePages = true;
}
else
{
  e.HasMorePages = false;
}
```

# How to Enable Printing in a Windows Forms Application



Print Page Event

- Use the PrintPageEventArgs class to read page settings such as margins and page height and width
- Establish fonts and brushes used in your output
- Establish regions on the page
- Construct output by positioning text and graphics by using the methods of the Graphics class

PrintPagesEventArgs Class

- **Graphics** Object
- **HasMorePages**
- **PageSettings**

DefaultPageSettings

PrintDocument Object

- Specify print settings
- Add printing logic to **PrintPage** Event
- Call the **Print** method

Print Method

- Calls **PrintPage**
- Checks **HasMorePages**

DIALOGS

- Document property of the Dialogs set to the PrintDocument object

## Introduction

To print from a Windows Form application, you will use a **PrintDocument** object to define the printer and characteristics of the print job, add programming logic to the **PrintPage** event, provide your users with run-time support (by using the standard dialog boxes provided in Visual Studio .NET), and then call the **Print** method.

## Procedure: Enabling printing in a Windows Forms application

To print a text document from a Windows Form application, you perform the following procedure:

1. Add an instance of the **PrintDocument** class to your application and create a **PrintPage** event procedure.

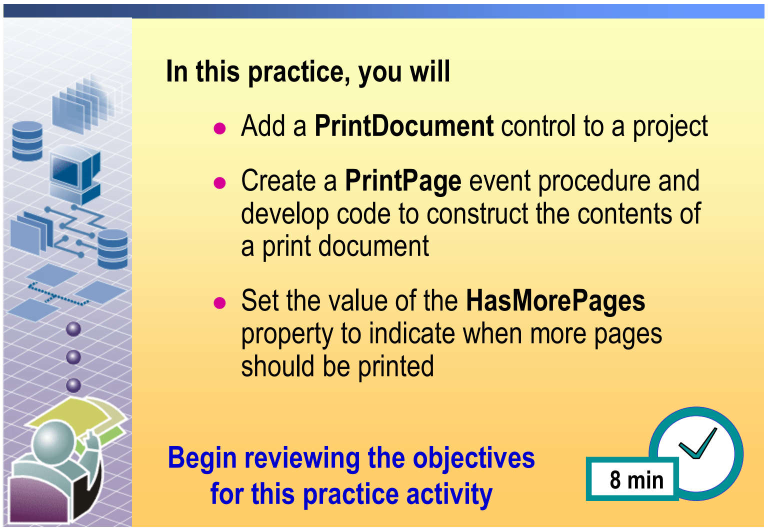
   To add a **PrintDocument** control to a Form, open the Toolbox, and then double-click **PrintDocument**. To create a **PrintPage** procedure, open the Design view of the Form, click **printDocument1**. Then, in the Properties window, click the **Events** button, and then double-click **PrintPage**.

2. Begin adding programming logic to the **PrintDocument1_PrintPage** procedure to construct the content of the print document.

   You can use the **PageSettings** property to define text and graphics regions on the page. A simple example is assigning the value of PageSettings.Margins.Left to a variable that will be used to position text on the left side of the page. You must use the GDI+ drawing methods of the **PrintPageEventArgs.Graphics** object to construct the contents of the print document page.

3. Add support for previewing your print document so that you can test the code that you developed up to this point. One simple way to preview your print document is to add a **PrintPreviewControl** to your Form from the Toolbox.
4. Add additional programming logic to the **PrintDocument1_PrintPage** procedure that uses the **HasMorePages** property to indicate whether or not more pages must be printed. The manner in which you determine whether **HasMorePages** should be set to **True** depends on how your print document is being constructed.

# Practice: Adding Print Support to a Windows Forms Application



**In this practice, you will**

- Add a **PrintDocument** control to a project
- Create a **PrintPage** event procedure and develop code to construct the contents of a print document
- Set the value of the **HasMorePages** property to indicate when more pages should be printed

**Begin reviewing the objectives for this practice activity**

### Introduction

In this practice, you will add print support to a Windows Forms application. The practice involves adding a **PrintDocument** control to a project, creating a procedure to handle the **PrintPage** event, adding programming logic to the PrintPage procedure, and using the **HasMorePages** property. You will see some questions in between some steps. Try and answer the question before you move to the next step.

### Instructions

► **Open the practice project**

1. Use Windows Explorer to browse to *install_folder*\Practices\Mod06\Mod06_01\Starter.

> **Note** If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

2. Double-click the PrintProcess.sln solution file to open the project.

### ► Add the PrintDocument control to your application

1. In Solution Explorer, click **Form1.cs**, and then click the **View Designer** icon.
2. In the **Toolbox**, scroll down until you see the **PrintDocument** control, and then double-click **PrintDocument**.
3. What functionality does the **PrintDocument** class provide to your application?

   **The PrintDocument class enables a Windows Forms application to initiate the print process (by calling the PrintDocument.Print method) and provides access to printer and page settings that can be used to control the appearance of your print document.**

   ____________________________________________________________

   ____________________________________________________________

   ____________________________________________________________

### ► Create a custom PrintPage procedure

1. In Solution Explorer, click **Form1.cs**, and then click the **View Code** icon.
2. On the **View** menu, point to **Show Tasks**, and then click **Comment**.
3. In the Task List, double-click **TODO: Add PrintPage event constructor**.
4. Add the following code statement below the TODO line and press ENTER.

```
this.printDocument1.PrintPage +=
   new System.Drawing.Printing.PrintPageEventHandler(
   this.MyPrintPage);
```

5. What is the purpose of the **PrintDocument.PrintPage** event?

   **You will use the procedure that handles the PrintDocument.PrintPage event to construct each page of a print document individually, to control the number of print document pages that are created, to adjust the page settings for each page of a print document, and to cancel a print job when required. The Graphics object of the *PrintPageEventArgs* parameter is used to construct the contents of each page individually. The Graphics object will be passed to the print device by using the PrintDocument.Print method.**

   ____________________________________________________________

   ____________________________________________________________

   ____________________________________________________________

6. In the Task List, double-click **TODO: create the PrintPage event handler**.
7. Add the following code statements below the TODO line and press ENTER.

```
private void MyPrintPage(object sender,
   System.Drawing.Printing.PrintPageEventArgs e)
{
   e.Graphics.DrawString("Page " + currentPage.ToString() +
       " text will go here.", myFont, myBrush, X, Y);

   if (currentPage < totalPages)
   {
       currentPage += 1;

   }
}
```

8. Press F5, and then click **Print Preview**.
9. Close the **PrintPreviewDialog** dialog box.
10. Use the **NumericUpDown** control to add a second page to your print document, and then preview your document again.
11. Why doesn't your application display more than one page?

    **The HasMorePages property of the PrintPageEventArgs object is False by default. You must develop the programming logic for a print loop when the print job includes more than one document page.**

12. Close the PrintProcess application.

### ► Create a print loop to ensure that all of the document pages are printed

1. Modify the contents of the **if** statement near the bottom of your **MyPrintPage** procedure so that it appears as follows:

```
if (currentPage < totalPages)
{
   currentPage += 1;
   e.HasMorePages = true;

}
else
{
   e.HasMorePages = false;
}
```

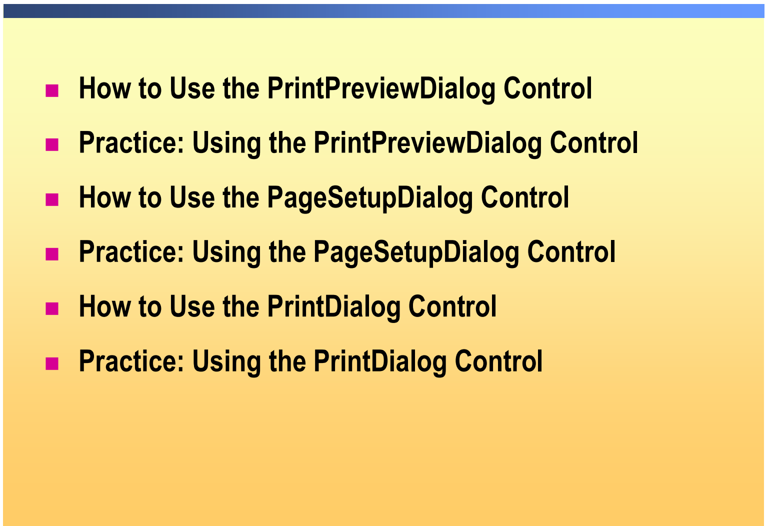
2. Start the PrintProcess application.
3. Use the **NumericUpDown** control to add a second page to your document, and then preview your print document again. You should now be able to see both pages of your print document.

### ► Close your application and Visual Studio .NET

If time permits, use the two buttons at the bottom of Form1 to display some additional information that is available through the PrintDocument object, and then examine the underlying code.

1. Close the PrintProcess application.
2. Save the PrintProcess application.
3. Close Visual Studio .NET.

# Lesson: Using the Print Preview, Page Setup, and Print Dialogs



- How to Use the PrintPreviewDialog Control
- Practice: Using the PrintPreviewDialog Control
- How to Use the PageSetupDialog Control
- Practice: Using the PageSetupDialog Control
- How to Use the PrintDialog Control
- Practice: Using the PrintDialog Control

### Introduction

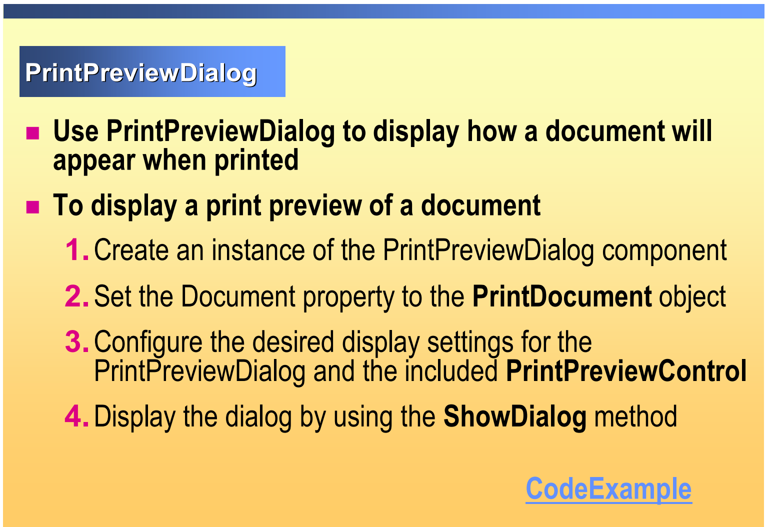
Visual Studio .NET includes three preconfigured dialog boxes—**PrintPreviewDialog**, **PageSetupDialog**, and **PrintDialog**—that enable your application user to preview a print document, specify page settings, and establish the printer and print job settings from a Windows Forms application. Using these preconfigured dialog boxes in a Windows Forms application not only eases development, but also provides a clean, efficient, and familiar interface to the end users. This lesson explains how to use the three print dialog boxes in a Windows Forms application.

### Lesson objectives

After completing this lesson, you will be able to:

- Display a print preview of the print document by using the **PrintPreviewDialog** and **PrintPreviewControl** controls.
- Use a **PrintPreviewControl** control to preview your print document.
- Specify page settings for a document by using the **PageSetupDialog** control.
- Select a printer and settings for the print job by using the **PrintDialog** control.

# How to Use the PrintPreviewDialog Control



PrintPreviewDialog

- **Use PrintPreviewDialog to display how a document will appear when printed**
- **To display a print preview of a document**
  1. Create an instance of the PrintPreviewDialog component
  2. Set the Document property to the **PrintDocument** object
  3. Configure the desired display settings for the PrintPreviewDialog and the included **PrintPreviewControl**
  4. Display the dialog by using the **ShowDialog** method

CodeExample

### Introduction

A common feature of applications is the ability to display a preview of the document to be printed. In Windows Forms applications, this feature is provided by using the **PrintPreviewDialog** control. The **PrintPreviewDialog** contains a **PrintPreviewControl** as well as controls for printing, zooming, displaying one or multiple pages, and closing the dialog box. The standard **PrintPreviewDialog** provides the user with a familiar tool for previewing documents, but you can develop your own custom preview form by using the same **PrintPreviewControl** used by the **PrintPreviewDialog** when your application or users require something different.

### Procedure

To use the **PrintPreviewDialog** to display a print preview of a document:

1. Add a **PrintPreviewDialog** control to your project by using the Toolbox, or create a **PrintPreviewDialog** at run time by using code such as this:

```
PrintPreviewDialog previewDialog = new
                  PrintPreviewDialog();
```

2. Set the **Document** property of the **PrintPreviewDialog** control to the PrintDocument component.

   After you have created a **PrintPreviewDialog** object, you must set its **Document** property to the **PrintDocument** object of your print document. You can set the **Document** property in the Properties window of the **PrintPreviewDialog** control at design-time or with the following code at run time:

```
previewDialog.Document = PrintDocument1;
```

3. Properties of the **PrintPreviewDialog**, and the included **PrintPreviewControl**, can be used to establish display settings showing the dialog box:

```
previewDialog.WindowState =
              FormWindowState.Maximized;
previewDialog.PrintPreviewControl.StartPage = 0;
previewDialog.PrintPreviewControl.Zoom = 1.0;
```

4. Display the **PrintPreviewDialog** by using the **ShowDialog** method.

   Like all other dialog boxes, you can use the **ShowDialog** method to display the **PrintPreview** dialog box at run time:

```
previewDialog.ShowDialog();
```

### Example

The following code shows an example of how to use the **PrintPreviewDialog** control to display a print preview of a document. The example assumes that a **PrintPreviewDialog** control and a **PrintDocument** control were added to a Form and that the **PrintPreview** procedure is called from an event handler.

```
private void PrintPreview()
{
  // ensure that the first page of the print document is shown
  printPreviewDialog1.PrintPreviewControl.StartPage = 0;

  // display the document two pages at a time
  printPreviewDialog1.PrintPreviewControl.Columns = 2;

  // maximize the size of the dialog box on the display screen
  printPreviewDialog1.WindowState = FormWindowState.Maximized;

  // display the contents of the appropriate print document
  printPreviewDialog1.Document = printDocument1;
  printPreviewDialog1.ShowDialog();

}
```

### PrintPreviewControl

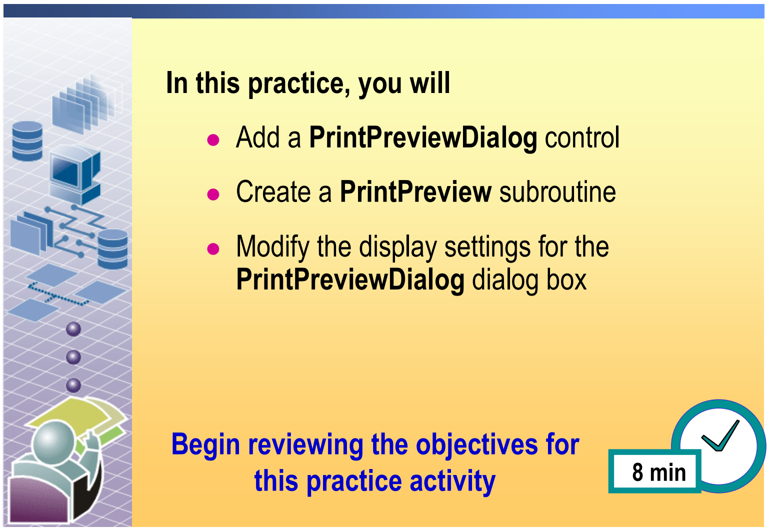
The **PrintPreviewControl** is supplied in the Toolbox as a separate control that you can use to preview a print document. You use the **PrintPreviewControl** control when you want to define your own print-preview user interface. It has no buttons or other user interface elements.

Some of the properties of the **PrintPreviewControl** control include **Zoom**, **Columns**, **Rows**, and **StartPage**. These properties are available whether you are using the **PrintPreviewControl** separately or as part of the **PrintPreviewDialog**.

```
PrintPreviewControl.Columns = 2;
PrintPreviewDialog.PrintPreviewControl.Columns = 2;
```

For more information about **PrintPreviewControl**, in the Visual Studio .NET Help documentation, search by using the phrase **PrintPreviewControl**.

# Practice: Using the PrintPreviewDialog Control



In this practice, you will

- Add a **PrintPreviewDialog** control
- Create a **PrintPreview** subroutine
- Modify the display settings for the **PrintPreviewDialog** dialog box

Begin reviewing the objectives for this practice activity

8 min

## Introduction

In this practice, you will add a **PrintPreviewDialog** control and program logic to a project so that application users can preview a document before printing it.

## Instructions

### ► Open the practice project

1. Use Windows Explorer to browse to *install_folder*\Practices\Mod06\Mod06_02\Starter.
2. Double-click the PrintPreviewDialog.sln solution file to open the project.

### ► Add the PrintPreviewDialog control to your application

1. In Solution Explorer, click **Form1.cs**, and then click **View Designer**.
2. In the Toolbox, scroll down until you see the **PrintPreviewDialog** control, and then double-click **PrintPreviewDialog**.
3. Where does the **PrintPreviewDialog** appear in the Design View window?

   **The PrintPreviewDialog control is added to the component tray at the bottom of the Design View window.**

### ► Create a PrintPreview procedure that displays the print document

1. In Solution Explorer, click **Form1.cs**, and then click **View Code**.
2. On the **View** menu, point to **Show Tasks**, and then click **Comment**.
3. In the Task List, double-click **TODO: create PrintPreview procedure**.
4. Add the following code statements below the TODO line.

```
private void PrintPreview()
{
   printPreviewDialog1.Document = printDocument1;
   printPreviewDialog1.ShowDialog();
}
```

5. Run the PrintPreviewDialog application, and use the menu or button to display the print document.
6. Is there a way to make the dialog box larger by default?

   **Yes. The WindowState property of the dialog box can be used to set the default window state to Maximized.**

   ____________________________________________________________

7. Close the PrintPreviewDialog application.

### ► Modify the display settings for the PrintPreviewDialog dialog box

1. Add the following code line to the top of the **PrintPreview** procedure.

```
printPreviewDialog1.WindowState =
      FormWindowState.Maximized;
```

2. Run the PrintPreviewDialog application, and use the menu or button to display the print document.
3. Change the displayed page number to **page 4**, and then close the dialog box.
4. Redisplay the print document. Notice that the starting page is the page that was displayed when the dialog box was closed.
5. Close the application, and then add the following code lines to the top of the **PrintPreview** procedure.

```
printPreviewDialog1.PrintPreviewControl.StartPage = 0;
printPreviewDialog1.PrintPreviewControl.Zoom = 1.0;
```

6. Run the PrintPreviewDialog application, and use the menu or button to display the print document. Verify that the **Print Preview** dialog box always shows page 1 at 100% when opened.
7. Optional: Open the **PrintPreviewDialog**, change the page that is being displayed to **page 2**, and then close and reopen the preview dialog box to verify that the initial page displayed is always page 1.
8. Close the dialog box, and then close the application.

**If time permits**

### ► Examine some additional methods for displaying a print document

1. In Design view, enable the two other buttons on Form1.
2. Start the application, and use each button to display the print document.
3. Close the application, and examine the code used to display the document.
4. When would you want to use a **PrintPreviewControl** rather than a **PrintPreviewDialog** to display a print document?

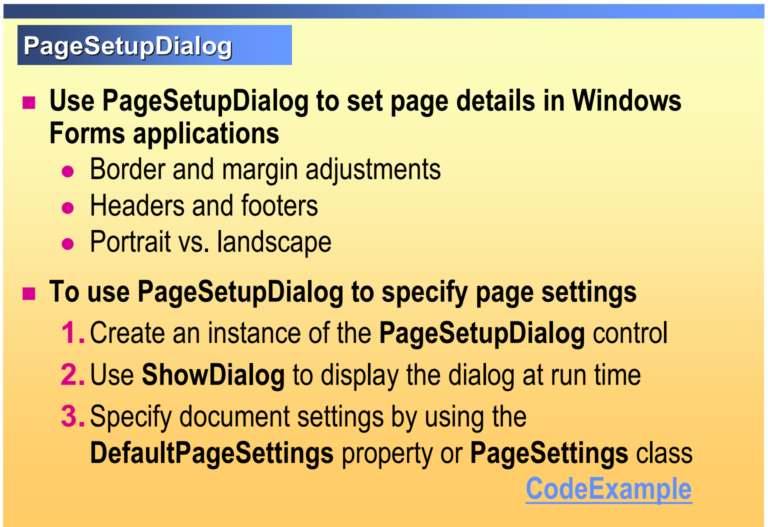
   **You would use a PrintPreviewControl to display a document when you need to customize the print preview capabilities of an application.**

   ____________________________________________________________

   ____________________________________________________________

5. Close the solution.

# How to Use the PageSetupDialog Control



PageSetupDialog

- **Use PageSetupDialog to set page details in Windows Forms applications**
  - Border and margin adjustments
  - Headers and footers
  - Portrait vs. landscape
- **To use PageSetupDialog to specify page settings**
  1. Create an instance of the **PageSetupDialog** control
  2. Use **ShowDialog** to display the dialog at run time
  3. Specify document settings by using the **DefaultPageSettings** property or **PageSettings** class

CodeExample

## Introduction

The **PageSetupDialog** control is a pre-configured dialog box that you can use to set page layout details for printing in Windows Forms applications. The **PageSetupDialog** control provides an easy solution to specify page settings instead of configuring your own dialog box. With **PageSetupDialog**, you can allow users to set border and margin adjustments, headers and footers, and portrait versus landscape orientation by using a familiar tool.

## Procedure: Using the PageSetupDialog

To use **PageSetupDialog** to specify page settings:

1. Create an instance of **PageSetupDialog**.

   The first step in using the **PageSetupDialog** in a Windows Forms application is to add a **PageSetupDialog** control to your form from the Toolbox or create a new instance of the **PageSetupDialog** control in code as shown below.

   ```
   PageSetupDialog PageDlg = new PageSetupDialog();
   ```

2. Supply **PageSetupDialog** with a **PageSettings** object that can be used to store page settings and used for both the **PrintDocument** object and the **PageSetupDialog**.

   ```
   …
   PageDlg.PageSettings = MyPageSettings;
   ```

3. Use the **ShowDialog** method to display the dialog box at run time.

   To display the dialog box at run time, use the **ShowDialog** method as shown in the following code.

   ```
   PageDlg.ShowDialog();
   ```

4. Apply the modified page settings to your **PrintDocument** object.

   To apply the new page settings to your document, use the **DefaultPageSettings** property of the **PrintDocument** object.

   ```
   printDocument1.DefaultPageSettings = MyPageSettings;
   ```

### Example

The following code shows an example of how to use the **PageSetupDialog** control.

```
private void PageSetup()
{
  try {
      PageSetupDialog pageDialog = new PageSetupDialog();

      if (storedPageSettings == null) {
          storedPageSettings = new PageSettings();
      }

      pageDialog.PageSettings = storedPageSettings ;
      pageDialog.ShowDialog();

  }

  catch(Exception ex) {
      MessageBox.Show("An error occurred - " + ex.Message);
  }

}
```

### Customize sections of the PageSetup dialog box

The user can enable sections of the **PageSetup** dialog box to manipulate printing, margin, and paper orientation, and size.

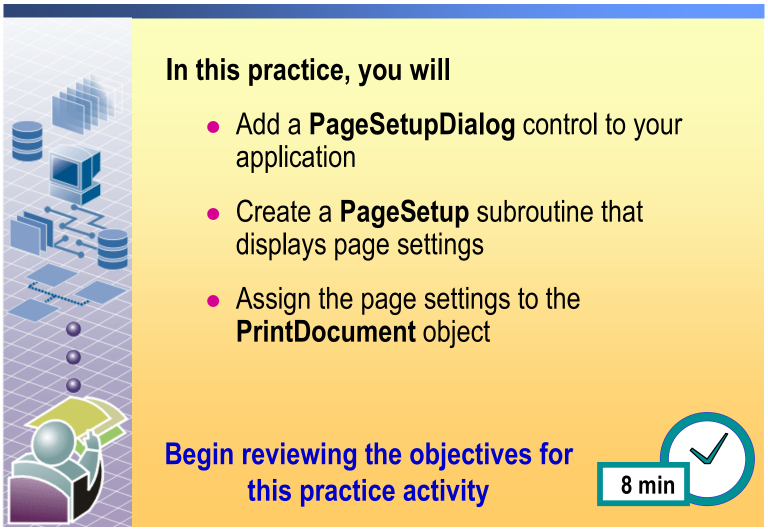
Use the **Margins** and **MinMargins** properties to specify margins.

```
pageSetupDialog1.PageSettings.Margins.Top = 200;
pageSetupDialog1.PageSettings.Margins.Left = 200;
pageSetupDialog1.PageSettings.Margins.Bottom = 100;
pageSetupDialog1.PageSettings.Margins.Right = 100;

pageSetupDialog1.MinMargins.Top = 85;
pageSetupDialog1.MinMargins.Left = 75;
pageSetupDialog1.MinMargins.Bottom = 100;
pageSetupDialog1.MinMargins.Right = 100;
```

Set the **AllowPrinter**, **AllowOrientation**, and **AllowPaper** properties to **True** to allow users to specify these properties.

```
pageSetupDialog1.AllowOrientation = false;
pageSetupDialog1.AllowPaper = false;
pageSetupDialog1.AllowPrinter = false;
```

# Practice: Using the PageSetupDialog Control



**In this practice, you will**

- Add a **PageSetupDialog** control to your application
- Create a **PageSetup** subroutine that displays page settings
- Assign the page settings to the **PrintDocument** object

**Begin reviewing the objectives for this practice activity**

### Introduction

In this practice, you will add a **PageSetupDialog** control and program logic to a project so that application users can specify page settings before printing a document.

### Instructions

**► Open the practice project**

1. Use Windows Explorer to browse to *install_folder*\Practices\Mod06\Mod06_03\Starter.
2. Double-click the PageSetupDialog.sln solution file to open the project.

**► Add the PageSetupDialog control to your application**

1. In Solution Explorer, click **Form1.cs**, and then click **View Designer**.
2. In the Toolbox, scroll down until you see the **PageSetupDialog** control, and then double-click **PageSetupDialog**.
3. Where does the **PageSetupDialog** control appear in the Design View window?

   **The PageSetupDialog control is added to the component tray at the bottom of the Design View window.**

   ____________________________________________________________

### ► Create a PageSetup procedure that displays page settings

1. In Solution Explorer, click **Form1.cs**, and then click **View Code**.
2. On the **View** menu, point to **Show Tasks**, and then click **Comment**.
3. In the Task List, double-click **TODO: create PageSetup procedure**.
4. Add the following code statements below the TODO line.

```
private void PageSetup()
{
   pageSetupDialog1.ShowDialog();
}
```

5. Run the PageSetupDialog application, and use the menu or button to display the **Page Setup** dialog box. Why does an error occur?

   **The PageSetupDialog control requires that you supply a PageSettings object that represents the page settings for the document.**

6. Close the application, and then add the following code lines to the top of the PageSetup procedure.

```
myPageSettings = new PageSettings();
pageSetupDialog1.PageSettings = myPageSettings;
```

7. Run the PageSetupDialog application, and display the **Page Setup** dialog box.
8. Change the Top and Left margins to 0.5 inches, and then click **OK**.
9. Display a preview of the print document. Why didn't the new settings take effect?

   **Although the PageSetupDialog automatically places the new settings into the MyPageSettings object, you still have to assign the page settings to the PrintDocument object.**

10. Close the **PrintPreviewDialog** and the application.

### ► Assign the page settings to the PrintDocument object

1. Add the following code lines to the bottom of the **PageSetup** procedure.

```
if (myPageSettings != null)
{
   printDocument1.DefaultPageSettings = myPageSettings;
}
```

2. Run the PageSetupDialog application, and use the menu or button to display the **PageSetup** dialog box.
3. Change the Top and Left margins to 0.5 inches, and then preview the print document.
4. Reopen the **PageSetup** dialog box. Notice that the page settings are lost.

5. Click **Cancel**, close the application, and then, in the **PageSetup** procedure, replace the "myPageSettings = new PageSettings();" code line with the following code lines.

```
if (myPageSettings == null)
{
   myPageSettings = new PageSettings();
}
```

6. Run the PageSetupDialog application, and use the **PageSetup** dialog box to modify the document's page settings. Notice that settings are not lost when the dialog box is closed.
7. Close the application.

**If time permits**

► **Examine some additional members of the PageSetupDialog control**

1. In the Code Editor, examine the code in the **pageSetupButton_Click** procedure.
2. Remove the comment characters from the front of the code lines one section at a time, and view the changes to the **PageSetup** dialog box and the **Print Preview** dialog box by running the application and opening the two dialog boxes.
3. How could you use these additional members of the **PageSetupDialog** control?

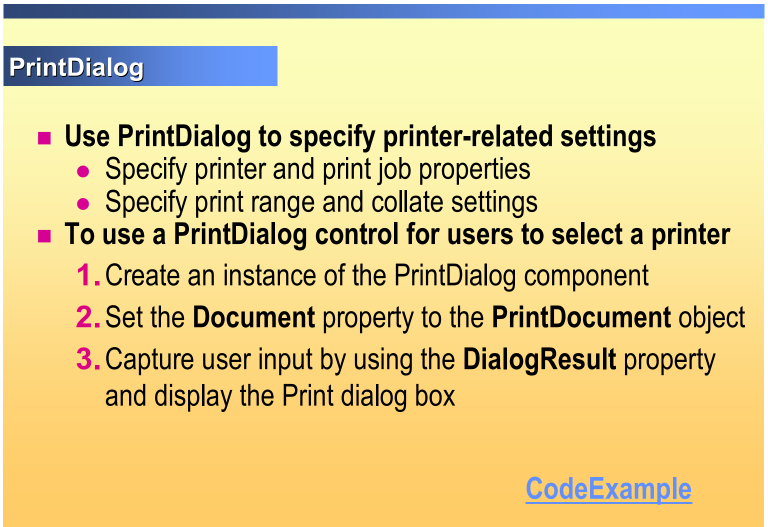
   **These additional members could be used to customize the PageSetupDialog and control the range of settings that a user is allowed to select.**

   ____________________________________________________________

   ____________________________________________________________

4. Close the application and Visual Studio .NET.

# How to Use the PrintDialog Control



PrintDialog

- **Use PrintDialog to specify printer-related settings**
  - Specify printer and print job properties
  - Specify print range and collate settings
- **To use a PrintDialog control for users to select a printer**
  1. Create an instance of the PrintDialog component
  2. Set the **Document** property to the **PrintDocument** object
  3. Capture user input by using the **DialogResult** property and display the Print dialog box

CodeExample

**Introduction**

Often, users will need to select a printer or set print properties before a particular print job. This can be done by using the **PrintDialog** control.

**Procedure**

To use the **PrintDialog** control to print a document:

1. Add a **PrintDialog** control to your project by using the Toolbox, or create an instance of the PrintDialog control at run time as follows:

   ```
   PrintDialog printDialog = new PrintDialog();
   ```

2. Set the **Document** property of the **PrintDialog** object to the PrintDocument component.

   You must set the **Document** property of the **PrintDialog** control to the **PrintDocument** object for document you are printing. Set the **Document** property of the **PrintDialog** control at design time in the Properties window, or at run time by using code, as follows:

   ```
   printDialog.Document = printDocument1;
   ```

3. Capture the user's response to the **PrintDialog** by using a **DialogResult** object when constructing the code statement that displays the dialog box.

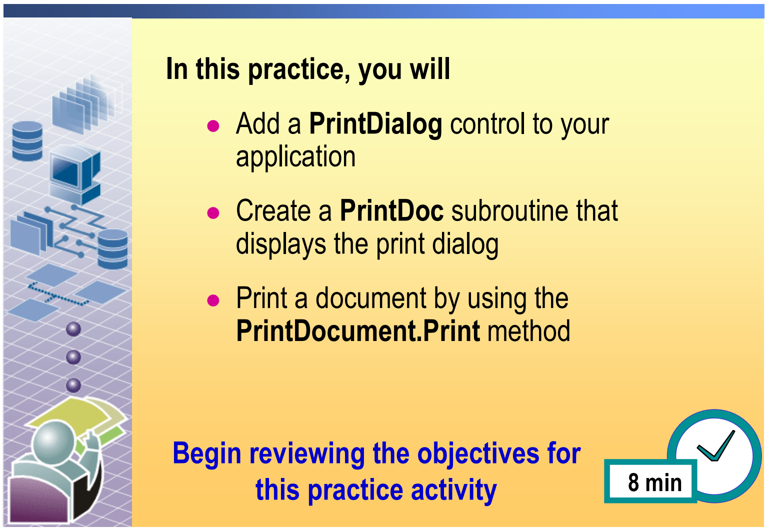
   ```
   ...
   DialogResult UserResponse = printDialog.ShowDialog();
   ```

### Example

The following code uses the **PrintDialog** control to provide the user with an opportunity to modify printer and print job settings before printing the document.

```
private void PrintDoc()
{
  PrintDialog1.Document = PrintDocument1;

  DialogResult userResponse;
  userResponse = PrintDialog1.ShowDialog();
  if (userResponse == DialogResult.OK)
  {
      PrintDocument1.Print();
  }
}
```

# Practice: Using the PrintDialog Control



**In this practice, you will**

- Add a **PrintDialog** control to your application
- Create a **PrintDoc** subroutine that displays the print dialog
- Print a document by using the **PrintDocument.Print** method

Begin reviewing the objectives for this practice activity

8 min

**Introduction**

In this practice, you will add a **PrintDialog** control and program logic to a project so that application users can specify printer and print job settings before printing a document.

**Instructions**

### ► Open the practice project

1. Use Windows Explorer to browse to *install_folder*\Practices\Mod06\Mod06_04\Starter.
2. Double-click the PrintDialog.sln solution file to open the project.

### ► Add the PrintDialog control to your application

1. In Solution Explorer, click **Form1.cs**, and then click the **View Designer** icon.
2. In the Toolbox, scroll down until you see the **PrintDialog** control, and then double-click **PrintDialog**.
3. Where does the **PrintDialog** control appear in the Design View window?

   **The PrintDialog control is added to the component tray at the bottom of the Design View window.**

   ____________________________________________________________

### ► Create a PrintDoc procedure that displays the Print dialog box

1. Open Form1 in the Code Editor.
2. On the **View** menu, point to **Show Tasks**, and then click **Comment**.
3. In the Task List, double-click **TODO: create PrintDoc procedure**.
4. Add the following code statements below the TODO line.

```
private void PrintDoc()
{
   printDialog1.ShowDialog();
}
```

5. Run the PrintDialog application, and click the menu item or button that is used display the **Print** dialog box. Why doesn't the **Print** dialog box open?

   **The PrintDialog.Document property must be set before the PrintDialog can be displayed.**

   ____________________________________________________________

6. Close the application, and then add the following code line to the top of the PrintDoc procedure.

```
printDialog1.Document = printDocument1;
```

### ► Print a document by using the PrintDocument.Print method

1. Run the PrintDialog application, and then display the **Print** dialog box.
2. Click **OK**.
3. Why didn't the document print?

   **The PrintDialog does not call the PrintDocument.Print method—you must add code to do this yourself. However, this means that you have to know which dialog box button the user clicked. You can determine this by using a DialogResult object.**

   ____________________________________________________________

   ____________________________________________________________

**Note** For more information about the DialogResult object, see the Visual Studio .NET Help documentation.

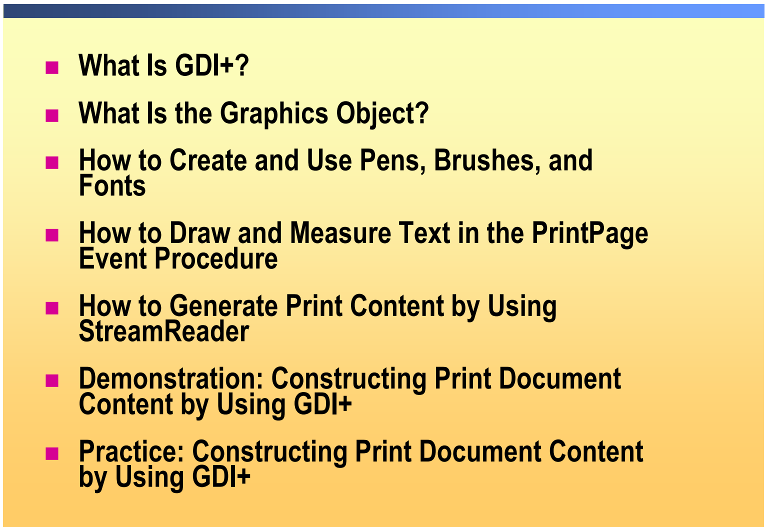
4. Close the application, and then replace the "printDialog1.ShowDialog()" code line in the PrintDoc procedure with the following lines of code.

```
DialogResult userResponse;
userResponse = printDialog1.ShowDialog();
if (userResponse == DialogResult.OK)
{
   printDocument1.Print();
}
```

5. Run the PrintDialog application, and display the print dialog box.
6. Click **OK**, and then, inside in the **Output File Name** box, type **C:\Text**
7. Click **OK**.

   Notice that a **Printing** dialog box opens automatically, providing the user with an opportunity to cancel the print job.
8. Close the application, and close Visual Studio .NET.

# Lesson: Constructing Print Document Content by Using GDI+



- What Is GDI+?
- What Is the Graphics Object?
- How to Create and Use Pens, Brushes, and Fonts
- How to Draw and Measure Text in the PrintPage Event Procedure
- How to Generate Print Content by Using StreamReader
- Demonstration: Constructing Print Document Content by Using GDI+
- Practice: Constructing Print Document Content by Using GDI+

**Introduction**

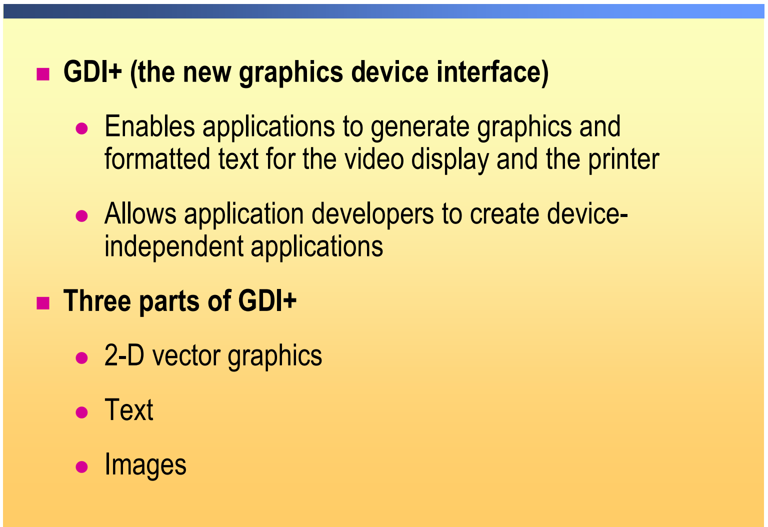
In Windows Forms applications, you use GDI+ and the **PrintPage** event of the **PrintDocument** object to construct the contents of a print document. This lesson introduces GDI+ and covers how to use GDI+ methods to draw the text and graphics content of a print document from in a .NET Windows Forms application.

**Lesson objectives**

After completing this lesson, you will be able to:

- Identify the components of GDI+ and use GDI+ in your applications.
- Use a **Graphics** object as a canvas for creating the contents of a print document.
- Create and use the pens, brushes, and fonts that are required to draw text and graphics objects.
- Use the drawing methods of a **Graphics** object to draw text and two-dimensional (2-D) vector graphics.
- Generate print content by using StreamReader.

# What Is GDI+?



- **GDI+ (the new graphics device interface)**
  - Enables applications to generate graphics and formatted text for the video display and the printer
  - Allows application developers to create device-independent applications
- **Three parts of GDI+**
  - 2-D vector graphics
  - Text
  - Images

### Introduction

GDI+ is a graphics device interface that you can use to draw two-dimensional vector graphics, text, and bitmapped images. GDI+ expands on the features of GDI by providing new features such as gradient brushes and alpha blending. GDI+ makes graphic programming easier and more flexible. It enables applications to generate graphics and formatted text for the video display and the printer. It also allows application developers to create device-independent applications. For example, you can create a single **PrintPage** procedure to construct graphical content that can be printed to most graphics printers or be shown as a print preview on a graphics display screen.

### Three parts of GDI+

The services of GDI+ fall into three main categories:

- 2-D vector graphics

  Vector graphics involves drawing lines, curves, and figures that are defined by a set of points on a coordinate system. GDI+ provides classes and structures that store information about these objects. For example, the **Rectangle** object stores information about the location and size of a rectangle, and the **Graphics** class has methods for drawing lines, curves, and other shapes.

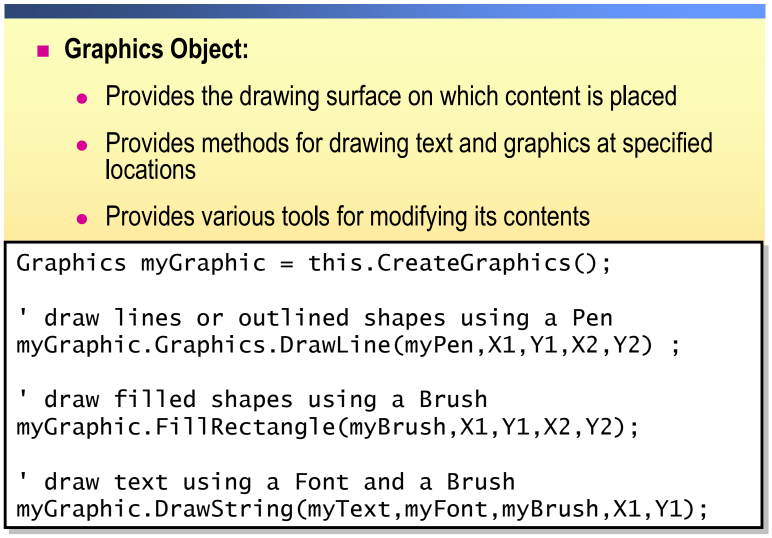
- Text

  GDI+ allows you to draw text in a variety of fonts, sizes, and styles.

- Images

  Certain graphics cannot be displayed as 2-D vector graphics and must be displayed as bitmaps. GDI+ provides the **Bitmap** class for displaying, manipulating, and saving bitmaps. For more information about bitmaps, see Appendix A, “Using Filled Shapes and Images,” in the student workbook.

# What Is the Graphics Object?



- **Graphics Object:**
  - Provides the drawing surface on which content is placed
  - Provides methods for drawing text and graphics at specified locations
  - Provides various tools for modifying its contents

```
Graphics myGraphic = this.CreateGraphics();

' draw lines or outlined shapes using a Pen
myGraphic.Graphics.DrawLine(myPen,X1,Y1,X2,Y2) ;

' draw filled shapes using a Brush
myGraphic.FillRectangle(myBrush,X1,Y1,X2,Y2);

' draw text using a Font and a Brush
myGraphic.DrawString(myText,myFont,myBrush,X1,Y1);
```

### Introduction

To draw the text, lines, and shapes that make up the content of a print document, you must use a **Graphics** object.

### Role of the Graphics object in GDI+

The **Graphics** object is central to GDI+ and the construction of print document content. It provides the drawing surface for the GDI+ drawing methods.

To construct print document content with GDI+:

- Set the properties of the **Graphics** object.
- Call the methods of the **Graphics** object to draw text, lines, and shapes.

### Graphics object methods

The **Graphics** object provides an extensive assortment of methods that can be used to draw text, lines, and shapes. There are also methods for scaling, transforming, and measuring the contents that have or will be drawn on its surface. Some of the most common methods are listed in the following table.

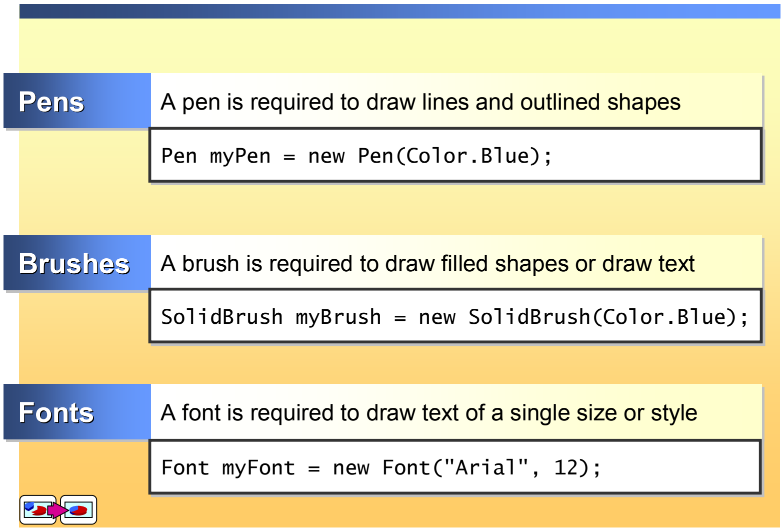
| Method | Description |
|---|---|
| **Clear** | Clears the entire drawing surface and fills it with the specified background color. |
| **DrawLine** | Draws a line connecting the two points specified by coordinate pairs. |
| **DrawRectangle** | Draws a rectangle specified by a coordinate pair, a width, and a height. |
| **DrawString** | Draws the specified text string at the specified location with the specified **Brush** and **Font** objects. |
| **FillRectangle** | Fills the interior of a rectangle. |
| **MeasureString** | Measures the specified string when drawn with the specified font and formatted with the specified format. |

## Example

The following code examples show how to draw objects by using the **Graphics** object provided by the *PrintPageEventArgs* parameter of the **PrintDocument.PrintPage** event.

```
//Create a graphics object
Graphics myGraphics = this.CreateGraphics();

// draw lines or outlined shapes using a Pen
myGraphics.DrawLine(myPen,X1,Y1,X2,Y2) ;
myGraphics.DrawRectangle(myPen, X1, Y1, X2, Y2);

// draw filled shapes using a Brush
myGraphics.FillRectangle(myBrush,X1,Y1,X2,Y2);

// draw text using a Font and a Brush
myGraphics.DrawString(myText, myFont, myBrush, X1, Y1);

// measure text width on the Graphics object using a Font
textWidth = myGraphics.MeasureString(myText, myFont).Width;
```

# How to Create and Use Pens, Brushes, and Fonts



**Pens**

A pen is required to draw lines and outlined shapes

```
Pen myPen = new Pen(Color.Blue);
```

**Brushes**

A brush is required to draw filled shapes or draw text

```
SolidBrush myBrush = new SolidBrush(Color.Blue);
```

**Fonts**

A font is required to draw text of a single size or style

```
Font myFont = new Font("Arial", 12);
```

### Introduction

You use **Pen**, **Brush**, and **Font** objects to construct 2-D vector graphics, text, and bitmap images with GDI+.

### Procedure: Creating pens

A pen is required to draw lines, curves, and outlined shapes. To create a new pen, you must specify a color. You also have the option to specify values for the width and line style properties:

1. Create a new pen.

```
Pen myPen = new Pen(Color.Blue);
```

2. Apply width and style properties.

   The **Width** property for a **Pen** is of type **Integer** and has a default value of **1** (units are established by the **Graphics** object). The **DashStyle** property is also of type **Integer**, but your code will be easier to read if you use the **DashStyle** enumeration provided by the **Drawing2D** namespace. Although the **Width** property can be set when you instantiate a new **Pen**, the line style must be applied after the **Pen** has been created. The following code examples demonstrate creating pens with various width and style properties. Defaults are 1 for Width, and Style for Solid.

```
Pen penWideRed = new Pen(Color.Red, 10);
Pen penDashedBlack = new Pen(Color.Black, 6);
penDashedBlack.DashStyle = DashStyle.Dash;
penDashedBlack.DashCap = DashCap.Round;
```

### Procedure: Creating brushes

**Brush** objects are required for drawing text and filled shapes. You can create brushes that produce Solid, Hatched, Textured, and Gradient fills.

- Create a new **Brush**.

  The following code examples demonstrate creating a solid blue brush and a linear gradient brush by using white and light blue blended horizontally.

```
Brush brushBlue = new SolidBrush(Color.Blue);
LinearGradientBrush myGrBrush = new LinearGradientBrush(
                         lgRectangle,
                         Color.White,
                         Color.LightBlue,
                         LinearGradientMode.Horizontal);
```

For more information about Hatched, Textured, and Gradient Brushes see “Brushes and Filled Shapes” in the .NET Framework software development kit (SDK) and see Appendix A, “Using Filled Shapes and Images,” in the student workbook.

### Procedure: Creating fonts

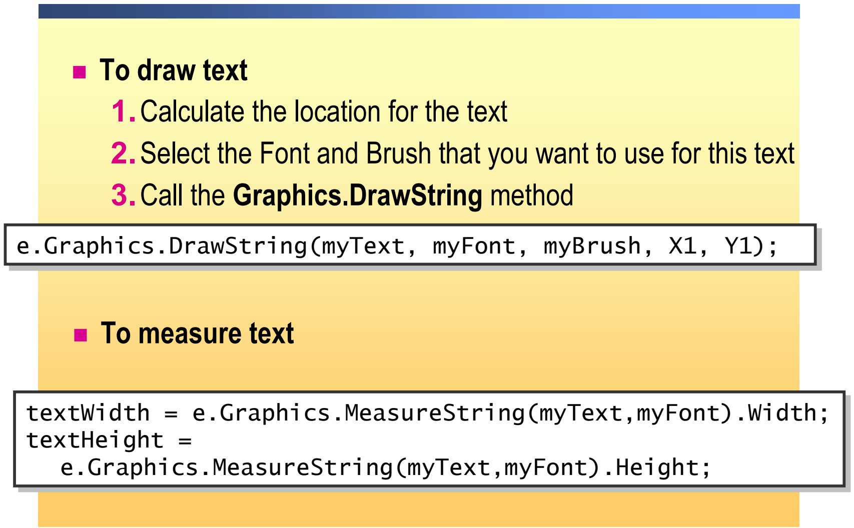
Before you can draw text with GDI+, you must construct a **Font** object. The declaration statement for a **Font** object can include parameters for the **FontFamily** (such as Arial), **Size**, **Style**, and the **GraphicsUnits** used by the **Size** parameter. The **FontFamily** and **Size** properties are required when creating a new font.

- Create a new **Font** object.

  The following code example creates an Arial font of size 10 and a Lucida font with a style setting of bold and a size of 12 millimeters.

```
Font smallFont = new Font("Arial", 10);
Font largeFont;
largeFont = new Font("Lucida",
                     12,
                     FontStyle.Bold,
                     GraphicsUnit.Millimeter);
```

# How to Draw and Measure Text in the PrintPage Event Procedure



- **To draw text**
  1. Calculate the location for the text
  2. Select the Font and Brush that you want to use for this text
  3. Call the **Graphics.DrawString** method

```
e.Graphics.DrawString(myText, myFont, myBrush, X1, Y1);
```

- **To measure text**

```
textWidth = e.Graphics.MeasureString(myText,myFont).Width;
textHeight =
  e.Graphics.MeasureString(myText,myFont).Height;
```

### Introduction

GDI+ enables you to draw and measure text.

You can draw text by using the **DrawString** method of the **Graphics** object. To draw text, you must have a **Brush** object and a **Font** object.

If you need to determine the width or height of text as it will be drawn on the page of a print document or another **Graphics** object, you can use the **MeasureString** method of the **Graphics** object. For example, you must know the width and length of a string as it appears on the page to center the string in a region of the page. **MeasureString** can also be used to determine how many lines of text will fit on a page, whether a line of text will fit between the margins, and to right-align text.

### Procedure: Drawing text

To construct text for your print document in the **PrintPage** event procedure:

1. Calculate the location on the page where the text will be placed.
2. Select the **Font** and **Brush** that you want to use for this text.

   You can either use **Fonts** and **Brushes** that you already created or create new ones. For more information about creating Fonts and Brushes, see the topic How to Create and Use Pens, Brushes, and Fonts in this lesson.
3. Call the **DrawString** method.

   To construct the text, call the **DrawString** method of the **Graphics** object.

```
e.Graphics.DrawString(myText, myFont, myBrush, X1, Y1);
```

### Procedure: Measuring text

To measure text, add the code as shown to the **PrintPage** event handler.

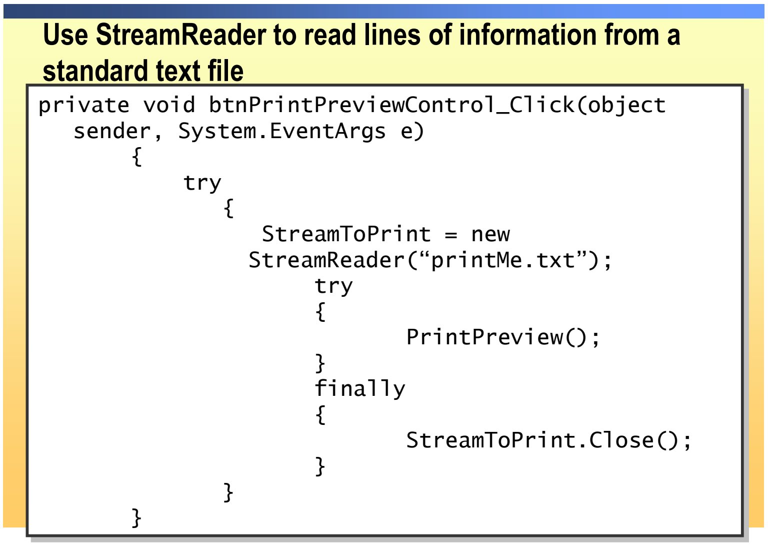
The following code measures the width and height of the specified string.

```
textWidth = e.Graphics.MeasureString(myText, myFont).Width;
textHeight = e.Graphics.MeasureString(myText,
                                       myFont).Height;
```

Another option for getting the height of your text is to use the **GetHeight** method for **Font** and pass it the **Graphics** object as follows.

```
textHeight = myFont.GetHeight(e.Graphics);
```

# How to Generate Print Content by Using StreamReader



Use StreamReader to read lines of information from a standard text file

```
private void btnPrintPreviewControl_Click(object
   sender, System.EventArgs e)
        {
            try
               {
                  StreamToPrint = new
                 StreamReader(“printMe.txt”);
                      try
                      {
                             PrintPreview();
                      }
                      finally
                      {
                             StreamToPrint.Close();
                      }
               }
        }
```

## Introduction

In addition to using GDI+ to generate print content, you can also use StreamReader as an input provider. You use StreamReader to read lines of information from a standard text file.

## Example

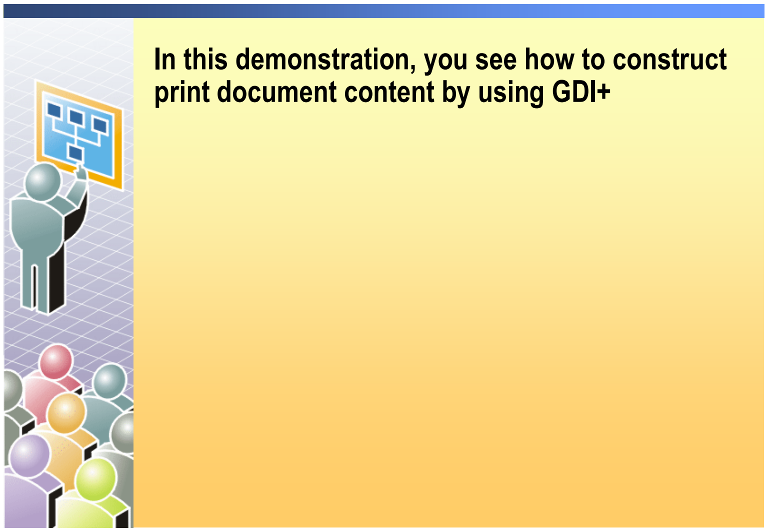
The following example demonstrates how to use StreamReader with a **PrintPreviewControl**. When the **Print Preview** button is clicked, the stream of text is read from the file PrintMe.txt.

```
private void btnPrintPreviewControl_Click(
              object sender, System.EventArgs e)
{
  try
  {
      StreamToPrint = new StreamReader(FilePath);
      try
      {
          PrintPreview();
      }
      finally
      {
          StreamToPrint.Close();
      }
  }
  catch(Exception ex)
  {
      MessageBox.Show(ex.Message);
  }
}
```

The following code shows that if no text is being read from StreamReader, the printing exits, or else it checks if there are more pages to print.

```
// inside PrintPage event handler
while (currentLine < linesPerPage)
{
  textLine = StreamToPrint.ReadLine();
  if(textLine == null)
  {
      break;
  }

  // position text, draw text, increment currentLine

}

//If more lines of text exist in the file, print another page.
if (TextLine != null)
{
  e.HasMorePages = true;
}
else
{
  e.HasMorePages = false;
}
```

# Demonstration: Constructing Print Document Content by Using GDI+



In this demonstration, you see how to construct print document content by using GDI+

**Introduction**

In this demonstration, you will see how to create print document content by using GDI+.

**Instructions**

► **Open the projects**

1. Start two instances of Visual Studio .NET.
2. Open the PurchaseOrderApplication solution file in one instance of Visual Studio .NET and the PrintingForm2555Class solution file in the other. The solution files can be found under *install_folder*\Sampapps, in the folders named OrderApplication and PrintingForm2555Class.

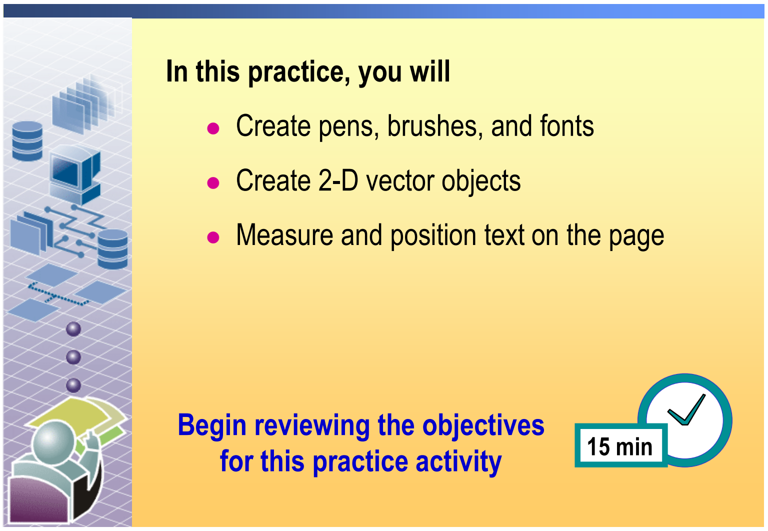
► **Examine and modify code in the PrintingForm2555Class project**

1. Open PrintingForm2555Class.cs in the Code Editor.
2. Mention that this class inherits from the **PrintDocument** class.
3. Describe the procedures that are used to: define the regions of the form, draw the text, and 2-D vector objects that make up the blank form, receive text content from the host application, and fill in the content on the form.
4. Move down to the **PrintingEmptyForm2555** procedure, and then show the code that is used to create the fonts and brushes for this document.
5. Scroll down and show some of the GDI+ code statements that draw the lines and filled shapes.
6. Scroll down further and show the process required to center the purchase item labels on the header row of the purchase item table.
7. Scroll back up to the top of this procedure and change some of the pen and brush properties.
8. Rebuild the class.

### ► Examine and run the PurchaseOrderApplication project

1. Open MainForm.cs in the Code Editor.
2. Describe the procedures that are used to: read content from the form, pass form content to the printing class, preview the print document, modify page settings, and print the document.
3. In Solution Explorer, remove the reference to the **PrintingForm2555Class** from the **PurchaseOrderApplication** project, and then add a reference to the newly built class by browsing to the bin folder of the PrintingForm2555Class folder and opening the PrintingFomr2555Class.dll file.
4. Rebuild the PurchaseOrderApplication solution, and then run the application.
5. Open the **Print Preview** dialog box to show your modifications.

# Practice: Constructing Print Document Content by Using GDI+



In this practice, you will

- Create pens, brushes, and fonts
- Create 2-D vector objects
- Measure and position text on the page

Begin reviewing the objectives for this practice activity

### Introduction

In this practice, you will construct print document content by using 2-D vectors and text in GDI+.

### Instructions

► **Open the practice project**

1. Use Windows Explorer to browse to *install_folder*\Practices\Mod06\Mod06_05\Starter.
2. Double-click the PrintPageCode.sln solution file to open the project.

► **Create pens, brushes, and fonts**

1. Open Form1 in the Code Editor.
2. On the **View** menu, point to **Show Tasks**, and then click **Comment**.
3. In the Task List, double-click **TODO: create pens**.
4. Add the following code statements below the TODO line.

```
Pen penWideRed = new Pen(Color.Red, 10);
Pen penDashedBlack = new Pen(Color.Black, 6);
penDashedBlack.DashStyle = DashStyle.Dash;
penDashedBlack.DashCap = DashCap.Round;
```

5. In the Task List, double-click **TODO: create brushes**.
6. Add the following code statements below the TODO line.

```
Brush brushBlue = new SolidBrush(Color.Blue);
Brush brushBlack = Brushes.Black;
```

7. In the Task List, double-click **TODO: create fonts**.

8. Add the following code statements below the TODO line.

```
Font fontMedium = new Font("Arial", 14);
Font fontLargeBold;
fontLargeBold = new Font("Arial", 36, FontStyle.Bold);
```

9. Do any of the code statements that you entered require a reference to the **Drawing2D** namespace?

   **Yes. The code statements used to set the DashStyle and DashCap use enumerations provided by the Drawing2D namespace.**

   ________________________________________________________

   ________________________________________________________

### ► Create 2-D vector objects

1. In the Task List, double-click **TODO: draw outline shapes**.
2. Add the following code statements below the TODO line.

```
e.Graphics.DrawLine(penWideRed,
                     hPos1,
                     vPos1,
                     hPos2,
                     vPos2);
e.Graphics.DrawRectangle(penDashedBlack,
                         hPos1,
                         vPos1,
                         rectWidth,
                         rectHeight);
```

3. In the Task List, double-click **TODO: draw filled shapes**.
4. Add the following code statements below the TODO line.

```
Rectangle fillRectangle;
fillRectangle = new Rectangle(
              lgRectangle.Left,
              lgRectangle.Bottom,
              lgRectangle.Width,
              lgRectangle.Height);
e.Graphics.FillRectangle(brushBlue, fillRectangle);
```

### ► Measure and position text on the page

1. In the Task List, double-click **TODO: draw centered text**.
2. Add the following code statements below the TODO line.

```
textWidth = e.Graphics.MeasureString(centeredText,
                         fontMedium).Width;
textHeight = fontMedium.GetHeight(e.Graphics);
hPos = rectText.Left + (rectText.Width - textWidth) / 2;
vPos = rectText.Top + (rectText.Height - textHeight) / 2;
e.Graphics.DrawRectangle(Pens.Black, rectText);
e.Graphics.DrawString(centeredText,
                     fontMedium,
                     brushBlack,
                     hPos,
                     vPos);
```

3. Run your application, and click **Print Preview** to view the GDI+ output.
4. Close the preview form, click **Draw gradient text**, and then click **Print Preview**.
5. How would you draw right-aligned text?

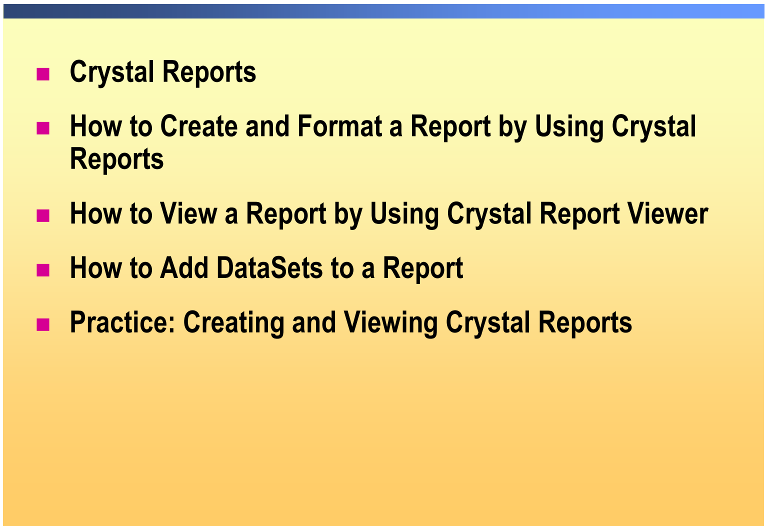
   **You can draw right-aligned text by measuring the text string and establishing a horizontal position that is equal to the right-side position that you define minus the width of the text string.**

   ____________________________________________________________

   ____________________________________________________________

   If time permits, examine the code used to create the gradient filled text.
6. Save your application, and then close Visual Studio .NET.

# Lesson: Creating Reports by Using Crystal Reports



- Crystal Reports
- How to Create and Format a Report by Using Crystal Reports
- How to View a Report by Using Crystal Report Viewer
- How to Add DataSets to a Report
- Practice: Creating and Viewing Crystal Reports

**Introduction**

There are many ways to present data to users. For example, you could write code to loop through recordsets and print them inside your Windows-based application. However, by using such methods, any work beyond basic formatting can be complicated and difficult to program.

With Crystal Reports for Microsoft Visual Studio .NET, you can quickly create complex and professional-looking reports. Instead of writing code, you use the Crystal Report Designer interface to create and format the report that you need. The powerful Report Engine processes the formatting, grouping, and charting criteria that you specify.

This lesson introduces Crystal Reports for Visual Studio .NET and also covers how to use Report Expert to create and format reports.

**Lesson objectives**

After completing this lesson, you will be able to:

- Explain how Crystal Reports helps in creating reports.
- Create and format a report by using Crystal Reports.
- View a report by using Crystal Report Viewer.
- Add datasets to a report.

# Crystal Reports

## Crystal Reports

- Is the standard reporting tool in .NET
- Allows you to create a report from the beginning or use one of the Report Expert Wizards

## Benefits

- You can use any programming language
- Report viewers for Windows-based and Web applications
- Run time customization of reports
- Easy interaction with reports
- Data visualization and analysis capabilities

http://msdn.microsoft.com/vstudio/partners/tools/crystaldecisions.asp

### Introduction

Crystal Reports is the standard reporting tool in Visual Studio .NET. It allows you to create a report from scratch or use one of the Report Expert Wizards. Before you learn how to use Crystal Reports for creating and formatting reports, you must understand the benefits of using Crystal Reports.

### Benefits of using Crystal Reports

Crystal Reports:

- Allows you to choose the language and project.

  Use the programming language of your choice and access Crystal Report Designer from any project.

- Provides report viewers for Windows-based and Web applications.

  Crystal Reports provides two report viewers that you can use to view your report in your application: Web Forms Viewer for Web applications and Windows Forms Viewer for Windows-based applications.

- Allows run-time customization of reports.

  Crystal Reports enables the viewer to interact with other controls at run time. With run-time customization, users can view different reports or change the format, data selection, or export options of an existing report.

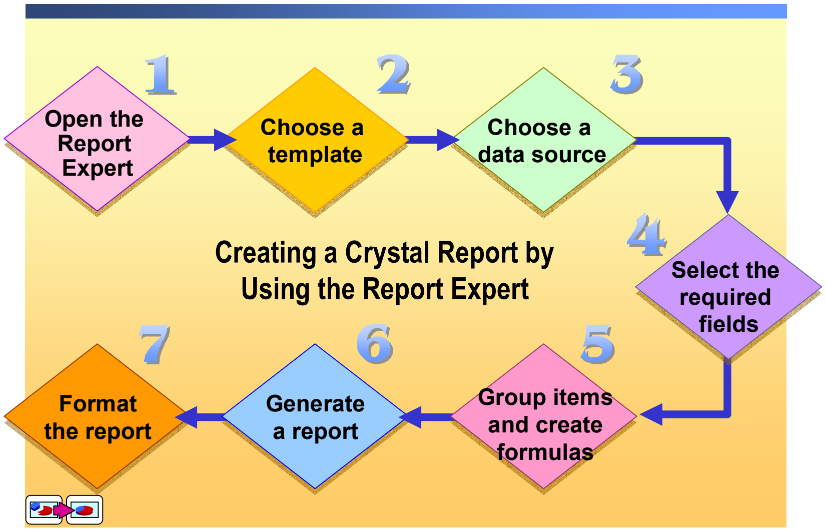
- Allows users to interact easily with reports.

  Because Crystal Reports can interact with other controls, users are able to filter report information by clicking a button or selecting from a combo box.

- Provides data-visualization and analysis capabilities.

  Crystal Reports provides developers with data visualization and analysis capabilities. It uses an open and flexible architecture—with standards like extensible markup language (XML)—to allow you to share reports and information over the Web. It also offers features such as details on charts, report navigation, and text search.

# How to Create and Format a Report by Using Crystal Reports



Creating a Crystal Report by Using the Report Expert

### Introduction

You can use Crystal Reports to add reporting capabilities to your application. Crystal Reports allows you to create a report from the beginning or use the Report Expert, which helps you create a report by using a wizard.

### Procedure: Create and format a report by using Report Expert

To create and format reports by using the Report Expert:

1. Open Report Expert.

   a. In Solution Explorer, right-click the project name, point to **Add**, and then select **Add New Item** from the shortcut menu.

   b. In the **Add New Item** dialog box, in the Templates pane, click **Crystal Report**.

   c. In the **Name** box, type a report name with an .rpt extension.

   d. Click **Open** to invoke Crystal Report Designer, which will help you create and design the new report.

   e. In Crystal Report Gallery, select the **Using the Report Expert** option.

2. Choose a template.

   Report Expert provides various templates for creating a report. Some of the templates are:

   - Standard
   - Form Letter
   - Form
   - Cross-Tab
   - Subreport
   - Mail Label
   - Drill Down
   - For more information about each of the templates, in the Visual Studio .NET Help documentation, search by using the phrase **Crystal Reports Experts**.

3. Choose a data source.

   Select the data source that your report will reference. You can use more than one data source in a report. You also choose the database tables that you want to use in the report. Crystal Reports can automatically link the tables, or you can specify how you want the tables linked. Database tables are linked so that records from one database match related records from another.

4. Select the required fields.

   After selecting the data source, select the fields that you want to display on the report.

5. Group items and create formulas.

   When you first insert a database field into your report, the data in the fields appears in the order in which it was originally entered into the database. Grouping, sorting, and totaling help turn disorganized data into useful information on a report.

   In most cases, the data needed for a report already exists in database table fields. Sometimes, you need to put additional data on the report that does not exist in any of the data fields. In cases such as this, you must create a formula. There are four different groups of formulas in Crystal Reports: report, conditional formatting, selection, and search. The majority of formulas in a report use the report and conditional formatting formulas.

   For more information about how to create and use these formulas, in the Visual Studio .NET Help documentation, search by using the phrase **Formula Overview**.

6. Generate the report.

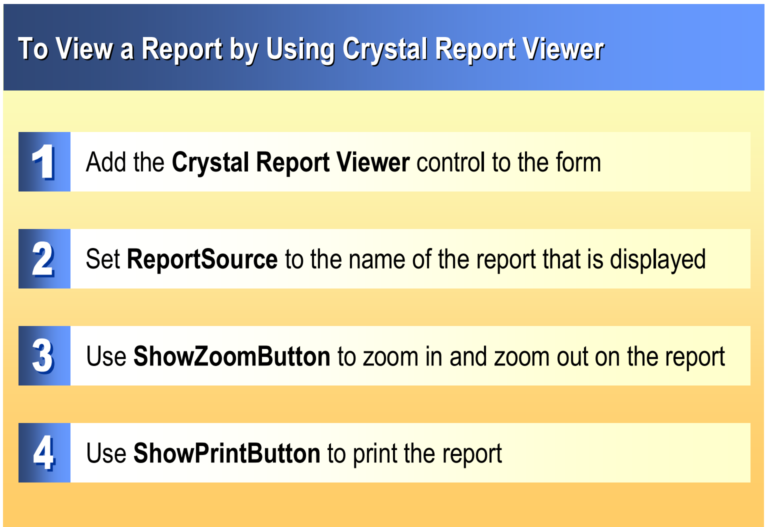
   After you have added the required fields and formulas to the report, click **Finish** to generate the report.

7. Format the report.

   After a report is generated, you might want to make changes to the layout and design of the report, as well as the appearance of text, objects, or entire report sections. You can use various formatting options to change the layout of your reports.

   - To format an object, right-click the object, and then click **Format**.
   - To format a report section, right-click the report section, and then click **Format Section**.

# How to View a Report by Using Crystal Report Viewer



To View a Report by Using Crystal Report Viewer

1 Add the **Crystal Report Viewer** control to the form

2 Set **ReportSource** to the name of the report that is displayed

3 Use **ShowZoomButton** to zoom in and zoom out on the report

4 Use **ShowPrintButton** to print the report

### Introduction

Visual Studio .NET includes a **Crystal Report Viewer** control that enables you to view a report at run time. To specify the report that you want to view, you set the **ReportSource** property of the viewer.

### Procedure: Viewing a report

To view a report by using the **Crystal Report Viewer** control:

1. Add a **Crystal Report Viewer** control to the form.

   The Controls Toolbox includes the **Crystal Report Viewer** control. To add the control to the form, drag the control from the Toolbox to the form.

2. Set the **ReportSource** property of the viewer.

   The next step is to set the **ReportSource** property of the Crystal Report viewer to the name of the report that is to be displayed. You can set the **ReportSource** property in the **Properties** window at design time or you can set it at run time.

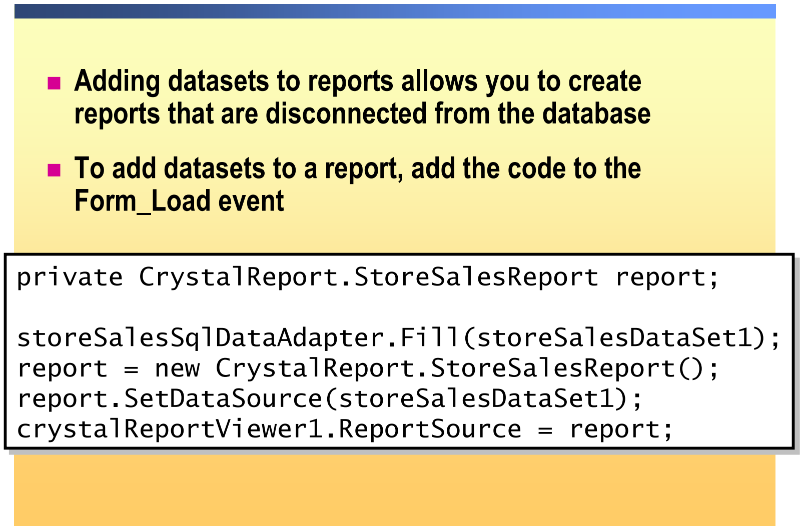
3. Use the **ShowZoomButton** property to zoom in or zoom out on reports.

   The Crystal Report viewer includes the **ShowZoomButton** to zoom in or zoom out on reports. To enable this property, set the **ShowZoomButton** property to **True** in the Properties window.

4. Use the **ShowPrintButton** property to print the report.

   The **ShowPrintButton** property enables you to print the report. Set the **ShowPrintButton** property to **True** to enable the button on the viewer.

# How to Add DataSets to a Report



- Adding datasets to reports allows you to create reports that are disconnected from the database
- To add datasets to a report, add the code to the Form_Load event

```
private CrystalReport.StoreSalesReport report;

storeSalesSqlDataAdapter.Fill(storeSalesDataSet1);
report = new CrystalReport.StoreSalesReport();
report.SetDataSource(storeSalesDataSet1);
crystalReportViewer1.ReportSource = report;
```

### Introduction

Crystal Reports for Visual Studio .NET supports reports that access ADO.NET datasets.

### Procedure: Adding an ADO.NET dataset to a report

To add an ADO.NET dataset to a report:

1. Create an instance of the report.

   To make the report display the data at run time, you must first create an instance of the report:

   ```
   private CrystalReport.StoreSalesReport report;
   report = new CrystalReport.StoreSalesReport();
   ```

2. Generate a design time instance of a dataset.

   The design time instance of the dataset exposes the fields in the dataset to Report Expert. The design time dataset contains only the data description, and not the actual data. Therefore, when working with a report connected to a dataset object, you cannot browse field data in Crystal Report Designer at design time.

3. Populate the dataset.

   ```
   storeSalesSqlDataAdapter.Fill(storeSalesDataSet1);
   ```

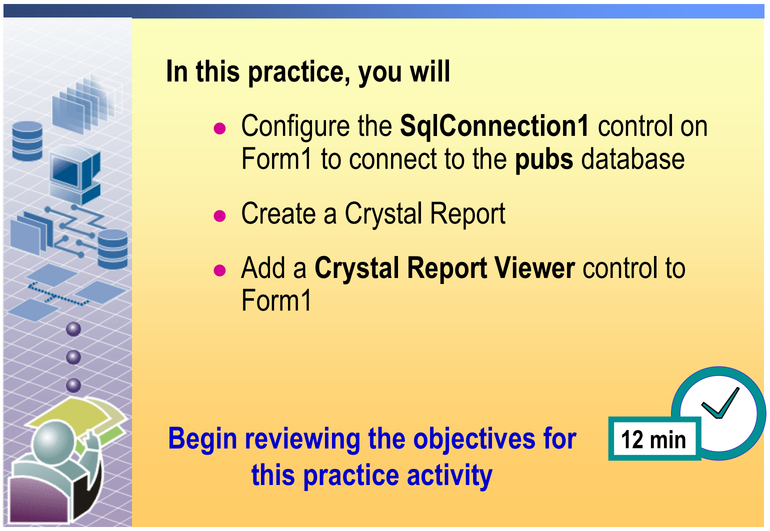
4. Call the **SetDataSource** method of the report, and pass an instance of the populated dataset.

   ```
   report.SetDataSource(StoreDataSet1);
   ```

5. Set the **ReportSource** property of the Crystal Report Viewer to the instance of the report.

   ```
   crystalReportViewer1.ReportSource = report;
   ```

# Practice: Creating and Viewing Crystal Reports



**In this practice, you will**

- Configure the **SqlConnection1** control on Form1 to connect to the **pubs** database
- Create a Crystal Report
- Add a **Crystal Report Viewer** control to Form1

Begin reviewing the objectives for this practice activity

12 min

**Introduction**

In this practice, you will create and view Crystal Reports.

**Instructions**

► **Open the practice project**

1. Use Windows Explorer to browse to *install_folder*\Practices\Mod06\Mod06_06\Starter.
2. Double-click the CrystalReport.sln solution file to open the project.

► **Configure the sqlConnection1 control on Form1 to connect to the Pubs database**

1. Open Form1 in Design view.
2. Select **sqlConnection1**, and from the Properties window, click the **ConnectionString** property, click the arrow, and then click **New Connection**.

> **Note** Use the existing connection information for the **Pubs** database if it already exists and skip to the next procedure.

3. In the **Data Link Properties** dialog box, type *computername***MOC** where *computername* is the name of your computer.
4. Select the **Use Windows NT Integrated Security** option.
5. Select the **Pubs** database from the drop-down list, click **Test Connection** to ensure that you can access the **Pubs** database, click **OK**, and then click **OK** again.

### ► Create a Crystal Report

1. On the **File** menu, click **Add New Item**.
2. On the **Add New Item** dialog box, under **Template**, click **Crystal Report**.
3. Set the name of the report to **StoreSalesReport.rpt**, and then click **Open**.
4. Close the **Crystal Decisions Registration Wizard** dialog box.
5. On the **Crystal Report Gallery** dialog box, click **OK**.
6. On the **Data** tab, expand **Project Data**, expand **ADO.NET DataSets**, and then expand **CrystalReport.storeSalesDataSet**.
7. Click **Insert Table**, and then click **Next**.
8. On the **Fields** tab, in Available Fields list, click **stor_id**, and then click **Add**.
9. Repeat step 8 to add all of the fields that are listed below **stores** table in the **Available Fields** list, and then click **Next**.
10. On the **Group** tab, in the **Available Fields** list, below **Report Fields**, click **stores.stor_name**, click **Add**, and then click **Next**.
11. On the **Total** tab, in the **Summarized Fields** list, click **stores.qty**, click **Remove**, and then click the **Style** tab.
12. Set the title to **Store Sales**, and then click **Finish**.

### ► Add a Crystal Report Viewer control to Form1

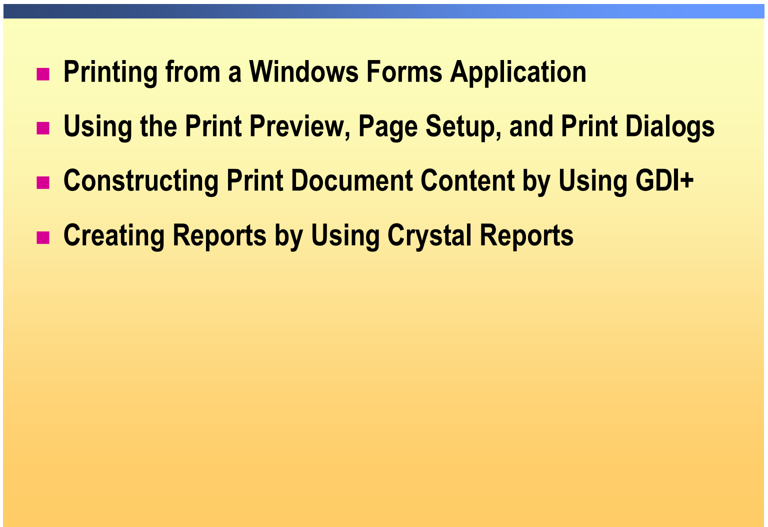
1. Open Form1 in Design view.
2. In the Toolbox, double-click **CrystalReportViewer**.
3. Set the **Dock** property of the CrystalReportViewer so that the control fills the entire form.
4. Open Form1 in the Code Editor.
5. On the **View** menu, point to **Show Tasks**, and then click **Comment**.
6. In the Task List, double-click **TODO 1: create an instance of StoreSalesReport**.
7. Add the following code statement below the TODO line:

   ```
   private CrystalReport.StoreSalesReport report;
   ```
8. In the Task List, double-click **TODO 2: Call the SetDataSource method of report and pass storeSalesDataSet1**.
9. Add the following code statements below the TODO line:

   ```
   report = new CrystalReport.StoreSalesReport();
   report.SetDataSource(storeSalesDataSet1);
   crystalReportViewer1.ReportSource = report;
   ```
10. Build and run your application.
11. Use the Crystal Report Viewer control to examine your report.

    Notice the features provided on the viewer's toolbar.
12. Save your project, and then close Visual Studio .NET.

# Review



- Printing from a Windows Forms Application
- Using the Print Preview, Page Setup, and Print Dialogs
- Constructing Print Document Content by Using GDI+
- Creating Reports by Using Crystal Reports

1. List the tasks involved in printing simple text file output in Visual Studio .NET.

   **Use a StreamReader object to read in the contents of the text file and then draw the contents of the StreamReader line-by-line onto the Graphics object inside your PrintDocument.PrintPage event handler. The Graphics object is supplied by the PrintPageEventArgs class, which is the event arguments parameter (e) for the PrintPage event handler.**

2. What property is used to capture user input from a dialog box?

   **The DialogResult property of the parent form is used to capture the action taken to close a dialog box. For example, DialogResult can be used to determine whether the PrintDialog was closed by clicking OK or by clicking Cancel.**

3. What is the role of the **Graphics** object in printing by using GDI+?

   **The Graphics object acts as the canvas on which the GDI+ methods draw.**

4. What methods of GDI+ are used to draw text and graphics?

   **Methods such as DrawRectangle and FillEllipse are used to draw 2-D vector shapes. The DrawString method is used to draw text.**

5. What kind of object does the following code create?

```
Pen myPen = new Pen(Color.Black);
Graphics g = this.CreateGraphics();
g.DrawEllipse(myPen, 20, 30, 10, 50);
```

   **This Visual C# .NET code produces the 2-D vector outline shape of an ellipse by using a solid black pen that has the default width of 1 pixel (the default graphic unit).**

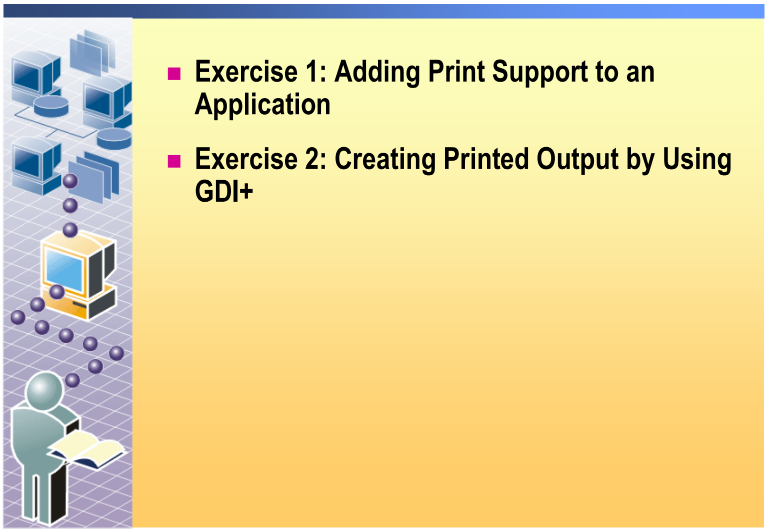
6. List some of the benefits of using Crystal Reports.

   **Crystal Reports provides a fast and easy way to generate professional looking reports.**

7. List the steps involved in creating a report by using Crystal Reports.

   **Establish the database connection to your data, add a Crystal Report (.rpt file) to your project, configure the report by using the Crystal Report Gallery, add a CrystalReportViewer control to your form, declare a report object and assign an instance of your report (.rpt) to it, use the SetDataSource property of your report object to populate it with data, and then assign the report object to the ReportSource property of your CrystalReportViewer.**

# Lab 6.1: Printing Formatted Documents



- Exercise 1: Adding Print Support to an Application
- Exercise 2: Creating Printed Output by Using GDI+

## Objectives

After completing this lab, you will have demonstrated your ability to:

- Reference a namespace so that you don't have to fully qualify each namespace member in your code.
- Use the **PrintDocument** class and the **PrintPage** event to enable printing in a Windows Forms application.
- Use the **PageSetupDialog**, **PrintPreviewDialog**, and **PrintDialog** classes to provide user control over the print process.
- Create pens, brushes, and fonts for use with GDI+ drawing methods.
- Use GDI+ methods to draw outline and filled shapes on a print document.
- Measure and position text on a print document.

**Note** This lab focuses on the concepts in Module 6, "Printing and Reporting in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. As a result, this lab may not comply with Microsoft security recommendations.

## Prerequisites

Before working on this lab, you must have:

- The knowledge and skills to develop a simple Windows Forms application by using a Visual Studio .NET–compatible programming language.
- The knowledge and skills to use elements of the **System.Drawing** namespace and the **PrintDocument** class in a Visual Studio .NET–based application.
- The knowledge and skills to use the **PrintPreviewDialog**, **PageSetupDialog**, and **PrintDialog** classes in a Visual Studio .NET–based application to provide user control of the print process.
- The knowledge and skills to use the two-dimensional (2-D) vector and Text drawing capabilities of GDI+ in a Visual Studio .NET–based application.

## Scenario

Members of the Northwind Traders sales force need to print purchase order documents while they are at a customer site. The purchase order documents—form number NT-2555P (portrait) and NT-2555L (landscape)—have specific requirements associated with document layout, text formatting, and the appearance of 2-D vector graphics objects. You are an application developer at Northwind Traders. The department that you work in is developing a purchase order application that will be used by the Northwind Traders sales force. You have been assigned the task of completing the code sections of the purchase order application that support printing. The Northwind Traders Legal and Media departments have given you the three requirements tables that describe these forms. The tables are included at the end of this lab.

To complete this task, you must add basic print support to a project, enable users to control the print process by using dialog boxes, and develop the code statements necessary to complete the construction of the print document by using GDI+.

## Estimated time to complete this lab: 45 minutes

# Exercise 1
# Adding Print Support to an Application

In this exercise, you will open two existing Visual Studio .NET projects, create a reference to the **Drawing** and **Drawing2D** namespaces, create a procedure that handles the **PrintPage** event, develop code that ensures that all pages of a print document are included in a print job, and add support for the **PrintPreviewDialog**, **PageSetupDialog**, and **PrintDialog** dialog boxes to your application. This exercise assesses your knowledge of the print process and your ability to use the **PrintDocument** class and the three dialog boxes to provide the application user with control of the print process.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab06_1\Ex01\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab06_1\Ex01\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

## Scenario

The user interface of the purchase order application has been developed and the underlying code is functioning as intended. You have reviewed the application code and the code contained in a custom document printing class that inherits from the **PrintDocument** class. The custom printing class that your department created will be used to print the purchase order forms that are used by Northwind Traders. Most of the code that is used to construct (draw) purchase order documents has already been written. You will now begin adding code to your application and the printing class so that users can print purchase order documents.

The Northwind Traders sales force has requested that the application be capable of printing in either portrait or landscape mode and that page margins be fixed in accordance with the Legal department and Media department requirements for the purchase order document. The sales force would also like to have the option to review the purchase order document with a customer before it is printed.

As the application developer, you will start by opening the purchase order application and the printing class in separate projects and checking to see what code still needs to be added to the two projects to support document printing. Then, you will add support for the **PrintPreviewDialog**. In addition to providing the sales force with an easy way to preview a purchase order before printing, the **PrintPreview** dialog box enables you to view the print document as you develop the code that constructs the document. You will then add support for the **PageSetupDialog** class to your application and create the code that is required to show this dialog box. By checking various page layout settings, you can ensure that the print document is constructed in accordance with the page layout parameters needed by your Legal and Media departments. Because your customer also wants the option to print without previewing the document, you will also add support for the **PrintDialog** class.

| Tasks | Additional information |
|---|---|
| **1.** Open two instances of Visual Studio .NET. In the first instance, open the Lab06Application.sln file. In the second instance, open the Lab06Class.sln file. To open the solution files, in the Lab06Application and Lab06Class folders, respectively, browse to *install_folder*\Labfiles\Lab06_1\Ex01\Starter.<br>You can open two instances of Visual Studio .NET when you want to work on an application and an external class library at the same time. | **a.** For more information about opening a project file and starting an application, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. For additional information about starting an application from in Designer, in Index, search by using the phrase **Debugging Windows Applications**. |
| **2.** Use the Task List in the Lab06Class.cs file to locate the code section 'TODO: Programmatically reference required namespaces' and then add code statements that reference the **Drawing** and **Drawing2D** namespaces.<br>When you create a reference to a namespace, you can refer to the namespace members without having to fully qualify member names in your code. This makes your code easier to develop and easier to read. | **a.** For more information about the **Drawing** and **Drawing2D** namespaces and why you should use them, see the following resources:<br>• Lesson: Constructing Print Document Content Using GDI+ in Module 6, "Printing and Reporting in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. For additional information about creating a reference to a namespace, search by using the phrases **References and the Imports Statement**, **Namespaces** and **Imports Statement**. For additional information about the **Drawing** and **Drawing2D** namespaces, search by using the phrases **System.Drawing** and **System.Drawing.Drawing2D**. |

| Tasks | Additional information |
|---|---|
| **3.** Use the Task List in the Lab06Class.cs file to locate the code section 'TODO: Create the declaration statement for the PrintPage Procedure' and then create the declaration statement for a procedure named **Lab06Class_PrintPage** that handles the **PrintPage** event. Add code statements to the procedure that call the **PrintingEmptyForm2555** and **PrintingContentsForm2555** procedures.<br>The **PrintPage** procedure handles all requests for a page of the print document. The **PrintPageEventArgs** class that is passed to the PrintPage procedure contains the **Graphics** object on which you construct the print document. | **a.** For more information about the **PrintPage** event and the **PrintPageEventArgs** class, see the following resources:<br>• Lesson: Printing From a Windows Forms Application in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Adding Print Support to a Windows Forms Application in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. For additional information about the **PrintPage** event, search by using the phrases **Creating Standard Windows Forms Print Jobs** and **PrintDocument.PrintPage Event**. For additional information about **the PrintPageEventArgs** class, search by using the phrases **PrintPageEventArgs Class** and **PrintPageEventArgs Members**. |
| **4.** Use the Task List in the Lab06Class.cs file to locate the code section 'TODO: Determine if more pages must be printed' and then create a code section that tells the event handler for the **PrintPage** event that there are more pages to print when **currentPurchaseItemNumber** is less than **totalPurchaseItems**, otherwise specify that there are no more pages to print.<br>In Visual Studio .NET, you must create your own programming logic to set the **HasMorePages** property to generate print documents that consist of more than one page. | **a.** For more information about how to specify that there are additional pages to be printed, see the following resources:<br>• Lesson: Printing From a Windows Forms Application in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Adding Print Support to a Windows Forms Application in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **PrintPageEventArgs.HasMorePages Property**. |

| Tasks | Additional information |
|---|---|
| **5.** Rebuild the Lab06Class project. In the Lab06Application project, in Solution Explorer, update the reference to Lab06Class.<br>After you rebuild a class library, ensure that your applications reference the new version. | **a.** For more information about referencing an external class, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Project References**, **Adding and Removing References**, and **Preparing and Managing Builds**. |
| **6.** Open the code editor view of MainForm.cs in the Lab06Application project, and configure the Task List to display comments.<br>Using the Task List and TODO comments can help you remember development tasks that must still be done. | **a.** For more information about the code editor view and the Task List, see the following resource:<br>• The Visual Studio .NET Help documentation. For help with the code editor, search by using the phrase **Managing the Code Editor and View**. For help with Task List, search by using the phrase **Task List Views**. |
| **7.** Use the Task List to locate the code section 'TODO: Create an instance of the PrintPreviewDialog class'. Add code below the comment line that creates an instance of the **PrintPreviewDialog** class named **form2555PreviewDialog**. | **a.** For more information about the **PrintPreviewDialog** class and how to add an instance of this dialog box to your application, see the following resources:<br>• Lesson: Using the Print Preview, Page Setup, and Print Dialogs in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*<br>• Practice: Using the PrintPreviewDialog Control in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Introduction to the Windows Forms PrintPreviewDialog Control**. |

| Tasks | Additional information |
|---|---|
| **8.** Use the Task List to locate the code section 'TODO: Create an instance of the **PageSetupDialog** class'. Add code below the comment line that creates an instance of the **PageSetupDialog** class named **form2555SetupDialog**.<br>The **PageSetupDialog** class enables users to modify the page settings of a print document. You can use it to test the code that constructs the print document by displaying the print document with various page setting values. You will get a chance to modify page settings later in this lab exercise. | **a.** For more information about adding the **PageSetupDialog** class to an application, see the following resources:<br>• Lesson: Using the Print Preview, Page Setup, and Print Dialogs in Module 6, "Printing and Reporting in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Using the PageSetupDialog Control in Module 6, "Printing and Reporting in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrases Introduction to the Windows Forms PageSetupDialog Component, PageSetupDialog Class, and PageSetupDialog Members. |
| **9.** Use the Task List to locate the code section 'TODO: Create an instance of the PrintDialog class'. Add code below the comment line that creates an instance of the **PrintDialog** class named **form2555PrintDialog**. | **a.** For more information about adding support for the **PrintDialog** class to your application, see the following resources:<br>• Lesson: Using the Print Preview, Page Setup, and Print Dialogs in Module 6, "Printing and Reporting in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Using the PageSetupDialog Control in Module 6, "Printing and Reporting in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Introduction to the Windows Forms PrintDialog Component** and **PrintDialog Component**. |

| Tasks | Additional information |
|---|---|
| **10.** Use the Task List to locate the code section 'TODO: Set purchaseItemNumber and purchaseItemCount'. Add code below the TODO comment that assigns a value of **0** to the *MainModule.purchaseItemNumber* variable and assigns the value of the **Count** property of MainModule.mainPOForm. ProductOrderPanel.Controls to the *MainModule***.***purchaseItemCount* variable.<br>In addition to developing the code logic that determines when **HasMorePages** is **True** or **False**, you must reset the variables that are used to make this determination every time a print document is generated. | **a.** The *purchaseItemNumber* and *purchaseItemCount* variables are passed to the Lab06Class class (inherits from PrintDocument) and used to determine when HasMorePages should be set to **True** or **False**. The *purchaseItemNumber* variable is the current purchase item being printed, and the *purchaseItemCount* variable is the total number of purchase items in the current purchase order. |
| **11.** Use the Task List to locate the code section 'TODO: Add support for a full screen preview of the print document'. Add code below the comment line that assigns the **form2555Document** instance of the Lab06Class class to the **Document** property of **form2555PreviewDialog** (the print preview dialog box) and displays the print document by using the full display screen.<br>Implementing support for the print preview dialog box early in the development process enables you to preview the print document as you develop the code that constructs your document. | **a.** For more information about the **PrintPreviewDialog** class and how it can be used in the print process, see the following resources:<br>• Lesson: Using the Print Preview, Page Setup, and Print Dialogs in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Using the PrintPreviewDialog Control in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Displaying Print Preview in Windows Applications**. |

| Tasks | Additional information |
|---|---|
| **12.** Rebuild and then start the Lab06Application project. Use the **File** menu or the **Print** button on the ToolBar control to demonstrate that you can now display the print preview dialog box. Close the Lab06Application executable file.<br>You can use the **Print Preview** dialog box to verify that the code in your **PrintPage** procedure is working correctly. Try adding 25 purchase items to a purchase order. The print document should now wrap onto a second page when previewed. | **a.** For more information about building and debugging your applications, and for information about running the Purchase Order application, see the following resources:<br>• Demonstration, Purchase Order Application, in Module 0, “Introduction,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Default and Custom Builds** and **Using the Debugger**. |
| **13.** Use the Task List to locate the code section 'TODO: Apply the initial page settings'. Add code below the comment line that assigns the *form2555PageSettings* variable to the **PageSettings** property of the **Page Setup** dialog box that you instantiated.<br>The page setup dialog box must be assigned an initial **PageSettings** value before it can be displayed for the first time; otherwise, an error will occur. | **a.** The **form2555PageSettings** object is scoped at the module level so that user-specified settings are maintained between viewings of the **PageSetupDialog** class.<br>**b.** For more information about the **PageSettings** class, see the following resources:<br>• Lesson: Using the Print Preview, Page Setup, and Print Dialogs in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Using the PageSetupDialog Control in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **PageSettings Class**. |

| Tasks | Additional information |
|---|---|
| **14.** Use Task List to locate the code section 'TODO: Disable user access to the Margins section of the dialog'. Add code below the comment line that disables user access to the margin settings.<br>Restricting user access to only those page settings that are required will help to eliminate support issues that would be generated when users modify settings in an unpredictable manner. | **a.** For more information about members of the **PageSetupDialog** class, see the following resources:<br>• Lesson: Using the Print Preview, Page Setup, and Print Dialogs in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Using the PageSetupDialog Control in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **PageSetupDialog Members**. |
| **15.** Rebuild and then start the Lab06Application project. Use the **File** menu to open the page setup dialog box and change the page orientation from portrait to landscape. Use the print preview dialog box to view the print document. Close the Lab06Application executable file. | **a.** For more information about building and debugging your application, use the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Default and Custom Builds** and **Using the Debugger**. |
| **16.** Use the Task List to locate the code section 'TODO: Determine if the document should be printed'. Add code statements below the comment line that will send **form2555Document** to the printer when the application user clicks **OK** to close the print dialog.<br>You must use **DialogResult** to determine whether the user closed **PrintDialog** by clicking **Cancel** or by clicking **OK**. | **a.** For more information about using **DialogResult**, see the following resources:<br>• Lesson: Using the Print Preview, Page Setup, and Print Dialogs in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: Using the PageSetupDialog Control in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Retrieving the Result for Dialog Boxes**. |

| Tasks | Additional information |
|---|---|
| **17.** Rebuild and then start the Lab06Application project. Use the **File** menu to demonstrate that the print dialog box is working as intended. Close the Lab06Application executable file. | **a.** For more information about building and debugging your application, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Default and Custom Builds** and **Using the Debugger**. |
| **18.** Save the changes that you made to your code, and then close both solution files. | Additional information is not necessary for this task. |

# Exercise 2
# Creating Printed Output by Using GDI+

In this exercise, you will create pens, brushes, and fonts and then use them to draw 2-D vector objects and text at specific locations on a print document. This exercise provides you with an opportunity to assess your ability to complete the construction of a print document by using GDI+.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab06_1\Ex02\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab06_1\Ex02\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

## Scenario

Initial testing has revealed that the custom printing class does not accurately reproduce the legal and official representation of the organization's purchase order documents—Northwind Traders forms NT-2555P and NT-2555L. To correct this problem, you will review the data contained in the NT-2555P and NT-2555L requirements tables (the tables of data from the Media and Legal Departments that are included after this exercise) and develop GDI+ code statements that produce the required 2-D vector objects and text. You will begin by creating some of the pens, brushes, and fonts required to construct portions of the forms. After that, you will develop the code that draws some of the 2D vector shapes that are used to create the base forms. The last step will be to develop the code used to draw some of the text that appears on the NT-2555P and NT-2555L forms.

| Tasks | Additional information |
|---|---|
| **1.** In the first instance of Visual Studio .NET, browse to *install_folder*\Labfiles\Lab06_1\Ex02\Starter and open the Lab06Application.sln file. In the second instance, browse to *install_folder*\Labfiles\Lab06_1\Ex02\Starter and open the Lab06Class.sln file. | **a.** For more information about opening a project file and starting an application, see the following resources:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. For additional information about starting an application from in Designer, in Index, search by using the phrase **Debugging Windows Applications**. |
| **2.** Open the code editor view of the Lab06Class.cs file and use Task List to locate the code section 'TODO: Create the pageBorderPen'. Add code below the comment line that creates a pen named *pageBorderPen* and that has the following characteristics:<br>Color = Gray,<br>width = 3,<br>DashStyle = Dash. | **a.** For more information about creating pens, see the following resources:<br>• Lesson: Constructing Print Document Content Using GDI+ in Module 6, "Printing and Reporting in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: How to Construct Print Document Content Using GDI+ in Module 6, "Printing and Reporting in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Pens, Brushes, and Colors**; **Setting Pen Width and Alignment** (in the .NET Framework Developer's Guide) and **Using a Brush to Fill Shapes**. |

| Tasks | Additional information |
|---|---|
| **3.** Use the Task List to locate the code section 'TODO: Create the footerBrush'. Add code below the comment line that creates a solid black brush named footerBrush. | **a.** For more information about creating brushes, see the following resources:<br>• Lesson: Constructing Print Document Content Using GDI+ in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: How to Construct Print Document Content Using GDI+ in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrases Pens, Brushes, and Colors; Setting Pen Width and Alignment and Using a Brush to Fill Shapes. |
| **4.** Use the Task List to locate the code section 'TODO: Create the footerFont'. Add code below the comment line that creates a **Private** font named footerFont and that has the following characteristics: family = Microsoft Sans Serif, emSize = 7. | **a.** For more information about creating fonts, see the following resources:<br>• Lesson: Constructing Print Document Content Using GDI+ in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: How to Construct Print Document Content Using GDI+ in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Constructing Font Families and Fonts** (in the .NET Framework Developer’s Guide), **Fonts and Text** (in the .NET Framework Developer’s Guide), and **Using GDI+ Managed Classes** (in the .NET Framework Developer’s Guide). |

| Tasks | Additional information |
|---|---|
| **5.** Use the Task List to locate the code section 'TODO: Draw the border of the Customer Address table'. Create the code statements required to draw the border of the customer address table by using the following variables: *sectionOutlinePen*, *customerSectionLeft*, *customerSectionTop*, *customerSectionWidth*, *customerSectionHeight*.<br>GDI+ offers many ways to draw an outlined shape such as a rectangle. As a developer, it is important to consider whether the parameters used to draw a shape can be used again later in your program before you select a drawing method. | **a.** For more information about drawing 2-D vector shapes, see the following resources:<br>• Lesson: Constructing Print Document Content Using GDI+ in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: How to Construct Print Document Content Using GDI+ in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Pens, Lines, and Rectangles**; **Using a Pen to Draw Lines and Rectangles**; and **Graphics Methods**. |
| **6.** Use the Task List to locate the code section TODO: Draw a rectangle that defines the margin limits'. Create the code statements required to draw a rectangle that defines the limits of the page by using the following variables: *pageBorderPen, pageLeft, pageTop, pageWidth, pageHeight*. | **a.** For more information about drawing 2-D vector shapes, see the following resources:<br>• Lesson: Constructing Print Document Content Using GDI+ in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: How to Construct Print Document Content Using GDI+ in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Pens, Lines, and Rectangles**; **Using a Pen to Draw Lines and Rectangles**; and **Graphics Methods**. |

| Tasks | Additional information |
|---|---|
| **7.** Use the Task List to locate the code section 'TODO: Draw the shaded area of the Purchase Items table'. Create the code statements required to draw the shaded portion of the Purchase Items table by using the following variables: *purchaseSectionHeaderRowBackgroundBrush*, *purchaseSectionLeft*, *purchaseSectionTop*, *purchaseSectionWidth*, *purchaseSectionRowHeight*.<br>When you use filled shapes on a print document, consider how the document will look when it is photocopied or printed by using a gray scale. | **a.** For more information about filled shapes, see the following resources:<br>• Lesson: Constructing Print Document Content Using GDI+ in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: How to Construct Print Document Content Using GDI+ in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. For additional information about drawing 2-D filled shapes, search by using the phrases **Drawing Lines and Shapes with GDI+** and **GDI+ Graphics**. |
| **8.** Use Task List to locate the code section 'TODO: Draw the horizontal lines inside the Purchase Items table'. Create the code statements required to draw the interior lines by using the following variables: *sectionInteriorPen*, *purchaseSectionLeft*, *verticalPosition*, *purchaseSectionRight*, *verticalPosition*.<br>In addition to using the width and color properties to change the appearance of a line, you should consider using line styles and line caps. | **a.** For more information about drawing lines, see the following resources:<br>• Lesson: Constructing Print Document Content Using GDI+ in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: How to Construct Print Document Content Using GDI+ in Module 6, “Printing and Reporting in Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrases Using a Pen to Draw Lines and Rectangles, Drawing a Line with Line Caps, and Using a Pen to Draw Lines and Shapes. |

| Tasks | Additional information |
|---|---|
| **9.** Use the Task List to locate the code section 'TODO: Draw the footer text'. Create the code statements required to draw the contents of the *printText* variable by using the variables *footerFont*, *footerBrush*, *horizontalPosition*, and *verticalPosition*, where *horizontalPosition* has a value equal to the left margin value of the page and *verticalPosition* has a value equal to the bottom margin value of the page minus the height of the font.<br>By default, text is drawn below and to the right of the position specified in a **DrawString** command. | **a.** For more information about drawing and measuring text, see the following resources:<br>• Lesson: Constructing Print Document Content Using GDI+ in Module 6, "Printing and Reporting in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Practice: How to Construct Print Document Content Using GDI+ in Module 6, "Printing and Reporting in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Drawing Text with GDI+** and **Fonts and Text**. |
| **10.** Rebuild the Lab06Class project, reconstruct the reference to the class library in the Lab06Application project, and then view the updated print document by using Lab06Application. | **a.** For more information about building and debugging your application, use the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrases **Default and Custom Builds** and **Using the Debugger**. |
| **11.** Save your projects, and then close Visual Studio .NET. | Additional information is not necessary for this task. |

The following tables provide information required to construct the NT-2555P and NT-2555L forms, which are the official Northwind Traders purchase order documents.

| **Page Layout Table for Forms NT-2555P and NT-2555L** | | |
|---|---|---|
| **Document region/feature** | **Size constraint, NT-2555P (portrait)** | **Size constraint, NT-2555L (landscape)** |
| Top Margin | 1.0 inch (25.4 mm) | 0.85 inch (21.6 mm) |
| Left Margin | 1.2 inch (30.5 mm) | 1.6 inch (40.6 mm) |
| Right Margin | 0.8 inch (20.3 mm) | 1.0 inch (25.4 mm) |
| Bottom Margin | 1.0 inch (25.4 mm) | 0.7 inch (17.8 mm) |
| Document Title Area | 10% of available height*, full width | Same as NT-2555P |
| Address Table | 25% of available height*, full width | Same as NT-2555P |
| Purchase Table | 62.5% of available height*, full width | Same as NT-2555P |
| This table is incomplete. The complete **Page Layout Table for Forms NT-2555P and NT-2555L** table would contain additional information required to recreate the NT-2555P and NT-2555L forms. Information has been left out of this table because the additional information is not required to complete the exercises in this lab. | | |
| * The available height is the distance between the top and bottom margins. | | |

| **Vector Objects Table for Forms NT-2555P and NT-2555L** | | |
|---|---|---|
| **Document region/feature** | **Object type** | **Pen/brush details** |
| Page Border | Rectangle | Dashed line type, 3 point (pt) width, gray |
| Address Table, border lines | Rectangle | Solid line type, 2 pt width, blue |
| Address Table, interior lines | Line | Solid line type, 1 pt width, blue |
| Address Table, 'Ship To' rows | Filled Rectangle | Light-gray brush fill |
| Purchase Table, border lines | Rectangle | Solid line type, 2 pt width, blue |
| Purchase Table, interior lines | Line | Solid line type, 1 pt width, blue |
| Order Item Table, header row | Filled Rectangle | Light-blue brush fill |

**Font Specification Table for Forms NT-2555P and NT-2555L**

| **Document region/feature** | **Font details (font name, style, point size, and color)** | **Text layout (position in region)** |
|---|---|---|
| Company Name | Microsoft Sans Serif, Bold, 16 pt, Blue | Center-left position in Document Title region |
| Company Address | Microsoft Sans Serif, Regular, 8 pt, Blue | Immediately to the right of, and bottom-aligned to Company Name |
| Address Table Labels | Microsoft Sans Serif, Regular, 7 pt, Gray | Offset 10% of the font height from the top-left corner of the cell |
| Address Table Text | Microsoft Sans Serif, Regular, 10 pt, Black | Offset 25% of the font height from the left side of the cell and down 110% of the font height from the top of the cell |
| Order Item Table Header | Microsoft Sans Serif, Regular, 10 pt, Black | Centered in the cell |
| Order Item Table Text: Quantity, Unit Price, Discount | Microsoft Sans Serif, Regular, 9 pt, Black | Right justified in the cell with an offset of 25% of the font height |
| Order Item Table Text: Product Description, Unit Size | Microsoft Sans Serif, Regular, 9 pt, Black | Left justified in the cell with an offset of 25% of the font height |

msdn training

# Module 7: Asynchronous Programming

**Contents**

Microsoft

# Instructor Notes

**Presentation:**
**90 minutes**

**Lab:**
**15 minutes**

In this module, students learn how to use the techniques of asynchronous programming and multithreading to avoid blocking the user interface of an application.

After completing this module, students will be able to:

- Describe the Microsoft® .NET Framework asynchronous programming model.
- Modify a client application to use built-in .NET Framework support for asynchronous calls to methods.
- Describe how to add explicit support for asynchronous calls to any method.

**Required materials**

To teach this module, you need the Microsoft PowerPoint® file 2555A_07.ppt.

**Preparation tasks**

To prepare for this module:

- Read all of the materials for this module.
- Review the animation for this module.
- Complete the demonstrations, practice, and lab.

# How to Teach This Module

This section contains information that will help you to teach this module.

- If students are interested in referencing code examples in other languages, point them to "Language Equivalents" in Microsoft Visual Studio® .NET Help documentation. This section provides examples in languages such as Microsoft Visual Basic® .NET, C#, and Java.
- The lab at the end of this module is based on the Expense Report application in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C#™ .NET)*, and is intended to simulate a real-world environment in which students will demonstrate what they learned during the lecture and practice portions of the module. The lab does not provide step-by-step detailed instructions; instead, the students are given tasks to complete in the left column and a list of resources that they can use (if they need help) in the right column. Students get hands-on experience that they need by completing the practice activity in the module.

# Lesson: The .NET Asynchronous Programming Model

This section describes the instructional methods for teaching this lesson.

This is a brief overview lesson to define asynchronous programming and provide scenarios in which it is appropriate to use asynchronous programming. Wait to discuss implementation until you teach the next lesson.

# Lesson: The Asynchronous Programming Model Design Pattern

This section describes the instructional methods for teaching this lesson.

Emphasize to students that when they use .NET Framework classes that include support for asynchronous programming they do not have to implement the Begin*Operation* and End*Operation* methods. These are provided by the .NET Framework, and all they, as developers, must do is to call these methods.

### Overview of the Asynchronous Programming Model Design Pattern

Focus on the asynchronous callback technique for completion, because this is the technique that is used in the rest of the lesson and is the most typical technique to use for Windows Forms applications.

# Lesson: How to Make Asynchronous Calls to Any Method

This section describes the instructional methods for teaching this lesson.

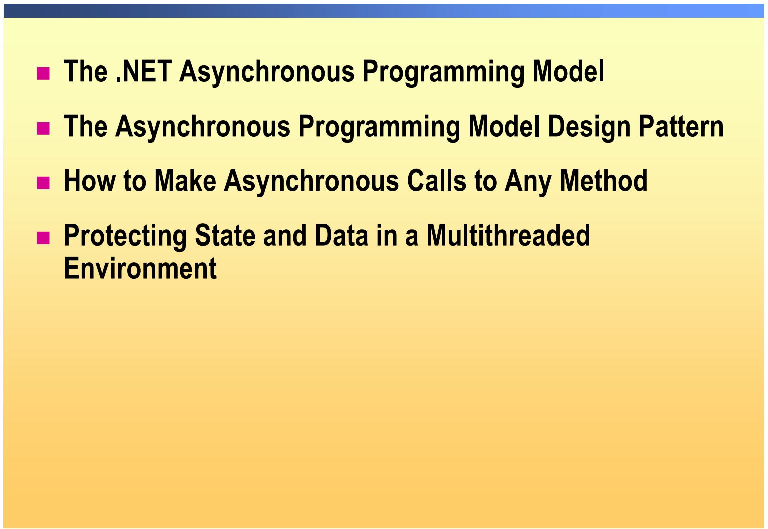
Point out to students that even when they create and use a delegate to call any method, the .NET Framework still provides the implementation of the **BeginInvoke** and **EndInvoke** methods.

# Lesson: Protecting State and Data in a Multithreaded Environment

This section describes the instructional methods for teaching this lesson.

Tell students that especially for Windows Forms application, they should be able to design the application to minimize the need for synchronization. They can accomplish this by selectively enabling and disabling user interface elements as asynchronous operations are performed.

# Overview



- The .NET Asynchronous Programming Model
- The Asynchronous Programming Model Design Pattern
- How to Make Asynchronous Calls to Any Method
- Protecting State and Data in a Multithreaded Environment

**Introduction**

A well-designed Microsoft® .NET Framework Windows Forms application includes techniques to prevent blocking user interaction.

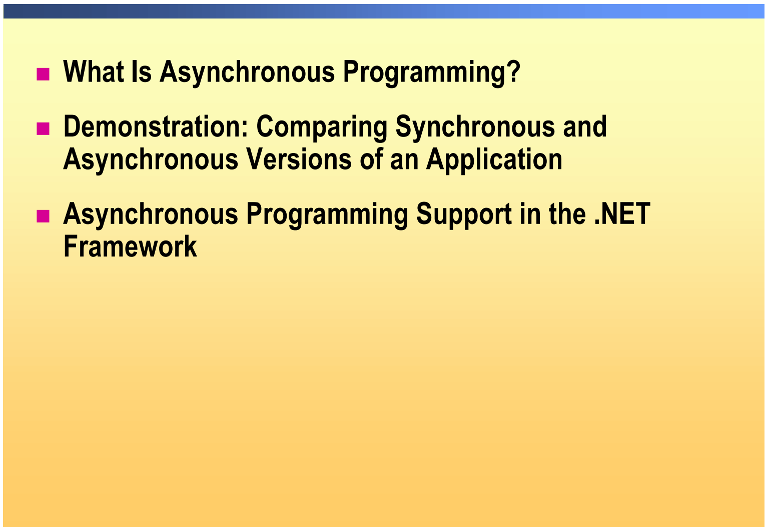
In this module, you will learn how to use the techniques of asynchronous programming and multithreading to avoid blocking the user interface of an application.

**Objectives**

After completing this module, you will be able to:

- Describe the .NET Framework asynchronous programming model.
- Modify a client application to use built-in .NET Framework support for asynchronous calls to methods.
- Describe how to add explicit support for asynchronous calls to any method.

# Lesson: The .NET Asynchronous Programming Model



- What Is Asynchronous Programming?
- Demonstration: Comparing Synchronous and Asynchronous Versions of an Application
- Asynchronous Programming Support in the .NET Framework

**Introduction**

The .NET Framework provides extensive support for asynchronous calls and multithreading.

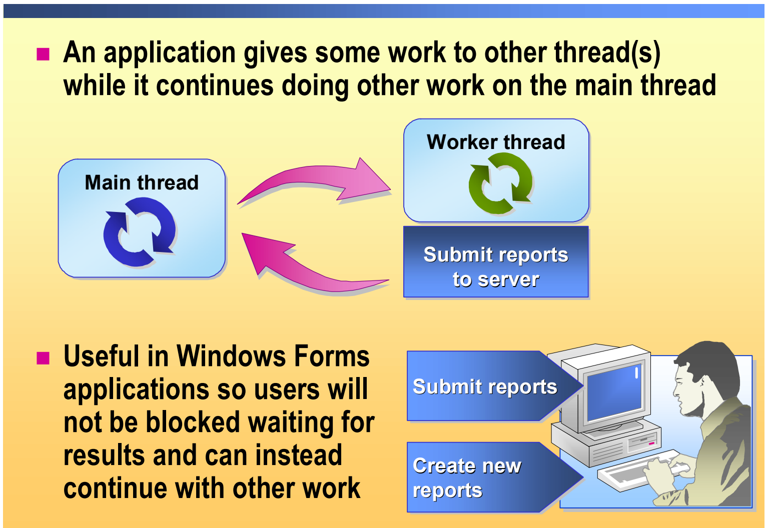
In this lesson, you will learn about asynchronous programming and. the NET asynchronous programming model.

**Lesson objectives**

After completing this lesson, you will be able to:

- Define asynchronous programming.
- List typical scenarios in which asynchronous programming is useful.
- Describe the .NET asynchronous programming model.

# What Is Asynchronous Programming?



- An application gives some work to other thread(s) while it continues doing other work on the main thread

Main thread

Worker thread

Submit reports to server

- Useful in Windows Forms applications so users will not be blocked waiting for results and can instead continue with other work

Submit reports

Create new reports

**Introduction**

One important measure of a Windows Forms application is its timely response to user interaction. One important technique developers have to support responsiveness is asynchronous programming.

**Definition**

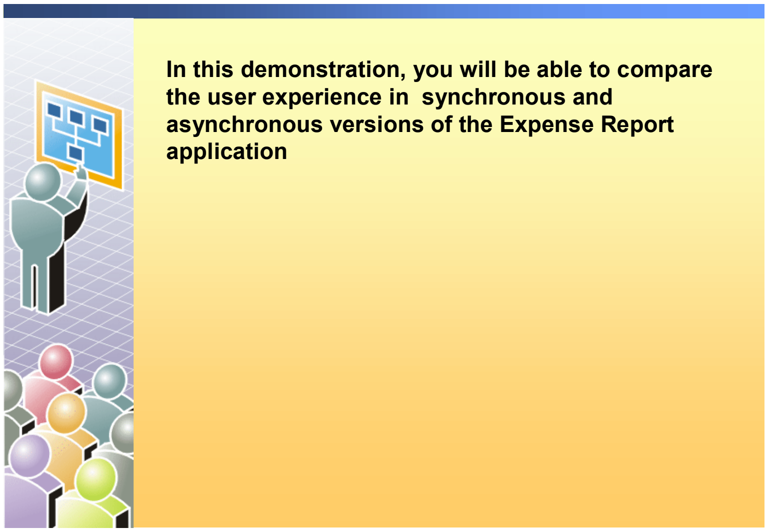
The central idea behind asynchronous programming is to be able to issue method calls to other components and to carry on with other work, without waiting for the operation to complete. The .NET Framework common language runtime provides rich support for asynchronous programming and handles the details of threading and data exchange.

**Example of asynchronous scenarios**

In a non-programming scenario, consider a painting crew that is painting the rooms in a house that is some distance from town. As they are painting walls of the rooms, the crew chief realizes that they did not bring enough trim paint for the woodwork. The crew chief can tell one of the crew members to go to the paint store to order and wait for more trim paint to be custom mixed to the appropriate color, while the rest of the crew continues to paint the walls. When the crew member finally returns with the trim paint, some members of the crew can switch to painting the woodwork in those rooms where the walls have dried.

There are quite a few scenarios in the programming world that involve potential wait times: requests for file or network input/output (I/O), Web access, and so forth. You can use asynchronous techniques in your applications in these scenarios to allow users to remain productive.

# Demonstration: Comparing Synchronous and Asynchronous Versions of an Application



**In this demonstration, you will be able to compare the user experience in synchronous and asynchronous versions of the Expense Report application**

### Introduction

In this demonstration, you see a comparison of the user experience in synchronous and asynchronous versions of the Expense Report application.

### Instructions

► **To demonstrate the synchronous behavior of the Expense Report application**

1. Run the synchronous version of the Expense Report application, ExpenseReport.exe, in *install_folder*\Democode\Mod07\Mod07_01\Sync.

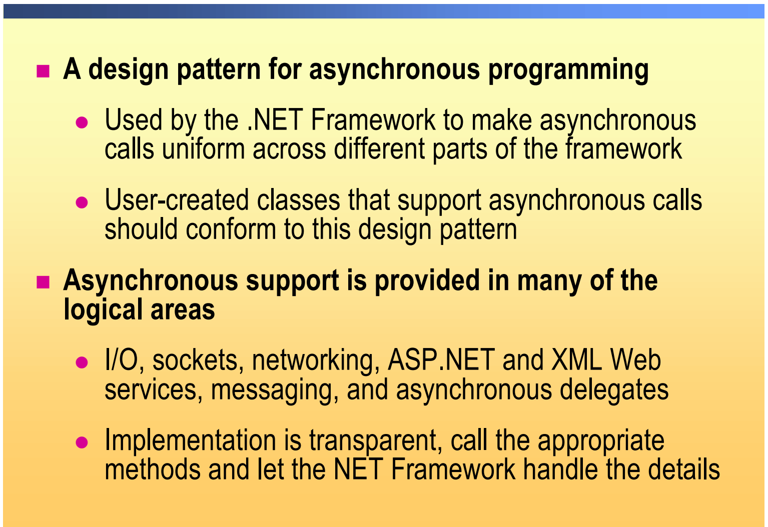
**Note** If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

2. In the main application window, click the **View Submitted Reports** button.
3. Notice that the application is not responsive while the XML Web service is being called. You can test this by trying to move the main application window, or by trying to minimize the main application window before the expense report list window opens.
4. After the expense report list window opens, close the application.

### ► To demonstrate the asynchronous behavior of the Expense Report application

1. Run the asynchronous version of the Expense Report application, ExpenseReport.exe, in *install_folder*\Democode\Mod07\Mod07_01\Async.
2. On the main application window, click the **View Submitted Reports** button.
3. Notice that the application is responsive while the XML Web service is being called. You can test this by moving the splitter control in the expense report list window, or by minimizing the main application window.
4. After data is loaded in the expense report list window, close the application.

# Asynchronous Programming Support in the .NET Framework



- **A design pattern for asynchronous programming**
  - Used by the .NET Framework to make asynchronous calls uniform across different parts of the framework
  - User-created classes that support asynchronous calls should conform to this design pattern
- **Asynchronous support is provided in many of the logical areas**
  - I/O, sockets, networking, ASP.NET and XML Web services, messaging, and asynchronous delegates
  - Implementation is transparent, call the appropriate methods and let the NET Framework handle the details

### Introduction

The .NET Framework provides a design pattern for asynchronous programming and built-in support for many areas which often involve making complex calls that take a considerable amount of time to complete.

### Support provided by the .NET Framework

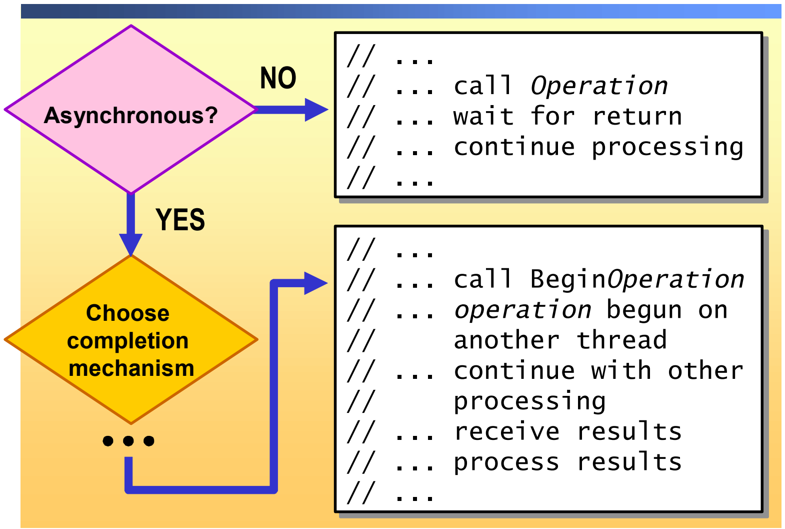
The .NET Framework provides a design pattern that makes asynchronous calls uniform across the different parts of the .NET Framework.

Classes that include built-in support have a pair of asynchronous methods for each synchronous method they contain. For example, if you want to read from a stream synchronously, you use the **Read** method of the **System.IO.Stream** class. If, however, you want to make the call asynchronously, then you would instead use the **BeginRead** and **EndRead** methods of the **System.IO.Stream** class.

In addition, the .NET Framework makes the details of threading that support asynchronous programming mechanisms largely transparent to the developer.

For more information about using the support provided for asynchronous programming by the .NET Framework in your applications, see the lesson The Asynchronous Programming Model Design Pattern in this module.

# Lesson: The Asynchronous Programming Model Design Pattern



```
// ...
// ... call Operation
// ... wait for return
// ... continue processing
// ...
```

```
// ...
// ... call BeginOperation
// ... operation begun on
//     another thread
// ... continue with other
//     processing
// ... receive results
// ... process results
// ...
```

### Introduction

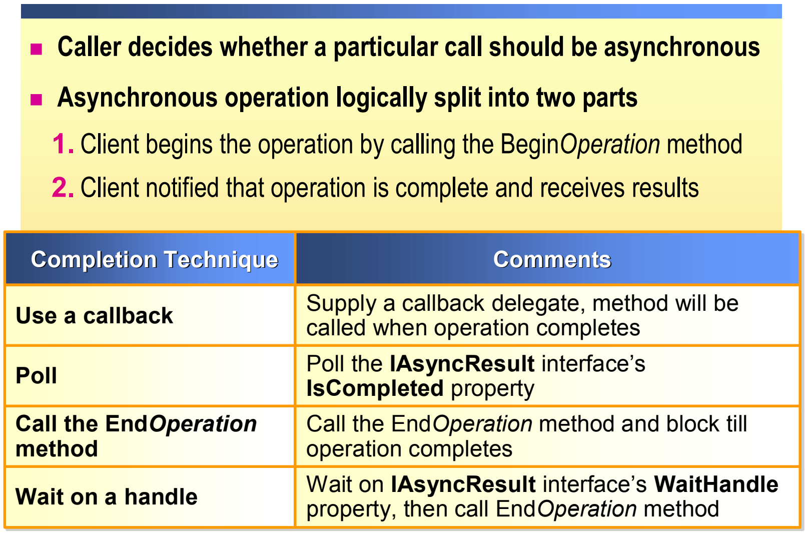
The asynchronous programming model design pattern provides a consistent and type-safe way to call methods asynchronously. This lesson concentrates on how to make asynchronous calls to classes that contain built-in asynchronous support. In the lesson How to Make Asynchronous Calls to Any Method, you will learn about the additional steps you must take to make asynchronous calls to classes that do not contain built-in asynchronous support.

### Lesson objectives

After completing this lesson, you will be able to:

- Describe the asynchronous programming model design pattern.
- List the four ways to be notified of completion and describe appropriate scenarios for each way.
- List the steps for using asynchronous callback as the notification method.
- Modify a client application to use built-in .NET Framework support for asynchronous calls to methods.

# Overview of the Asynchronous Programming Model Design Pattern



- **Caller decides whether a particular call should be asynchronous**
- **Asynchronous operation logically split into two parts**
  1. Client begins the operation by calling the Begin*Operation* method
  2. Client notified that operation is complete and receives results

| Completion Technique | Comments |
|---|---|
| **Use a callback** | Supply a callback delegate, method will be called when operation completes |
| **Poll** | Poll the **IAsyncResult** interface's **IsCompleted** property |
| **Call the End*Operation* method** | Call the End*Operation* method and block till operation completes |
| **Wait on a handle** | Wait on **IAsyncResult** interface's **WaitHandle** property, then call End*Operation* method |

### Introduction

One of the innovations provided by the asynchronous design pattern is that the caller can decide whether a particular call should be asynchronous.

### Beginning the operation

After you determine that you want to make an asynchronous call to one of the classes that provides built-in asynchronous support, you simply make a call to the class's appropriate Begin*Operation* method and determine what method you want to use to complete the call. The call to Begin*Operation* returns an object implementing this **IAsyncResult** interface. Depending on the completion technique used, this object may be used to help complete or get information about the asynchronous operation.

### Receiving completion notification and results

You must also determine how you want your application to be notified of the completion of the asynchronous call and to receive the results (if any) of the call.

### Callback

When you want to avoid blocking the thread that made the asynchronous call, use a callback method to be notified of the asynchronous call completion.

This is the technique most appropriate when you want to avoid blocking user interaction, for example in Windows Forms applications, or in other heavily event-driven processing scenarios. For this reason the remainder of this lesson will focus on using this technique.

When you use a callback, you typically call the End*Operation* method, described later in this topic, inside the callback method to retrieve the results of the asynchronous operation.

### Polling

If you want to maintain complete control over the flow of the application, then you can use a polling method. When using this technique, you will incur the overhead of manually checking the returned **IAsyncResult** interface's **IsCompleted** property to determine if the call is completed.

This technique is typically not appropriate for event-driven or user interaction scenarios.

### End*Operation*

The End*Operation* technique can be used when you have a finite amount of work to be done on the main thread that is not dependent on the asynchronous call. After that work is done and your application must receive the results of the asynchronous operation before it can do any further meaningful work, you can call the End*Operation* method to block until the asynchronous operation completes.

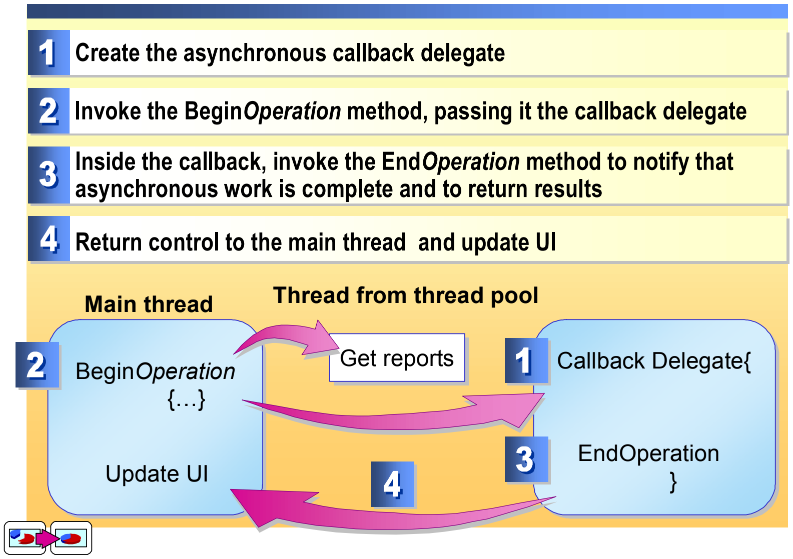
Be aware that an End*Operation* might block forever if the asynchronous call never completes due to a network or server failure. Therefore, the *wait on a handle* technique might be more appropriate to use in certain situations.

### Wait on a handle

With the *wait on a handle* technique, you can specify a maximum time-out to wait for an asynchronous operation to complete. You can also use the **System.Threading.WaitHandle.WaitAll** method to wait on multiple asynchronous operations and be notified when they have all completed.

For more information about using the polling and wait on a handle techniques, see Module 14, “Threading and Asynchronous Programming” in Course 2349B, *Programming with the Microsoft .NET Framework (Microsoft Visual C# .NET)*.

# Using the Design Pattern with an Asynchronous Callback for Completion



1 Create the asynchronous callback delegate

2 Invoke the Begin*Operation* method, passing it the callback delegate

3 Inside the callback, invoke the End*Operation* method to notify that asynchronous work is complete and to return results

4 Return control to the main thread  and update UI

**Introduction**

Before providing the details of the code used for making an asynchronous call with an asynchronous callback for completion, it is useful to provide an overview of the process.

**Process: Using asynchronous callbacks for completion**

When you use the callback as the completion technique, there are four main steps in the process:

1. To set up for the asynchronous call, create an asynchronous delegate that points to the callback method you want to have invoked when the asynchronous operation completes.

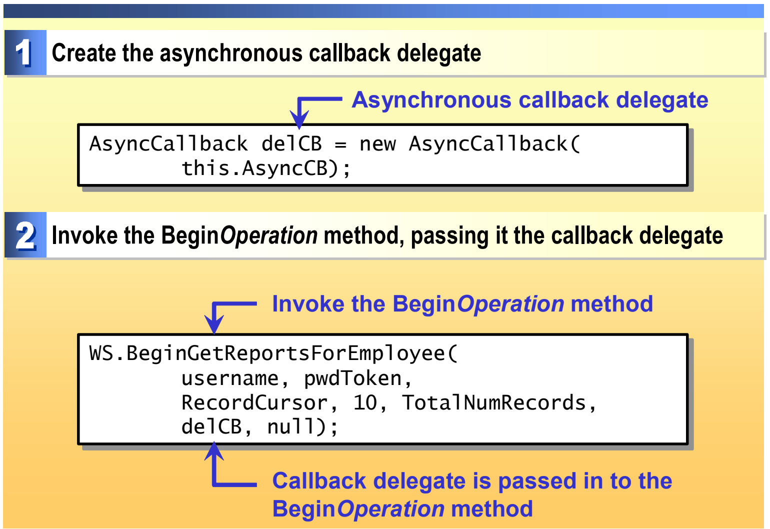
**Definition**

   A delegate is an object that you can use to call the methods of other objects. In other words, a delegate is an object-oriented, type-safe, secure way to provide the functionality of a function pointer in the .NET Framework.

2. Wherever appropriate in your application, initiate the asynchronous call by invoking the appropriate Begin*Operation* method, passing it the asynchronous delegate.
3. Inside the callback, invoke the appropriate End*Operation* method to retrieve results, if appropriate.
4. Return control to the main thread to update the user interface.

The remaining topics in this lesson provide the programming details related to these steps.

# How to Set Up and Initiate the Call



1 Create the asynchronous callback delegate

Asynchronous callback delegate

```
AsyncCallback delCB = new AsyncCallback(
        this.AsyncCB);
```

2 Invoke the Begin*Operation* method, passing it the callback delegate

Invoke the Begin*Operation* method

```
WS.BeginGetReportsForEmployee(
        username, pwdToken,
        RecordCursor, 10, TotalNumRecords,
        delCB, null);
```

Callback delegate is passed in to the Begin*Operation* method

**Introduction**

An appropriate scenario for using a callback to notify completion and return results of an asynchronous call is calling a method of an XML Web service to obtain information for a Windows Forms application.

**Procedure: Setting up and initiating the call**

1. Set up the asynchronous call by creating an asynchronous delegate that points to the callback method that you want to have invoked when the asynchronous operation completes, in this case the **AsyncCB** method of the form from which you make the call.

   ```
   AsyncCallback delCB = new AsyncCallback(
      this.AsyncCB);
   ```

   Notice that part of the .NET Framework support provided for asynchronous programming is the **AsyncCallback** delegate class. The signature for a method referred to by an AsyncCallback delegate is as follows.

   ```
   void AsyncCB (IAsyncResult ar);
   ```

   For more information about the **AsyncCallback** delegate, search for AsyncCallback delegate in the Microsoft Visual Studio® .NET Help documentation.

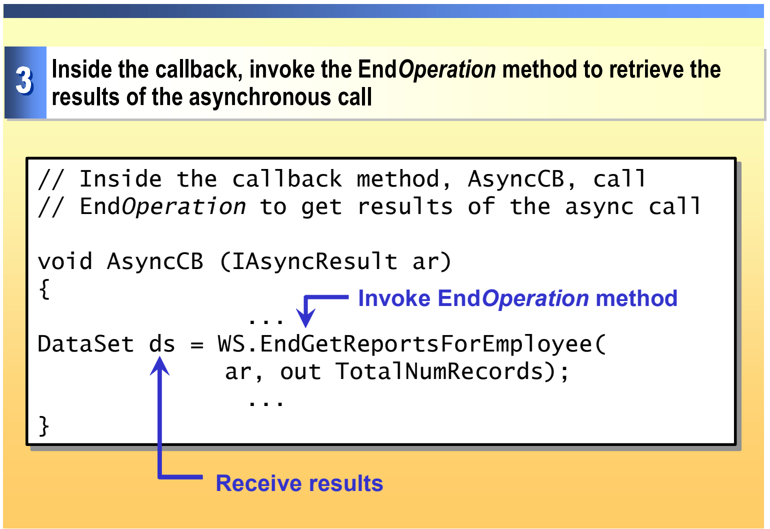
2. Initiate the asynchronous call by invoking the Begin*Operation* method, in this case **WS.BeginGetReportsForEmployee**, and passing it the callback delegate, in this case, **delCB**. Notice that the callback delegate is always passed in as the next to the last parameter.

   ```
   WS.BeginGetReportsForEmployee(
      username, pwdToken,
      RecordCursor, 10, TotalNumRecords,
      delCB, null);
   ```

The final parameter of the **BeginInvoke** method is of type **System.Object** and is for user data. This user data is passed into the callback when the operation is completed in the **AsyncState** property of the **IAsyncResult** parameter of the callback.

The .NET Framework will take care of allocating a thread from the thread pool to run the code in the **GetReportsForEmployee** method and, when it completes, will call the method pointed to by **delCB** (**AsyncCB**).

# How to Receive Completion Notification and Results



3 Inside the callback, invoke the End*Operation* method to retrieve the results of the asynchronous call

```
// Inside the callback method, AsyncCB, call
// EndOperation to get results of the async call

void AsyncCB (IAsyncResult ar)
{
                  ...
DataSet ds = WS.EndGetReportsForEmployee(
             ar, out TotalNumRecords);
                  ...
}
```

### Introduction

After the method that you called asynchronously has completed its work, the .NET Framework calls the asynchronous callback delegate that you designated.

### Procedure: Receiving completion notification and results

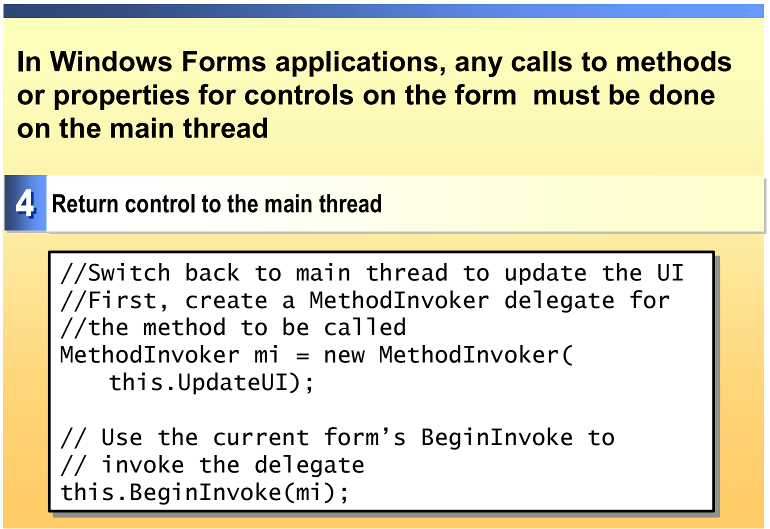
- Inside the asynchronous callback delegate, call the appropriate End*Operation* method and return any results.

```
// Inside the callback method, AsyncCB, call
// EndOperation to get the results of the async call

void AsyncCB (IAsyncResult ar)
{
               ...
ExpRepDataSet = WS.EndGetReportsForEmployee(
           ar, out TotalNumRecords);
               ...

}
```

### Exceptions

If the asynchronous operation throws an exception, it will be returned from the call to the End*Operation* method.

# How to Return Control to the Main Thread



**In Windows Forms applications, any calls to methods or properties for controls on the form must be done on the main thread**

**4 Return control to the main thread**

```
//Switch back to main thread to update the UI
//First, create a MethodInvoker delegate for
//the method to be called
MethodInvoker mi = new MethodInvoker(
    this.UpdateUI);

// Use the current form’s BeginInvoke to
// invoke the delegate
this.BeginInvoke(mi);
```

### Introduction

In Windows Forms applications, any changes to properties for controls on forms must be done on the main thread. Methods and properties for controls cannot be safely accessed from any other thread.

### Procedure: Returning control to the main thread

To return control to the main thread:

1. From inside the callback, switch back to the main thread.

```
//Switch back to main thread to update UI
//First, create a MethodInvoker delegate for
//the method to be called

MethodInvoker mi = new MethodInvoker(
   this.UpdateUI);
```

The .NET Framework provides a **MethodInvoker** delegate class that you can use to call generic methods. In the previous code sample, you create a delegate that refers to the **UpdateUI** method in the current object. The signature for a method referred to by a **MethodInvoker** delegate is as follows.

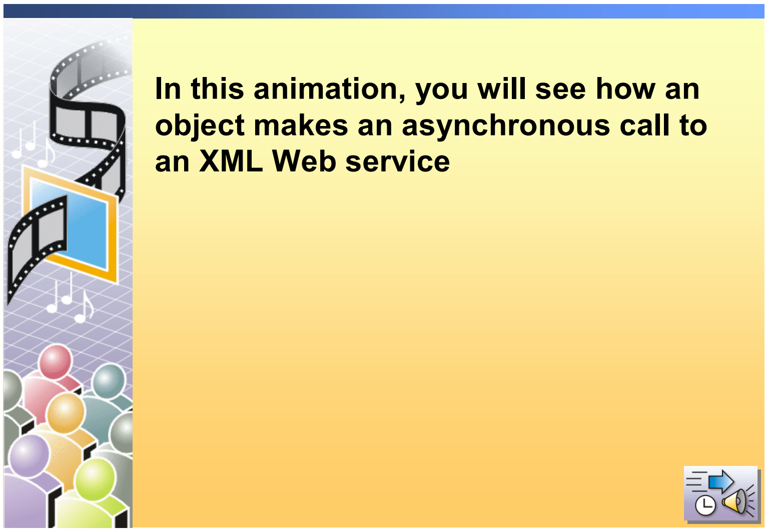
```
void UpdateUI();
```

2. Update the user interface.

```
//Use the current form’s BeginInvoke to
//invoke the delegate.

this.BeginInvoke(mi);
```

This call to the form’s **BeginInvoke** method will cause the method referred to by *mi* to be called on the main thread and therefore update the user interface (UI) safely. By calling the **BeginInvoke** method instead of the **Invoke** method, the worker thread used to make the asynchronous call returns to the thread pool faster, thus avoiding exhaustion of threads in the thread pool.

# Animation: Making an Asynchronous Call to an XML Web Service



In this animation, you will see how an object makes an asynchronous call to an XML Web service

In this animation, you will see how an object makes an asynchronous call to an XML Web service.

A Microsoft Windows Forms client application calls an XML Web service to retrieve and change information about expense reports. The XML Web service provides a **SubmitReport** method that takes an expense report, validates it, and adds it to the database of expense reports.

When a Web service reference is added to a Microsoft Visual Studio .NET project, a client proxy class for the XML Web service is generated. In addition to the synchronous **SubmitReport** method, the proxy includes the asynchronous **BeginSubmitReport** and **EndSubmitReport** methods.

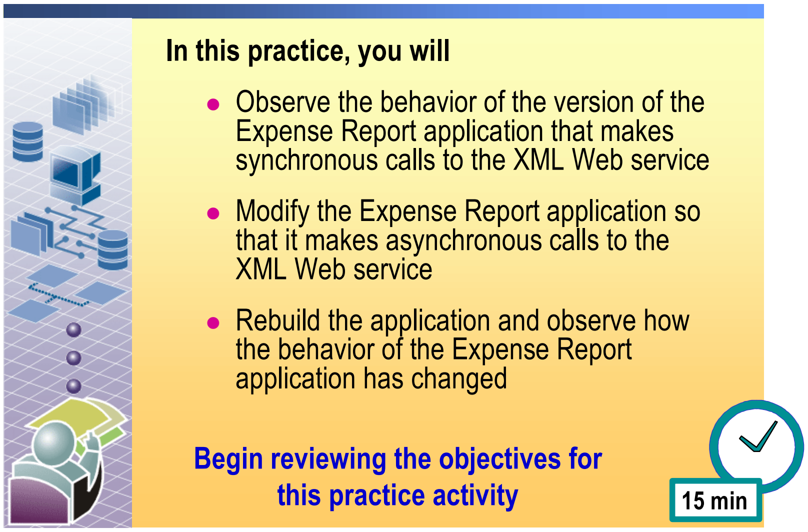
A callback method named **SubmitReportCallback** is used as the callback method for the asynchronous operation. The asynchronous operation is started by calling the proxy's **BeignSubmitReport** method, passing a delegate object that refers to the callback method. When the **BeginSubmitReport** method is called, it calls the **SubmitReport** method on another thread from the runtime's thread pool and returns immediately to the caller. The client is free to continue executing other code and the user interface remains responsive.

When the **SubmitReport** method completes, the callback method is invoked. The callback method invokes the **EndSubmitReport** method to retrieve the results of the XML Web service call. If the **SubmitReport** method completed successfully, the **EndSubmitReport** method returns the results; otherwise, an exception is thrown in the **EndSubmitReport** method for the callback to handle.

The application displays the results of the XML Web service call by using the **UpdateUI** method. The **UpdateUI** method manipulates controls to show the results. Controls can only be called safely from the application's main thread. The callback method is not executing on the application's main thread, and therefore cannot call the **UpdateUI** method directly.

Instead, it calls the form's thread-safe **BeginInvoke** method, passing it a delegate to the **UpdateUI** method. **BeginInvoke** then executes this method on the main thread.

# Practice: Making an Asynchronous Call to an XML Web Service



In this practice, you will

- Observe the behavior of the version of the Expense Report application that makes synchronous calls to the XML Web service
- Modify the Expense Report application so that it makes asynchronous calls to the XML Web service
- Rebuild the application and observe how the behavior of the Expense Report application has changed

Begin reviewing the objectives for this practice activity

### Introduction

In this practice, you will convert a synchronous XML Web service call to an asynchronous XML Web service call.

### Instructions

► **Observe the synchronous version of the Expense Report application**

1. Open the Mod07Practice.sln solution file in *install_folder*\Practices\Mod07\Mod07_01\Starter.
2. Open the MainForm.cs file and view the code.
3. Build the project and run the application.
4. Click **Call Web Service**. While the XML Web service call is happening, click the minimize button to try to minimize the application. Notice that the application is not responsive. It will not minimize until the XML Web service call completes.
5. Click **Exit** to quit the application.

► **Modify the Expense Report application to make asynchronous calls**

1. Show TODO comments in the Task List.

   To show TODO comments, on the **View** menu, point to **Show Tasks**, and then click **All**.

2. Find the first TODO comment in the code. Add the code to instantiate an **AsyncCallback** delegate that refers to the **WebserviceCallback** method of this class.

```
AsyncCallback asyncCB = new AsyncCallback(
                                this.WebserviceCallback);
```

3. Find the next TODO comment. Add the code to begin the call to the XML Web service.

```
WS.BeginGetReportsForEmployee(null, null, 0, 10,
                              TotalReports, asyncCB, null);
```

4. Find the next TODO comment and uncomment the line of code below the TODO comment.
5. Find the next two TODO comments. Comment out the rest of the code (the synchronous code) in the method.
6. Find the next TODO comment. Add the code to finish the call to the XML Web service.

```
ExpenseReports = WS.EndGetReportsForEmployee(result,
                                        out TotalReports);
```

7. Find the next TODO comment. Add the code to instantiate a **MethodInvoker** delegate that points to the **UpdateUI** method of this class.

```
MethodInvoker mi = new MethodInvoker(this.UpdateUI);
```

8. Find the next TODO comment. Add the code to call **BeginInvoke** on this form, passing the **MethodInvoker** delegate created in step 6.

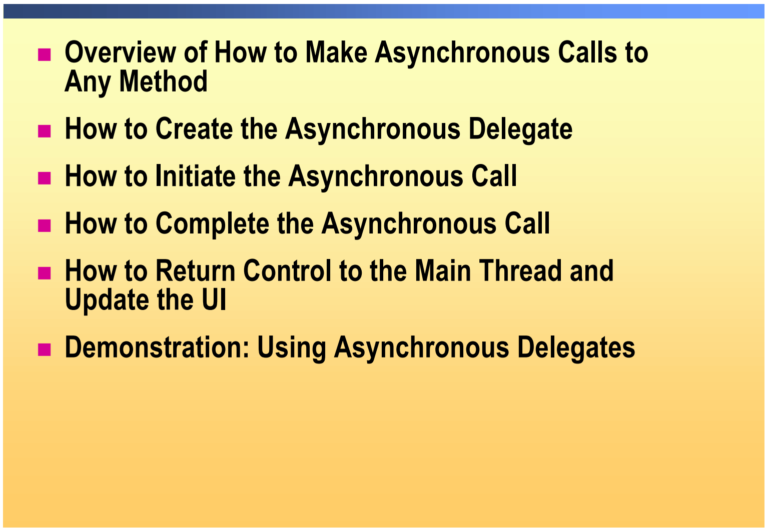
```
this.BeginInvoke(mi);
```

9. Save your files and build the project.

### ► Observe how the Expense Report application behavior has changed

1. Run the application.
2. Click **Call Web Service**. After the message **Async: Calling web service...** appears, try to minimize the window. Notice that the UI is still responsive during the call to the XML Web service.
3. Click **Exit** to quit the application.
4. Set breakpoints in the **CallBtn_Click**, **WebserviceCallback**, and **UpdateUI** methods. Rerun the application to see the flow of control of the asynchronous operation.

# Lesson: How to Make Asynchronous Calls to Any Method



- Overview of How to Make Asynchronous Calls to Any Method
- How to Create the Asynchronous Delegate
- How to Initiate the Asynchronous Call
- How to Complete the Asynchronous Call
- How to Return Control to the Main Thread and Update the UI
- Demonstration: Using Asynchronous Delegates

**Introduction**

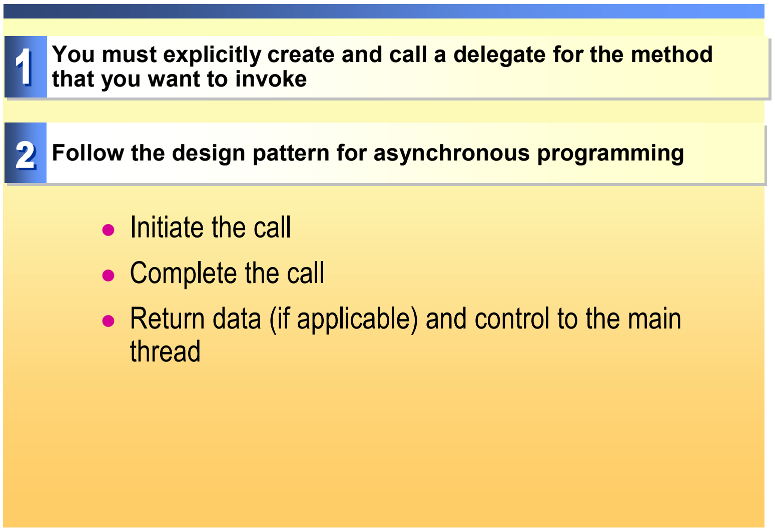
It is not necessary for a called object to provide explicit support for asynchronous calls. The .NET Framework provides services required for you to call any method asynchronously.

**Lesson objectives**

After completing this lesson, you will be able to:

- Describe how to make asynchronous calls to any method.
- Explicitly create and call asynchronous delegates.

# Overview of How to Make Asynchronous Calls to Any Method



1 You must explicitly create and call a delegate for the method that you want to invoke

2 Follow the design pattern for asynchronous programming

- Initiate the call
- Complete the call
- Return data (if applicable) and control to the main thread

****************************ILLEGAL FOR NON-TRAINER USE******************************

**Introduction**

When you make asynchronous calls to classes that contain built-in support for asynchronous programming, such as XML Web services and stream I/O, the compiler creates asynchronous delegates and their Begin*Operation* and End*Operation* methods for you.

**Explicitly create the asynchronous delegate**

When you want to asynchronously call methods in classes without built-in support, you must explicitly create and call the delegate for the method that you want to invoke.

**Definition**

A delegate is an object-oriented, type-safe, secure way to provide the functionality of a function pointer in the .NET Framework. Asynchronous delegates allow you to call a synchronous method in an asynchronous manner.

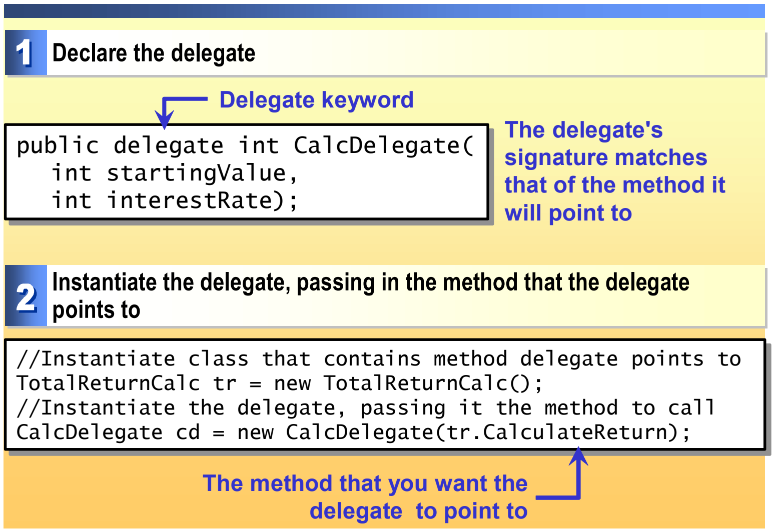
**Follow design pattern for asynchronous programming**

After creating the asynchronous delegate, you then follow the same process that you would use for making asynchronous calls to classes with built-in support:

1. Choose the completion mechanism that you want to use and do any setup the completion mechanism requires.
2. Initiate the asynchronous call.
3. Receive the completion notification and any results.
4. Return control to the main thread and update the UI, as appropriate.

Techniques for protecting state and data is covered in the topic How to Protect State and Data in a Multithreaded Environment in this module.

# How to Create the Asynchronous Delegate



1 Declare the delegate

Delegate keyword

```
public delegate int CalcDelegate(
   int startingValue,
   int interestRate);
```

The delegate's signature matches that of the method it will point to

2 Instantiate the delegate, passing in the method that the delegate points to

```
//Instantiate class that contains method delegate points to
TotalReturnCalc tr = new TotalReturnCalc();
//Instantiate the delegate, passing it the method to call
CalcDelegate cd = new CalcDelegate(tr.CalculateReturn);
```

### Introduction

After you have explicitly declared an asynchronous delegate, you can use the asynchronous methods of a delegate object created for you by the compiler. The .NET Framework will still make the threading infrastructure necessary for asynchronous programming transparent to you.

### Procedure: Declaring the delegate

To declare the delegate, use the **delegate** keyword.

```
public delegate int CalcDelegate(
  int startingValue,
  int interestRate);
```

The previous code declares a delegate to a method that returns an integer and takes two integers as parameters.

Because Microsoft Visual C#™ and Microsoft Visual Basic® .NET compilers support asynchronous delegates, they generate the **Invoke** method and the **BeginInvoke** and **EndInvoke** methods when you declare the delegate.

Use the **Invoke** method if you want to call the target method synchronously.

Use the **BeginInvoke** method to call the target method asynchronously. The runtime queues the request and returns immediately to the caller. The target method will be called on a thread from the thread pool. The original thread that submitted the request is free to continue executing in parallel to the target method, which is running on a thread pool thread.

If a callback has been specified on **BeginInvoke**, it is called when the target method returns. In the callback, the **EndInvoke** method is used to obtain the return value and the in/out parameters. If the callback is not specified on **BeginInvoke**, you can use the other asynchronous design pattern techniques, for example, polling, on the original thread that submitted a request.

### Procedure: Instantiating the delegate

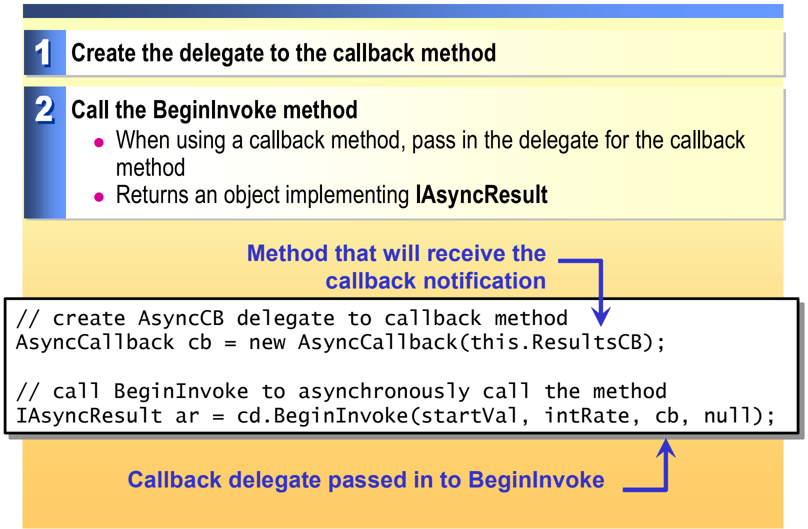
Instantiate the class that contains the method that the delegate will point to.

```
TotalReturnCalc tr = new TotalReturnCalc();
```

Then instantiate the delegate, passing in the method it points to.

```
CalcDelegate cd = new CalcDelegate(tr.CalculateReturn);
```

# How to Initiate the Asynchronous Call



1 Create the delegate to the callback method

2 Call the BeginInvoke method
- When using a callback method, pass in the delegate for the callback method
- Returns an object implementing **IAsyncResult**

Method that will receive the callback notification

```
// create AsyncCB delegate to callback method
AsyncCallback cb = new AsyncCallback(this.ResultsCB);

// call BeginInvoke to asynchronously call the method
IAsyncResult ar = cd.BeginInvoke(startVal, intRate, cb, null);
```

### Introduction

Using a callback method as the completion mechanism is most appropriate for Windows Forms applications. To use the callback method, you must create a delegate to safely invoke it.

### Procedure: Creating the delegate to the callback method

To create the delegate that allows you to call the callback method, instantiate an **AsyncCallback** object:

```
AsyncCallback cb = new AsyncCallback(this.ResultsCB);
```

In the code example above, **this** refers to the object from which you are making the asynchronous call. Alternatively, the method that handles the callback could be in a different object, if necessary for your scenario.

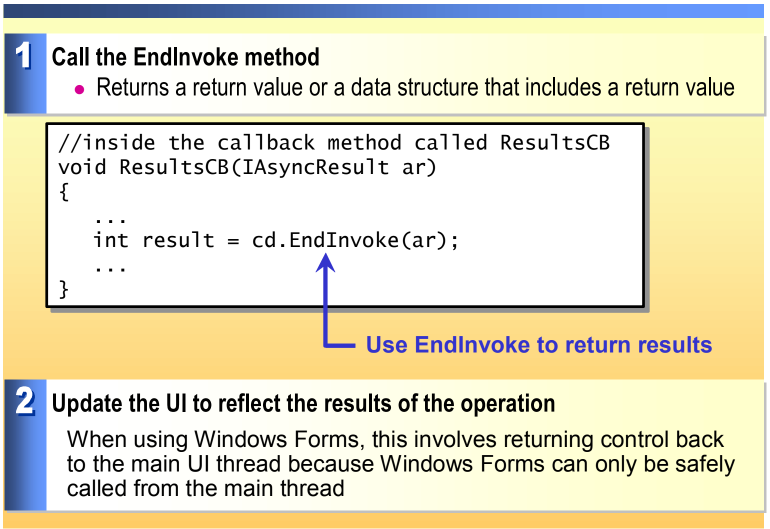
### Procedure: Calling the BeginInvokemethod

To call the **BeginInvoke** method, pass the callback delegate, named **cb** in the code example, to the **BeginInvoke** method as the second to last parameter.

```
IAsyncResult ar = cd.BeginInvoke(startVal, intRate, cb,
    null);
```

The final parameter of the **BeginInvoke** method is of type **System.Object** and is for user data. This user data is passed into the callback when the operation is completed in the **AsyncState** property of the **IAsyncResult** parameter of the callback.

# How to Complete the Asynchronous Call



1 **Call the EndInvoke method**

- Returns a return value or a data structure that includes a return value

```
//inside the callback method called ResultsCB
void ResultsCB(IAsyncResult ar)
{
   ...
   int result = cd.EndInvoke(ar);
   ...
}
```

Use EndInvoke to return results

2 **Update the UI to reflect the results of the operation**

When using Windows Forms, this involves returning control back to the main UI thread because Windows Forms can only be safely called from the main thread

**Introduction**

When the asynchronous operation completes, the .NET Framework calls your callback method. Your callback method, in turn, calls the **EndInvoke** method to notify the caller of completion. For Windows Forms applications, you must return control to the main thread before updating the user interface.

**Procedure: Calling the EndInvoke method**

You can call the **EndInvoke** method to retrieve the results of the operation.

```
int result = cd.EndInvoke(ar);
```

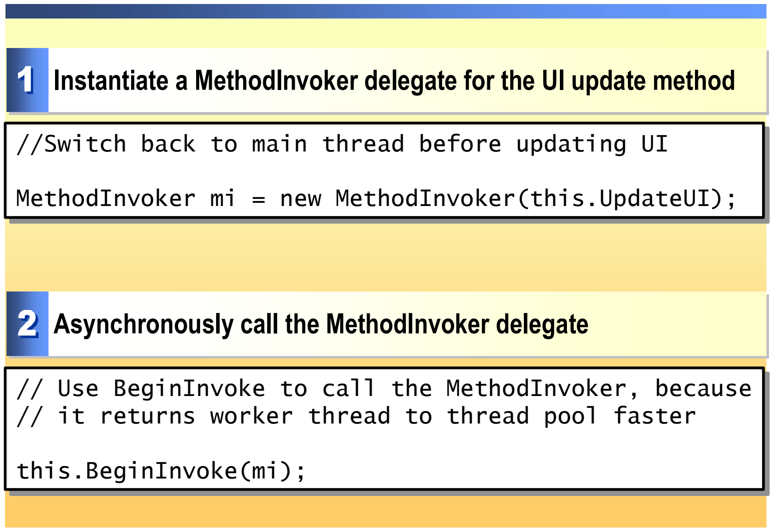
**Exceptions**

If the asynchronous operation throws an exception, it will be returned from the call to the **EndInvoke** method.

**Updating the UI**

In Windows Forms applications, you must return control to the main thread before updating the user interface.

# How to Return Control to the Main Thread and Update the UI



1 Instantiate a MethodInvoker delegate for the UI update method

```
//Switch back to main thread before updating UI

MethodInvoker mi = new MethodInvoker(this.UpdateUI);
```

2 Asynchronously call the MethodInvoker delegate

```
// Use BeginInvoke to call the MethodInvoker, because
// it returns worker thread to thread pool faster

this.BeginInvoke(mi);
```

### Introduction

As with the design pattern discussed in the previous lesson, you use the **MethodInvoker** delegate to transfer control to the main thread.

### Procedure: Instantiating a MethodInvoker delegate

```
MethodInvoker mi = new MethodInvoker(this.UpdateUI);
```

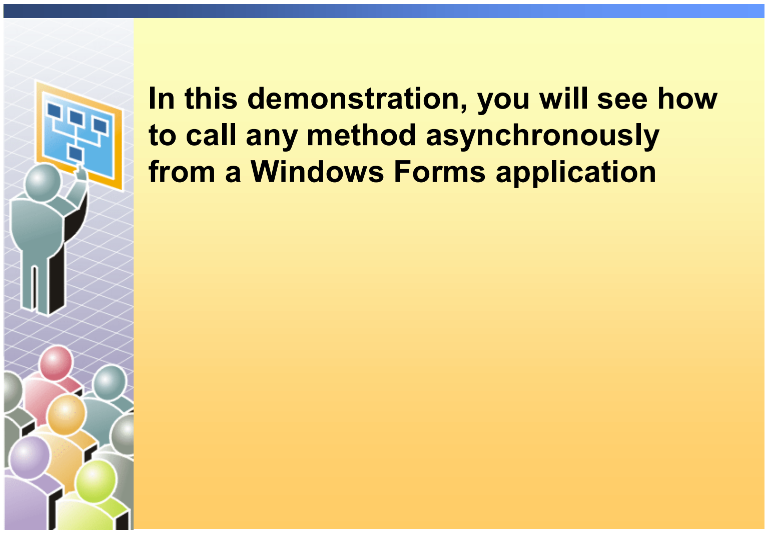
### Procedure: Invoking the MethodInvoker delegate asynchronously

```
this.BeginInvoke(mi);
```

By calling the **UpdateUI** method asynchronously, by using **BeginInvoke** instead of **Invoke**, the worker thread used to make the asynchronous call returns to the thread pool faster, thus promoting better application performance.

Notice that this step is necessary only if the UI in a Windows Forms application must be updated. Also notice that all of the steps after creating the delegate are the same in this case as they are in the case in which a delegate is not used.

# Demonstration: Using Asynchronous Delegates



In this demonstration, you will see how to call any method asynchronously from a Windows Forms application

**Introduction**

In this demonstration, you will see how to call a method asynchronously from a Windows Forms application.

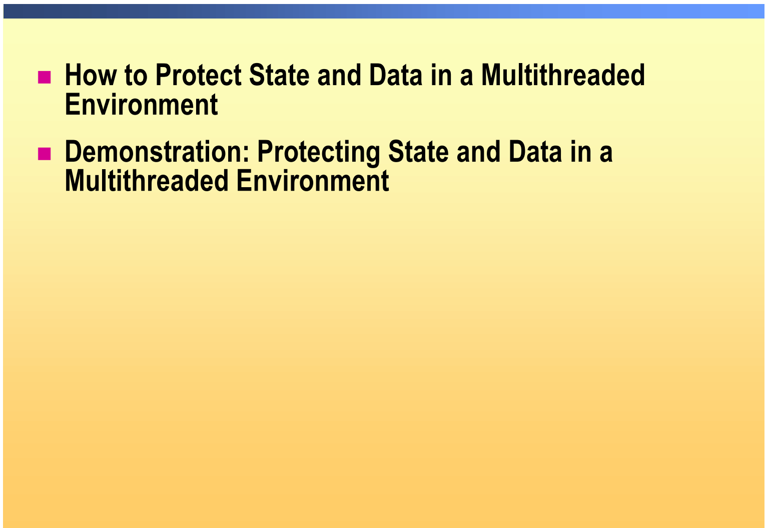
### ► To view the code for the application

1. Open the AsyncDelegates.sln solution file in *install_folder*\Democode\Mod07\Mod07_02\AsyncDelegates.
2. Open the Class1.cs source file and view the code.
3. Notice the **FactorizeDelegate** declaration near the top of the file. This shows an example of a delegate declaration.
4. Notice the **Factorize** method in the **PrimeFactorizer** class. This is the method the delegate will refer to and invoke asynchronously.
5. Notice the **FactorizeNumber** method in the **Simple** class. This is the method that instantiates a **PrimeFactorizer** object, then instantiates a delegate referring to the **Factorize** method in that class. Then it calls **BeginInvoke** on the delegate.
6. Notice the **FactorizedResults** method in the **Simple** class. This is the callback method that is invoked when the asynchronous operation completes.

### ► To view the flow of control as the application executes

1. Set breakpoints at the top of the **Factorize**, **FactorizeNumber**, and **FactorizedResults** methods.
2. Run the application, noticing the flow of control between the three methods mentioned above.

# Lesson: Protecting State and Data in a Multithreaded Environment



- How to Protect State and Data in a Multithreaded Environment
- Demonstration: Protecting State and Data in a Multithreaded Environment

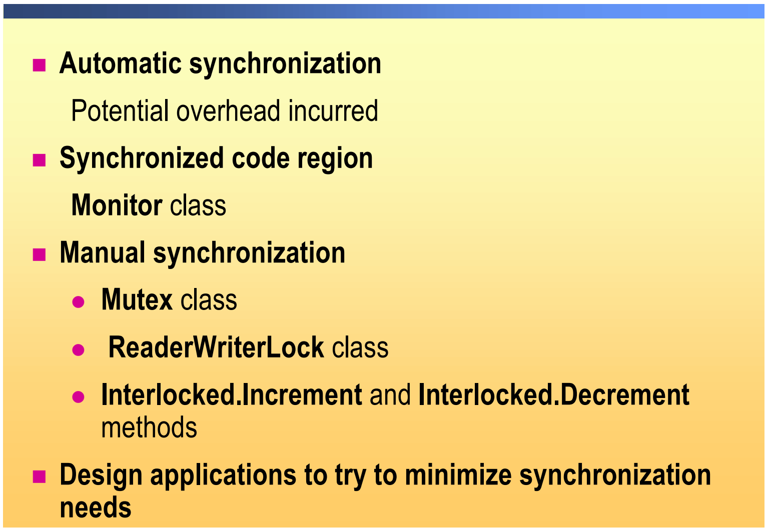
**Introduction**

No synchronization is the default for objects. Therefore, when you use asynchronous programming techniques, you must consider how to protect state and data in this multithreaded environment.

**Lesson objective**

After completing this lesson, you will be able to describe techniques to protect state and data in a multithreaded environment.

# How to Protect State and Data in a Multithreaded Environment



- **Automatic synchronization**

  Potential overhead incurred
- **Synchronized code region**

  **Monitor** class
- **Manual synchronization**
  - **Mutex** class
  - **ReaderWriterLock** class
  - **Interlocked.Increment** and **Interlocked.Decrement** methods
- **Design applications to try to minimize synchronization needs**

### Introduction

For many Windows Forms applications, little state protection code may be necessary, because most access of object state happens on the main thread. However, this will not be the case for every application. The **System.Threading** namespace provides classes and interfaces for synchronizing access to data to provide thread safety.

### Automatic synchronization

You can use the **SynchronizationAttribute** on any class that is derived from **ContextBoundObject** to synchronize all instance methods and fields. All objects in the same context domain share the same lock. Multiple threads are allowed to access the instance methods and fields, but only a single thread is allowed at any one time. Static members are not protected from concurrent access by multiple threads.

**Note** Using automatic synchronization can incur more overhead than using manual synchronization techniques.

### Synchronized code regions

Another technique that you can use is a compiler keyword to synchronize blocks of code, instance methods, and static methods. In Visual C#, the keyword is **lock**. This keyword uses the **Monitor** class to lock the object.

When you use the **lock** keyword, the compiler generates code. The Visual C# compiler emits a **try**/**finally** block with **Monitor.Enter** at the beginning of the **try**, and **Monitor.Exit** in the **finally** block. If an exception is thrown inside of the **lock** block, the **finally** handler runs to allow you to perform any clean-up work.

The C# statement, of the form **lock(x)** where **x** is an expression of a *reference-type*, is equivalent to the following except that **x** is evaluated only once:

```
System.Threading.Monitor.Enter(x);
try {
   ...
}
finally {
   System.Threading.Monitor.Exit(x);
}
```

The **lock** keyword marks a statement block as a critical section.

**lock(***expression***)** *statement_block,*

where *expression* specifies the object that you want to lock on. *expression* must be a reference type.

Typically, *expression* is **this** if you want to protect an instance variable, or **typeof(class)** if you want to protect a static variable or if the critical section occurs in a static method in the specified class. The *statement_block* includes the statements of the critical section.

For example, to synchronize access in a static method:

```
using System;
using System.Threading;

class Cache
{
  public static void Add(object x)
  {
      // method code that doesn't require exclusive access
      lock (typeof(Cache))
      {
          // code requiring exclusive access to static data
      }
      // method code that doesn't require exclusive access
  }
}
```

Or, to synchronize access in an instance method:

```
class Counter
{
  public override int Read(int threadNum)
  {
      // ...
      // method code that doesn't require exclusive access
      lock(this)
      {
          // code requiring exclusive access to instance data
      }
      // method code that doesn't require exclusive access
  }
}
```

### Manual synchronization

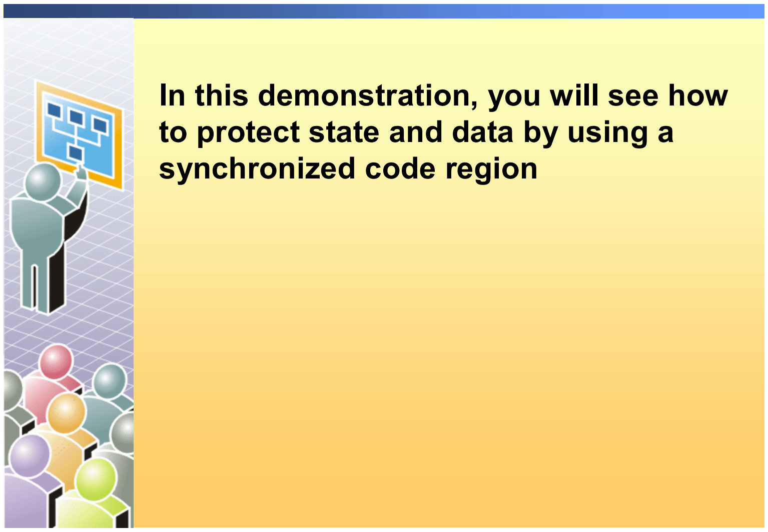
You can use the **Monitor**, **Interlocked**, **Mutex**, **ManualResetEvent**, **AutoResetEvent**, and **ReaderWriterLock** classes to acquire and release a lock to protect global, static, and instance fields and global, static, and instance methods.

### Designing applications to minimize synchronization needs

For many UI-based applications, little or no special synchronization code is required. All of the changes of the UI must happen on the main application thread, so this usually forces synchronous access to state information. Also, careful enabling and disabling of UI elements can prevent the application from getting into an undefined state. For example, the Expense Reporting sample application enables and disables buttons as asynchronous XML Web service calls are made to prevent threading-related problems. Therefore, no special synchronization code is required in the Expense Reporting application, other than the use of **MethodInvoker** delegates to switch control to the main thread upon completion of asynchronous calls.

For more information about synchronization techniques, see Module 14, “Threading and Asynchronous Programming” in Course 2349B, *Programming with the Microsoft .NET Framework (Microsoft Visual C# .NET)*.

# Demonstration: Protecting State and Data in a Multithreaded Environment



In this demonstration, you will see how to protect state and data by using a synchronized code region

**Introduction**

In this demonstration, you will see how to protect state and data by using a synchronized code region.

► **To view the code for the ThreadSafety application**

1. Open the ThreadSafety.sln solution file in *install_folder*\Democode\Mod07\Mod07_03\ThreadSafety.
2. Open the Class1.cs source file and view the code.
3. Notice the **Read** method in the **CounterUnsafe** class.

   This method reads the **count** member variable twice, with a delay in between the two readings. If other threads change the value of **count** during that interval, two different values will be read.
4. Notice the **Increment** method in the **CounterUnsafe** class.

   This method increments the **count** variable, with a long delay between when it reads the value of count and writes the updated value of **count**. If other threads try to update the value of **count** during this interval, the value of **count** written by this method overwrites the results of other calls.
5. Notice the **Read** method in the **CounterUsingLock** class.

   This method reads the **count** variable twice, with a delay in between the two readings. However, this method accesses **count** in a **lock** code block, so the value of **count** will not change between the two readings.
6. Notice the **Increment** method in the **CounterUsingLock** class.

   This method increments the **count** variable, with a long delay between when it reads the value of **count** and writes the updated value of **count**. However, this method accesses **count** in a **lock** code block, so the value of **count** cannot be changed while this method accesses it.

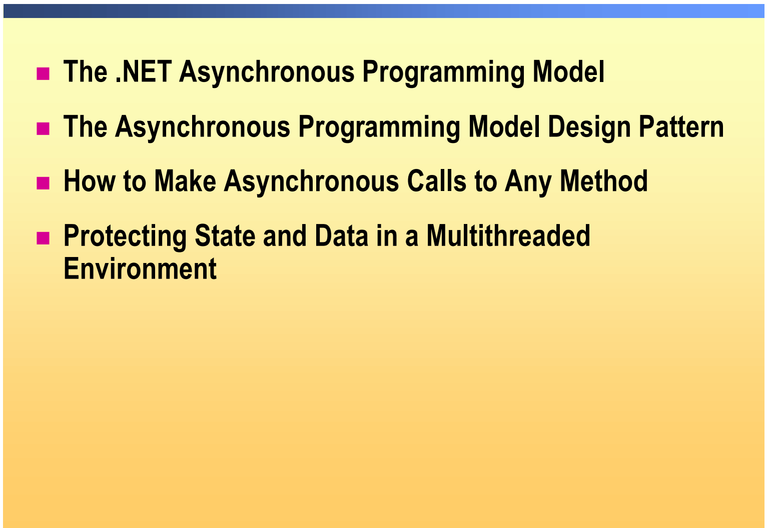
### ► To run the application and view the output

1. To open the Visual Studio .NET command prompt, click **Start**, point to **All Programs**, point to **Microsoft Visual Studio .NET**, point to **Visual Studio .NET Tools**, and then click **Visual Studio .NET Command Prompt**.
2. Change directories to *install_folder*\Democode\Mod07\Mod07_03\ThreadSafety\bin\debug. Run the application ThreadSafety.exe from the command line.
3. View the output of the test. The output should look similar to the following.

```
Unsafe test:
Start Resource writing (Thread=0) count: 0
Stop  Resource writing (Thread=0) count: 1
Start Resource reading (Thread=1)count: 1
Stop  Resource reading (Thread=1) count: 1
Start Resource writing (Thread=2) count: 1
Stop  Resource writing (Thread=2) count: 2
Start Resource reading (Thread=3)count: 2
Stop  Resource reading (Thread=3) count: 2
Start Resource writing (Thread=4) count: 2
Stop  Resource writing (Thread=4) count: 3
Start Resource reading (Thread=5)count: 3
Start Resource writing (Thread=6) count: 3
Stop  Resource reading (Thread=5) count: 3
Start Resource reading (Thread=7)count: 3
Stop  Resource reading (Thread=7) count: 3
Start Resource writing (Thread=8) count: 3
Stop  Resource writing (Thread=6) count: 4
Start Resource reading (Thread=9)count: 4
Stop  Resource reading (Thread=9) count: 4
Stop  Resource writing (Thread=8) count: 4
All Unsafe threads have completed.
Lock test:
Start Resource writing (Thread=0) count: 0
Stop Resource writing (Thread=0) count: 1
Start Resource reading (Thread=1)count: 1
Stop Resource reading (Thread=1) count: 1
Start Resource writing (Thread=2) count: 1
Stop Resource writing (Thread=2) count: 2
Start Resource reading (Thread=3)count: 2
Stop Resource reading (Thread=3) count: 2
Start Resource writing (Thread=4) count: 2
Stop Resource writing (Thread=4) count: 3
Start Resource reading (Thread=5)count: 3
Stop Resource reading (Thread=5) count: 3
Start Resource writing (Thread=6) count: 3
Stop Resource writing (Thread=6) count: 4
Start Resource reading (Thread=7)count: 4
Stop Resource reading (Thread=7) count: 4
Start Resource writing (Thread=8) count: 4
Stop Resource writing (Thread=8) count: 5
Start Resource reading (Thread=9)count: 5
Stop Resource reading (Thread=9) count: 5
All Lock threads have completed.
```

Notice that although **Increment** was called five times for the unsafe test, the value of the counter never reaches five. Also, the start and stop notifications are interspersed among different threads. In the lock version, the start and stop notifications are paired for each thread, and the counter is correctly incremented to five.

# Review



- The .NET Asynchronous Programming Model
- The Asynchronous Programming Model Design Pattern
- How to Make Asynchronous Calls to Any Method
- Protecting State and Data in a Multithreaded Environment

1. When and why would you use asynchronous programming techniques in an application?

   **You should use asynchronous programming techniques in an application with significant user interaction to avoid blocking the user interface during slow method calls, such as XML Web service calls.**

2. In a Windows Forms application, what are the main steps to follow to use the asynchronous programming design pattern with an asynchronous callback for completion?

   **The main steps are:**

   1. **Create the asynchronous callback delegate.**
   2. **Invoke the Begin*Operation* method, passing it the callback.**
   3. **Inside the callback, invoke the End*Operation* method for completion.**
   4. **Return control to the main thread and update the user interface.**

3. List the four ways to complete an asynchronous call.

   **Four ways to complete the asynchronous call are:**

   - **Using a callback**
   - **Polling**
   - **Calling End*Operation***
   - **Waiting on a handle**

4. Why do you need to return control to the main thread before updating the UI in Windows Forms applications?

   **You need to return control to the main thread before updating the UI because Windows Forms controls can be safely accessed only from the main thread.**

5. What additional step must you follow to asynchronously call a method of any class?

   **When you want to asynchronously call methods in classes without built-in asynchronous support, you must explicitly create the delegate for the method you want to invoke. After that, follow the same steps required to asynchronously call methods in a class with built-in asynchronous support.**

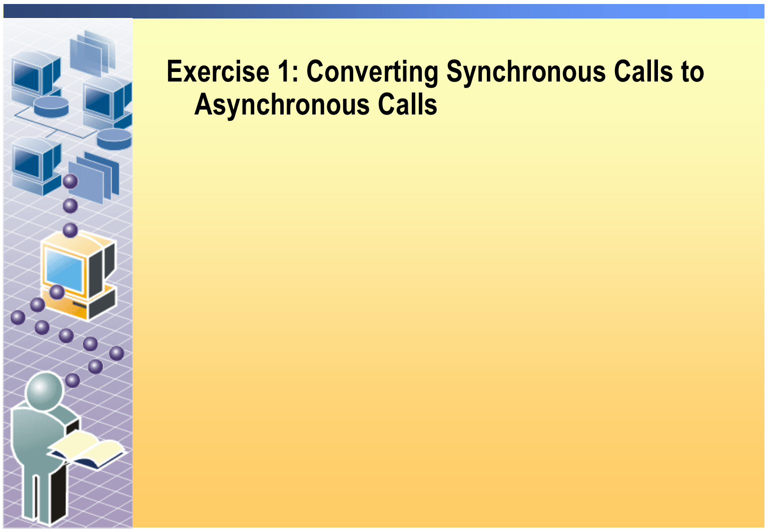
6. If an asynchronous operation throws an exception, when does your code receive the exception?

   **The exception is thrown when the End*Operation* method is called.**

7. What keyword is provided by the C# language for controlling access by multiple threads to a block of code?

   **The C# language provides the lock keyword to control access by multiple threads to a block of code.**

# Lab 7.1: Making Asynchronous Calls to an XML Web Service



Exercise 1: Converting Synchronous Calls to Asynchronous Calls

**Objectives**

After completing this lab, you will have demonstrated your ability to:

- Make asynchronous calls by using the .NET Framework asynchronous design pattern.
- Convert synchronous calls to asynchronous calls by using the .NET Framework asynchronous design pattern.
- Use the Visual Studio .NET debugger to follow the flow of asynchronous calls.

---

**Note** This lab focuses on the concepts in Module 7, “Asynchronous Programming,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. As a result, this lab may not comply with Microsoft security recommendations.

---

**Prerequisites**

Before working on this lab, you must have:

- The knowledge and skills to develop a simple Windows Forms application by using a Visual Studio .NET–compatible programming language.
- The knowledge and skills to debug an application by using Visual Studio .NET.
- The knowledge and skills to call XML Web services and make asynchronous calls by using the .NET Framework asynchronous design pattern.

**Scenario**

The Expense Report application makes numerous calls to an XML Web service. To improve the responsiveness and usability of the application, all of the XML Web service calls should be made asynchronously.

To make the calls asynchronous, you must perform the following tasks:

- Convert the synchronous XML Web service calls to asynchronous calls.
- Write the method that handles the asynchronous callback.
- Write the method that updates the user interface.

In the starter code for this lab, the asynchronous callback method and user interface update method are already written, and some of the XML Web service calls are already converted.

In this lab, you will convert the rest of the synchronous XML Web service calls to asynchronous calls and then use the Visual Studio debugger to follow the flow of the asynchronous calls in the resulting code.

**Lab Setup**

There are starter and solution files associated with this lab. Browse to *install_folder*\Labfiles\Lab07_1\Ex01\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab07_1\Ex01\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

**Estimated time to complete this lab: 15 minutes**

# Exercise 1
# Converting Synchronous Calls to Asynchronous Calls

In this exercise, you will convert some synchronous XML Web service calls to asynchronous calls. You will also use the Visual Studio debugger to view the flow of asynchronous calls and test your application to ensure that it has the expected results.

| Tasks | Additional information |
|---|---|
| **1.** Open the ExpenseReport project in Visual Studio .NET. Browse to *install_folder*\Labfiles\Lab07_1\Ex01\Starter to find the project files. | **a.** For more information about opening a project file and starting an application, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. For additional information about starting an application in the Designer, in Index, search by using the phrase **Debugging Windows Applications**. |
| **2.** Open the ExpenseReportList source file, and view the code. Refer to the synchronous versions of the calls to see which parameters to pass:<br>• The first four parameters should be the same.<br>• The fifth parameter should not use the **ref** keyword in the asynchronous version.<br>• The sixth parameter should be the callback delegate.<br>• The seventh parameter should be **null**. | **a.** See the TODO comments in the code for more detailed information about the tasks that you must perform to convert the synchronous XML Web service calls in the constructor to asynchronous calls.<br>**b.** For more information about how to make asynchronous calls, see the following resources:<br>• Practice: Making an Asynchronous Call to an XML Web Service in Module 7 "Asynchronous Programming," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Lesson: The Asynchronous Programming Model Design Pattern in Module 7 "Asynchronous Programming," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. For more information about the asynchronous design pattern, search by using the phrase **Including Asynchronous Calls**. |

| Tasks | Additional information |
|---|---|
| **3.** View the flow of asynchronous code by setting breakpoints in the **ExpenseReportList** constructor and the **GetReportsCB** and **UpdateReportsUI** methods.<br>**a.** Run the application in the Visual Studio debugger. Click **View Submitted Reports**. Step through the code, and make sure it executes as expected.<br>**b.** Confirm the correct behavior in the user interface by making sure that the expense reports are downloaded asynchronously when a new list window is opened. | **a.** For more information about debugging and controlling the flow of asynchronous calls, see the following resources:<br>• Practice: Making an Asynchronous Call to an XML Web Service in Module 7 "Asynchronous Programming," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. This practice contains information about how to debug an asynchronous call.<br>• Lesson: The Asynchronous Programming Model Design Pattern in Module 7 "Asynchronous Programming," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. This lesson contains information about the flow of control of an asynchronous call. |

# Module 8: Enhancing the Usability of Applications

**Contents**

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

# Instructor Notes

**Presentation:**
**60 minutes**

**Lab:**
**30 minutes**

In this module, students learn to use the accessibility, Help, and localization features available in the Microsoft® .NET Framework.

After completing this module, students will be able to:

- Use .NET Framework features to add and enable accessibility features in an application.
- Add support for context-sensitive Help, Help menus, and ToolTips to an application.
- Use localization properties and resource files to create a localized version of a .NET Framework Windows Forms application.

**Required materials**

To teach this module, you need the Microsoft PowerPoint® file 2555A_08.ppt.

**Preparation tasks**

To prepare for this module:

- Read all of the materials for this module.
- Complete the practices and lab.

# How to Teach This Module

This section contains information that will help you to teach this module.

- If students are interested in referencing code examples in other languages, point them to “Language Equivalents” in the Help documentation for the Microsoft Visual Studio® .NET development system. This section provides examples in languages such Microsoft Visual Basic® .NET, C#, and Java.
- Lab 8.1 is based on the Expense Report application in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C#™ .NET)* and is intended to simulate a real-world environment in which students will demonstrate what they learned during the lecture and practice portions of the module. The lab does not provide step-by-step detailed instructions; instead, the students are given tasks to complete in the left column and a list of resources that they can use (if they need help) in the right column. Students get the hands-on experience that they need by completing the practice activity in the module.

# Lesson: Adding Accessibility Features

This section describes the instructional methods for teaching this lesson.

For the most current information about accessibility support, you can direct students to the Microsoft Accessibility Web page at http://www.microsoft.com/enable/.

### Practice: Adding Accessibility Support to an Application

This practice requires a sound card to hear Microsoft Narrator. If the student computers do not have sound cards, make the practice an instructor-led demonstration instead.

# Lesson: Adding Help to an Application

This section describes the instructional methods for teaching this lesson.

### How to Add Context-Sensitive Help for Forms and Controls

In this topic, you can mention to students that one advantage of using HTML files for Help content is that this makes it very easy to update.

### Practice: Adding Help to an Application

The next practice in this lesson—called Practice: Adding ToolTips to an Application—is very short. If you prefer, you can delay this practice until you have taught the How to Display Help with the ToolTip Control topic and then combine the two practices.

### Practice: Adding ToolTips to an Application

This practice is very short. If you prefer, you can delay the previous practice—called Practice: Adding Help to an Application—until you have taught this topic and then combine the two practices.

# Lesson: Localizing an Application

This section describes the instructional methods for teaching this lesson.

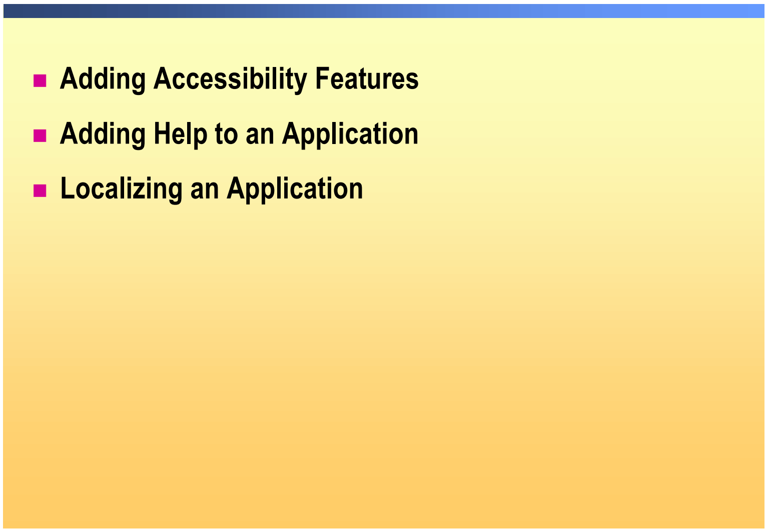
To help students understand the use of *culture* and *region*, as these terms are used in the context of localizing applications, you can ask students in the class to provide some examples from their own experience or provide some of your own.

# Lab 8.1: Enhancing the Usability of an Application

This section describes the instructional methods for teaching this lesson.

**Important** In Exercise 5, Localizing Resources in an Application, if you cut and paste the text from the Name column in the lab instructions, such as AppUnavailableMessage, from the softcopy of the module Word document directly into the resource file entries, you will get an unhandled exception. This is because the pasted string will include some non-printable characters that are not valid for a resource file.

# Overview



- Adding Accessibility Features
- Adding Help to an Application
- Localizing an Application

**Introduction**

Enhancing the usability of an application has several benefits. It can help all users learn how to use an application more quickly and more efficiently in their work. It can also make the application available for wider audiences—from people with disabilities, such as people who are blind or deaf, to worldwide audiences who speak different languages and who come from different cultures.

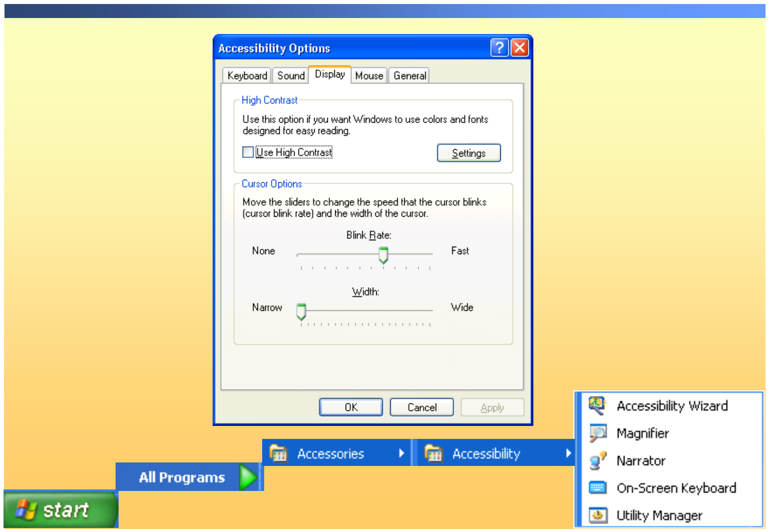
This module explores the support that is available in the Microsoft® .NET Framework that allows developers to more easily enhance the usability of the applications that they develop.

**Objectives**

After completing this module, you will be able to:

- Use .NET Framework features to add and enable accessibility features in an application.
- Add support for context-sensitive Help, Help menus, and ToolTips to an application.
- Use localization properties and resource files to create a localized version of a .NET Framework Windows Forms application.

# Lesson: Adding Accessibility Features



Use this option if you want Windows to use colors and fonts designed for easy reading.

☐ Use High Contrast

Move the sliders to change the speed that the cursor blinks (cursor blink rate) and the width of the cursor.

****************************ILLEGAL FOR NON-TRAINER USE******************************

**Introduction**

By following accessibility design practices and by using support for accessibility features in the .NET Framework, you can make your applications available to your customers or employees with accessibility needs.

This lesson will briefly describe the accessibility support available in the .NET Framework and how to make the forms and controls of your applications accessible. The lesson also covers how to test the application by using Microsoft Narrator, an accessibility utility that is included with the Microsoft Windows® XP operating system.

For more information about supporting accessibility in your applications, refer to the topics under “Accessibility” in the MSDN Library section titled “User Interface Design and Development.”

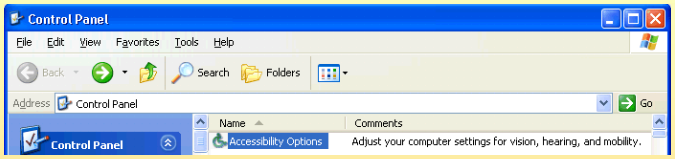
**Lesson objectives**

After completing this lesson, you will be able to:

- Describe the accessibility support that is available in the .NET Framework.
- Make the forms and controls in your application accessible.
- Test the accessibility of your application.

# Accessibility Support in the .NET Framework

- Accessibility options



Adjust your computer settings for vision, hearing, and mobility.

- Microsoft accessibility aids
  - Narrator
  - Magnifier
  - On-Screen Keyboard
- Developers can provide accessibility support by setting properties on forms and controls in their applications

****************************ILLEGAL FOR NON-TRAINER USE******************************

**Introduction**

One of the goals for Windows XP was to deliver easier and better customization that caters to specific hearing, vision, and mobility needs.

**Windows support**

The Windows operating system provides several accessibility options that users can set from Control Panel to control interaction with the keyboard, mouse, display, and sounds.

**Definition**

*Accessibility aids* are specialized programs and devices that help people with disabilities use computers more effectively. Some accessibility aids are built in to the Windows operating system. There are many types of aids. Some examples include:

- Screen enlargers for people who have low vision.
- Screen readers for people who are blind.
- Voice input utilities for people who control their computers by providing verbal commands to their computers instead of by using the keyboard or mouse.

In this lesson, you will use Narrator—the screen reader included with Windows XP—to demonstrate and test the accessibility properties that you can set on the user interface elements of your applications.

### Developer support

For more information about the guidelines for designing accessible applications, see the MSDN Library and the Microsoft Accessibility Web site http://www.microsoft.com/enable/.

Some examples of the guidelines include:

- The application must be compatible with specific system color, size, font, sound, and input settings.

  This provides a consistent user interface (UI) across all applications on the user's system. Users can configure their preferred settings for these elements by using Control Panel.
- Applications must be compatible with the High Contrast display option.

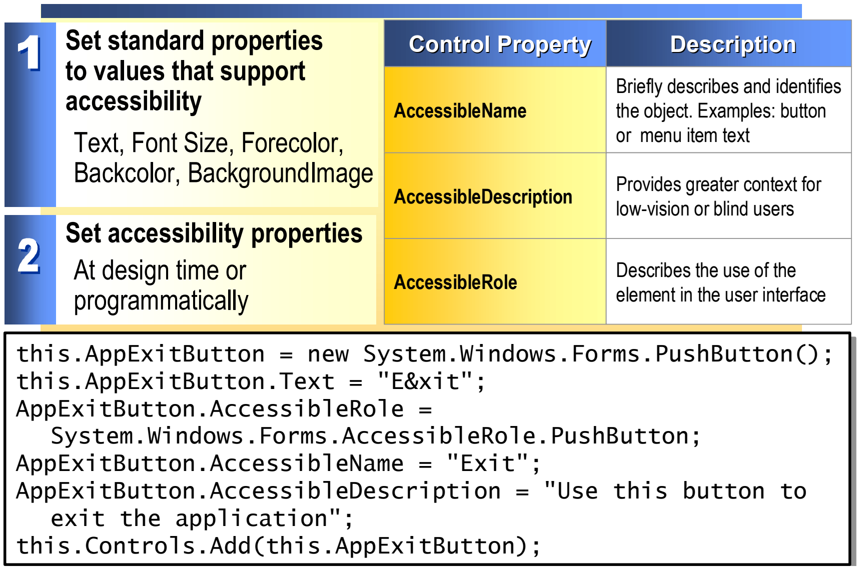
  Users who desire a high degree of screen legibility select the **Use High Contrast** check box on the **Display** tab of the **Accessibility Options** dialog box. When this option is selected, several restrictions are imposed on the application. For example, only system colors that can be selected through Control Panel, or colors set by the user, may be used by the application.
- The application must provide keyboard access to all features.

  This allows the user to interact with the application without requiring a pointing device, such as a mouse.
- Applications must not convey information by sound alone.

  Applications that convey information by sound must provide additional options to express this information. This includes on-screen messages.

Many of the standard user interface elements also have properties that you can set either at design time or programmatically in your applications to provide information for use by accessibility aids.

# How to Make Forms and Controls Accessible



**1** **Set standard properties to values that support accessibility**

Text, Font Size, Forecolor, Backcolor, BackgroundImage

**2** **Set accessibility properties**

At design time or programmatically

| Control Property | Description |
|---|---|
| AccessibleName | Briefly describes and identifies the object. Examples: button or menu item text |
| AccessibleDescription | Provides greater context for low-vision or blind users |
| AccessibleRole | Describes the use of the element in the user interface |

```
this.AppExitButton = new System.Windows.Forms.PushButton();
this.AppExitButton.Text = "E&xit";
AppExitButton.AccessibleRole =
   System.Windows.Forms.AccessibleRole.PushButton;
AppExitButton.AccessibleName = "Exit";
AppExitButton.AccessibleDescription = "Use this button to
   exit the application";
this.Controls.Add(this.AppExitButton);
```

## Introduction

The settings of various properties of a control or form can affect how accessible your application will be. Some properties are specifically designed for use by accessibility aids while others are more general properties like **FontSize**.

## Properties related to accessibility

This table provides accessibility considerations related to certain properties.

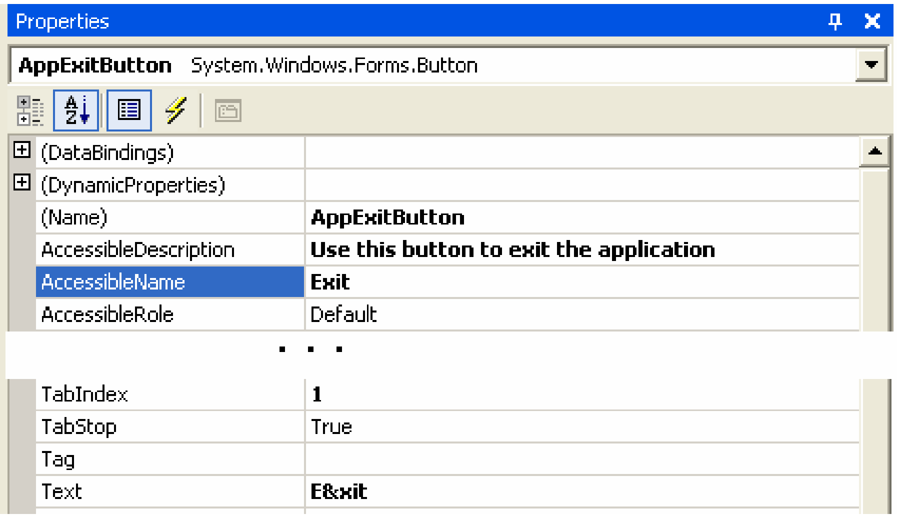
| Property | Considerations for accessibility |
|---|---|
| AccessibleName | This control property briefly describes and identifies the object, such as the text in a button, the name of a menu item, or a label displayed next to a text box control. Accessibility aids read the value of this property and report it to the user. For example, this text is read by Narrator and provided to the user audibly. |
| AccessibleDescription | This control property provides greater context beyond the **AccessibleName** property for low-vision or blind users. You can use this property to describe the visual appearance of the object such as buttons that contain graphics rather than text. Narrator does not read the value of this property, but other accessibility aids may. |
| AccessibleRole | This control property describes the use of the element in the user interface. For example, button, text box, animation, dialog box, Help balloon, list, and so forth. Leave this set to **(default)** unless you need to provide a more specific description for a custom or composite control. |
| TabIndex | Creates a sensible navigational path through the form. It is important for controls without intrinsic labels, such as text boxes, to have their associated label immediately precede them in the tab order. |

***(continued)***

| Property | Considerations for accessibility |
|---|---|
| Text | Use the ampersand (&) character to create access keys. Using access keys is part of providing documented keyboard access to features. |
| FontSize | If the font size is not adjustable, then it should be set to 10 points or larger. After the form's font size is set, all the controls added to the form thereafter will have the same size. |
| ForeColor | If this property is set to the default, then the user's color preferences will be used on the form. |
| BackColor | If this property is set to the default, then the user's color preferences will be used on the form. |
| BackgroundImage | Leave this property blank to make text more readable. |

### Setting accessibility properties at design time

At design time, use the Properties window to set the appropriate values for properties for a form or control.



Properties

**AppExitButton** System.Windows.Forms.Button

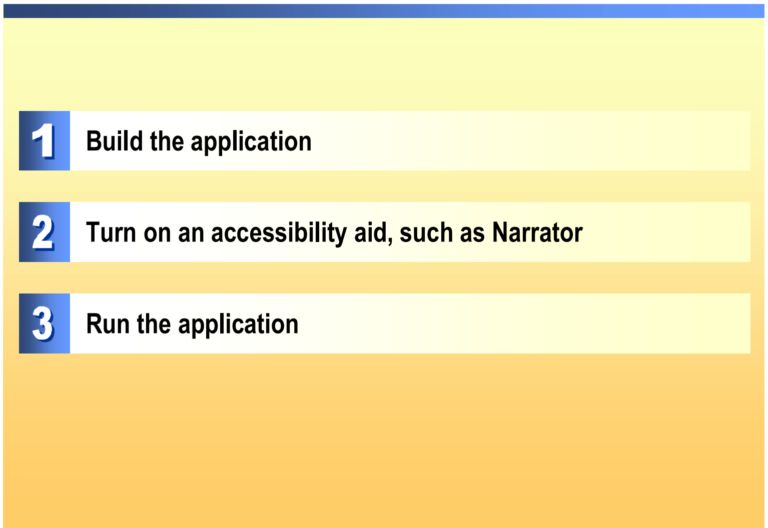
| | |
|---|---|
| ⊞ (DataBindings) | |
| ⊞ (DynamicProperties) | |
| (Name) | **AppExitButton** |
| AccessibleDescription | **Use this button to exit the application** |
| AccessibleName | **Exit** |
| AccessibleRole | Default |
| . . . | |
| TabIndex | **1** |
| TabStop | True |
| Tag | |
| Text | **E&xit** |

### Programmatically setting accessibility properties

You can also set the property values programmatically. In the following code, a button that has the appropriate property values is created and added to a form. The **this** keyword in the code refers to the form to which the button is added.

```
this.AppExitButton = new System.Windows.Forms.PushButton();
this.AppExitButton.Text = "E&xit";
AppExitButton.AccessibleRole =
System.Windows.Forms.AccessibleRole.PushButton;
AppExitButton.AccessibleName = "Exit";
AppExitButton.AccessibleDescription = "Use this button to exit
the application";
this.Controls.Add(this.AppExitButton);
```

# How to Test Accessibility



1 Build the application

2 Turn on an accessibility aid, such as Narrator

3 Run the application

**Introduction**

After you set the appropriate property values for your user interface elements to support accessibility and build your application, you can use one of the built-in accessibility aids, such as Narrator, to test the application.

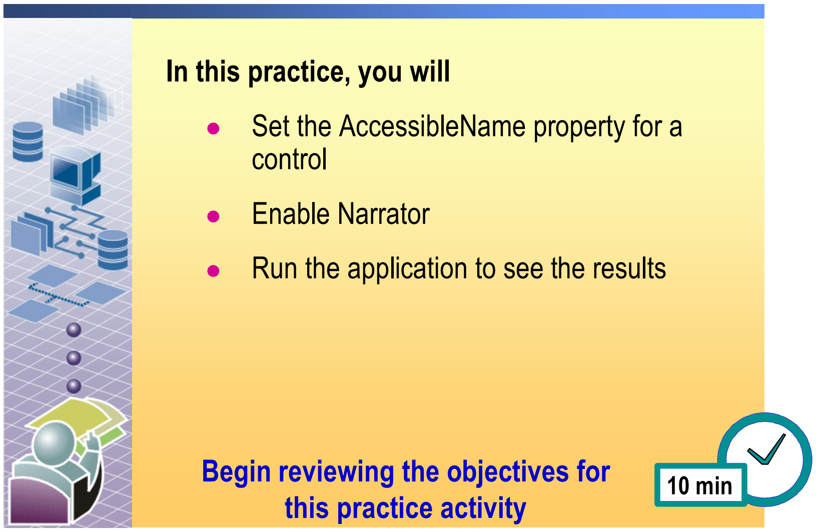
**Procedure: Testing the application with Narrator**

To test the accessibility features that you added to your application:

1. If you have not already done so, build your application.
2. Set the volume on your computer to a level that is comfortable for you.
3. On the **Start** menu, point to **All Programs**, point to **Accessories**, point to **Accessibility**, and then click **Narrator**.
4. Start your application, and navigate the user interface.
5. Notice how Narrator uses the accessibility property information to describe the user interface and your interaction with the application.

**Note** Notice that Narrator reads the information for one of the controls twice. This is because Narrator automatically reads the information from the control that has focus, then reads the information about the form, and finally reads the information about all the controls that appear on the form.

# Practice: Adding Accessibility Support to an Application



**In this practice, you will**

- Set the AccessibleName property for a control
- Enable Narrator
- Run the the application to see the results

Begin reviewing the objectives for this practice activity

### Introduction

In this practice, you will set the **AccessibleName** property for a control in the UsabilityDemo project, enable Narrator, and run the UsabilityDemo application to see the results.

### Instructions

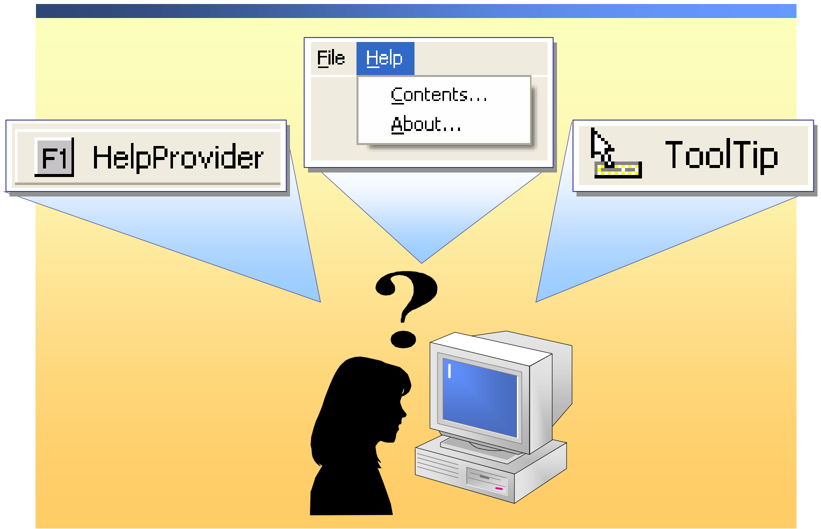
► **Add accessibility support to an application**

1. In Microsoft Visual Studio® .NET, open the UsabilityDemo project in *install_folder*\Practices\Mod08\Mod08_01\Starter.

> **Note** If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

2. Open the UsabilityDemo form.
3. Click the **Show Date/Time** button.
4. Set the **AccessibleName** property to **Shows the computer's current date and time**.
5. Make sure that the **AccessibleRole** property is set to **Default**.
6. Save your changes, and rebuild the project.
7. Set the volume on your computer to a level that is comfortable for you.
8. Start Narrator. On the **Start** menu, point to **All Programs**, point to **Accessories**, point to **Accessibility**, and then click **Narrator**.
9. Run the UsabilityDemo application either from the command prompt or from Windows Explorer.
10. As the application runs, listen to Narrator describe the UsabilityDemo form and the **Show Date/Time** button.

# Lesson: Adding Help to an Application



**Introduction**

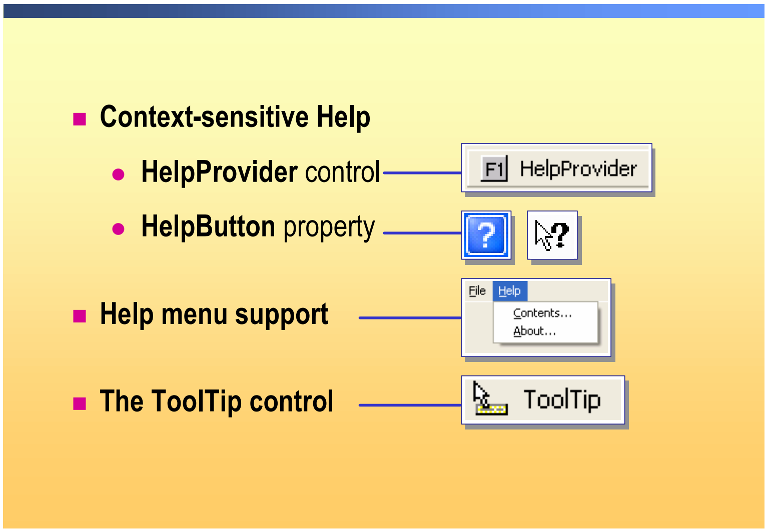
Adding Help to your application will make it easier for all users to learn how to use your application more quickly and more efficiently.

**Lesson objectives**

After completing this lesson, you will be able to:

- Describe the Help support available in the .NET Framework.
- Add Help to an application by adding context-sensitive Help, Help menus, and ToolTips.

# Help in the .NET Framework



- **Context-sensitive Help**
  - **HelpProvider** control
  - **HelpButton** property
- **Help menu support**
- **The ToolTip control**

*****************************ILLEGAL FOR NON-TRAINER USE******************************

### Introduction

The .NET Framework provides support for adding Help to your application by using three main areas:

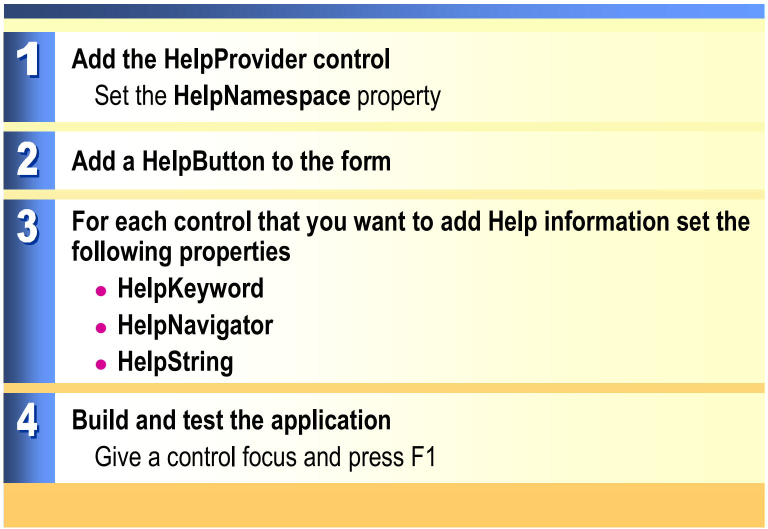
- Context-sensitive Help
- Help menus
- ToolTips

### Adding support for Help at design time

Visual Studio .NET provides several new controls that you can use when developing applications. To provide context-sensitive Help, you can use the **HelpProvider** control and the **HelpButton** property. To create a short Help text string that will appear when a user rests the mouse on a control, you can use the **ToolTip** control. The **HelpProvide**r and **ToolTip** controls do not appear on the form. They provide an interface between the controls and the properties that you can set to provide Help and ToolTips.

You can use the **MainMenu** control to add a Help menu to the application to provide easy access to Help topics.

# How to Add Context-Sensitive Help for Forms and Controls



1 **Add the HelpProvider control**

Set the **HelpNamespace** property

2 **Add a HelpButton to the form**

3 **For each control that you want to add Help information set the following properties**

- **HelpKeyword**
- **HelpNavigator**
- **HelpString**

4 **Build and test the application**

Give a control focus and press F1

### Introduction

Context-sensitive Help is the information that is displayed about the user interface element that has focus when the user presses F1 or clicks the **Help** button on a form and then clicks a user interface element.

Adding context-sensitive Help for forms and controls involves several steps.

### Procedure: Adding the HelpProvider control

The **HelpProvider** control enables Help in an application.

1. In Visual Studio .NET Design view, use the Toolbox to add a **HelpProvider** control to a form.
2. Associate help text with each control.
   - If you do not set the **HelpNamespace** property of the **HelpProvider** control, then information in the **HelpString** property for the control with focus displays in a small pop-up window near the control.
   - If you do set the **HelpNamespace** property of the **HelpProvider** control to a URL or a file, then when the user presses F1 at run time, the information from the URL or file specified in the **HelpNamespace** property displays.

### Procedure: Adding a Help button to the form

When the user clicks the **Help** button on the form at run time and then clicks a control, the text in the **HelpString** property for that control displays.

1. In Visual Studio .NET Design view, set the following properties for the form to which you want to add a **Help** button.
2. Set the **HelpButton** property to **True**.
3. Set the **MaximizeBox** property to **False**.
4. Set the **MinimizeBox** property to **False**.

### Set the Help information for each control

In Visual Studio .NET Design view, set the **HelpKeyword**, **HelpNavigator**, and **HelpString** properties for each control for which you want to add Help information.

If you set the **HelpNamespace** property in the **HelpProvider** control to an HTML page and you set the **HelpKeyword** property of a control, the Help system will try to locate an anchor tag in the Help file that has the keyword in it when the F1 key is pressed. For example, if the keyword is **FormHelp**, the Help system will look for <A NAME="FormHelp"> in the target HTML file. Because F1 is now overridden by setting the **HelpNamespace** property to bring up a separate Help file, the **Help** button can be used to mimic the F1 functionality.

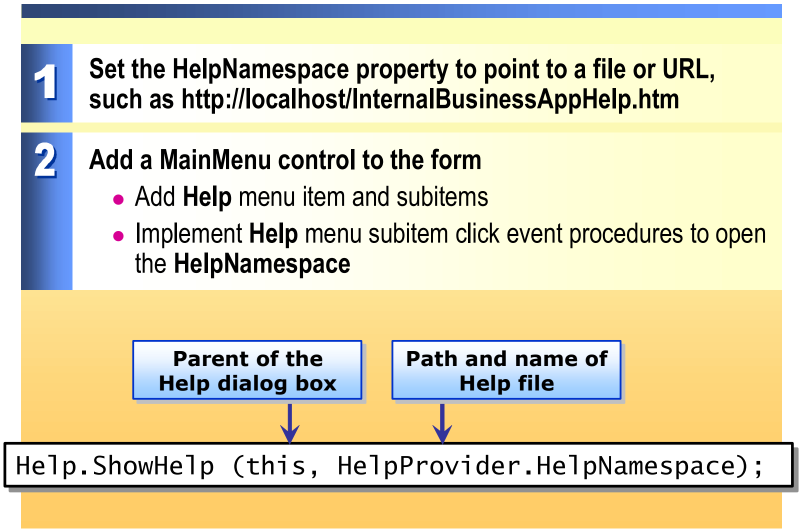
The **HelpNavigator** property determines how the keyword is used by the Help system to locate the information the user has requested. This property can take one of the following values.

| HelpNavigator property value | Description |
|---|---|
| AssociateIndex | Specifies that the index for a specified topic is performed in the specified URL. |
| Find | The search page for the specified URL is displayed. |
| Index | The index page for the specified URL is displayed. |
| TableOfContents | The table of contents page for the specified URL is displayed. |
| KeywordIndex | Specifies a keyword search for and the action to take in the specified URL. |
| Topic | The topic referenced by the specified URL is displayed. |

The **HelpString** property is used for pop-up Help (the Help that is displayed when a user presses F1 when a specific control has focus). If the **HelpNamespace** property on the **HelpProvider** is set, pressing F1 on a Help-enabled control will always bring up the target Help file rather than the pop-up window.

**Note** If you set the **HelpButton** property on a form to **True** and do not set the **MaximizeBox** and **MinimizeBox** properties to **False**, then the form will display the maximize box and minimize box, but it will not display the Help button.

# How to Link Help Topics to a Menu



1 Set the HelpNamespace property to point to a file or URL, such as http://localhost/InternalBusinessAppHelp.htm

2 Add a MainMenu control to the form

- Add **Help** menu item and subitems
- Implement **Help** menu subitem click event procedures to open the **HelpNamespace**

Parent of the Help dialog box

Path and name of Help file

```
Help.ShowHelp (this, HelpProvider.HelpNamespace);
```

*****************************ILLEGAL FOR NON-TRAINER USE******************************

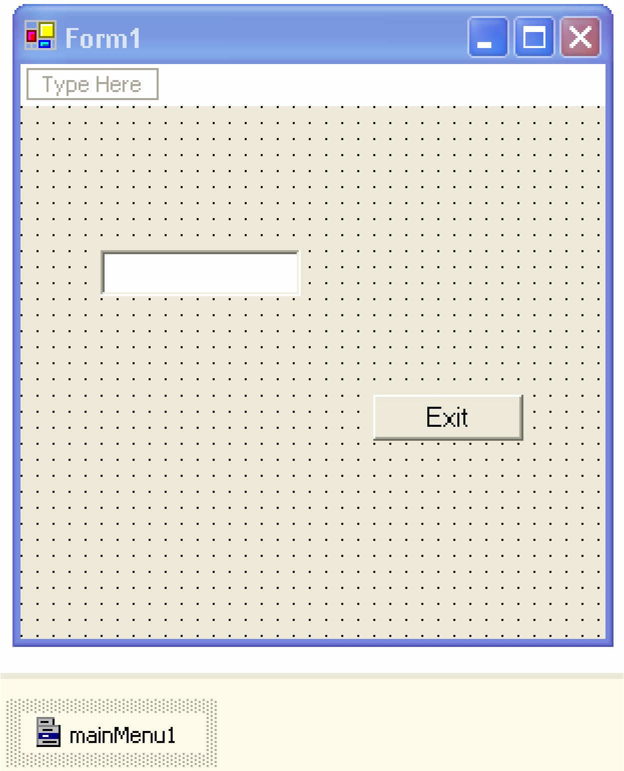
### Introduction

You can use the Visual Studio .NET **MainMenu** control to add a menu and submenus to your application.

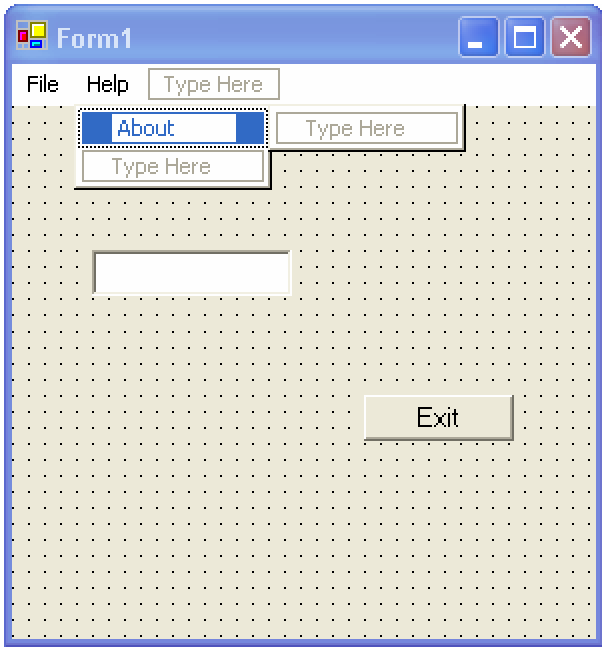
### Procedure: Linking Help topics to a menu

To link Help topics to a menu:

1. In Visual Studio .NET Design view, add the **MainMenu** control to the form.



2. Edit the menu to add the **Help** menu and **Help** menu items.



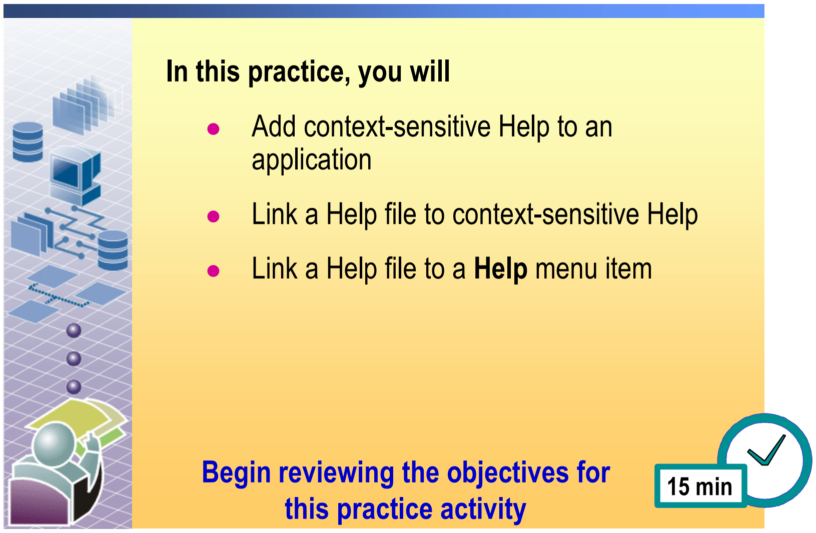
3. When the **Help** menu option is clicked at run time, it fires a click event. In the click event code for the menu item, add code to open the Help file:

```
Help.ShowHelp (this, HelpProvider.HelpNamespace);
```

In the previous code:

- The first parameter (**this**) is the control that identifies the parent of the **Help** dialog box. In this case, the form is the parent control of the **Help** dialog box.
- The second parameter (**HelpProvider.HelpNamespace**) is the path and name of the Help file. The URL can be of the form C:\..., file:///…, or http://....

# Practice: Adding Help to an Application



In this practice, you will

- Add context-sensitive Help to an application
- Link a Help file to context-sensitive Help
- Link a Help file to a **Help** menu item

Begin reviewing the objectives for this practice activity

15 min

**Introduction**

In this practice, you will add the **HelpProvider** control to the application and set the **HelpString** property of the **Show Date/Time** button to support context-sensitive Help. Then, you will set the **HelpNamespace** property on the **HelpProvider** control and the **HelpKeyword** and **HelpNavigator** properties of the **Show Date/Time** button to support access to Help files. You will also add the **Help** button to the form and complete the code for the **Help** menu to display the Help file.

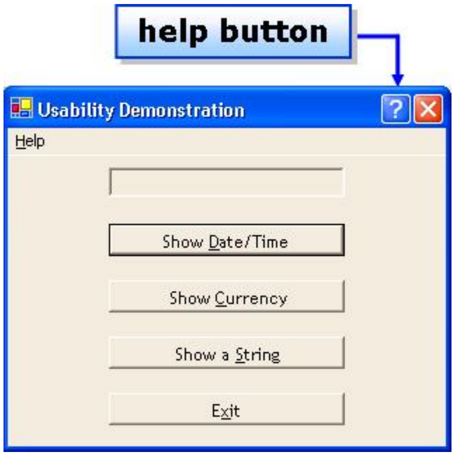
**Instructions**

► **Add context-sensitive Help to an application**

1. In Visual Studio .NET, open the UsabilityDemo.sln solution file in *install_folder*\Practices\Mod08\Mod08_02\Starter.
2. Open the UsabilityDemo.cs form in Design view.
3. On the **View** menu, click **ToolBox**.
4. In the **ToolBox**, double-click the **HelpProvider** control to add it to the form.
5. Change the **(Name)** property of the **HelpProvider** to **UsabilityDemoHelpProvider**.
6. Click the **Show Date/Time** button to display its properties.
7. Set the **HelpString on UsabilityDemoHelpProvider** property to **Shows the current date and time in the text box above**.
8. Build and run the application.
9. Tab through the controls until the **Show Date/Time** button has focus.
10. Press F1 to view the context-sensitive Help for the **Show Date/Time** button.

### ► Link to a Help file to provide context-sensitive Help

1. In the UsabilityDemo project, change the **HelpNamespace** property of the **UsabilityDemoHelpProvider** control to **http://localhost/UsabilityDemoHelpFile.htm**.
2. Click the **Show Date/Time** button to display its properties.
3. Set the **HelpKeyword on UsabilityDemoHelpProvider** property to **DateAndTime**.
4. Set the **HelpNavigator on UsabilityDemoHelpProvider** property to **Topic**.
5. Set the **HelpButton** property of the **UsabilityDemo** form to **True**.
6. Set the **MaximizeBox** property of the **UsabilityDemo** form to **False**.
7. Set the **MinimizeBox** property of the **UsabilityDemo** form to **False**.
8. Build and run the application.
9. Tab through the controls until the **Show Date/Time** button has focus.
10. Press F1 to view the topic in the Help file in Microsoft Internet Explorer.
11. Close Internet Explorer.
12. Click the help button in the upper right corner of the window to enable context-sensitive Help.



13. Click **Show Date/Time**.

    A small window will appear that explains what the **Show Date/Time** button does.

14. Close the application.

### ► Link Help to a Help menu in an application

1. In the UsabilityDemo.sln solution file, open the UsabilityDemo.cs form in Design view.
2. Click the **Help** menu to show the items in the menu.
3. Double-click the **Help** menu item in the **Help** menu to display the source code for the **Click** event for the menu item.
4. Show TODO comments in the Task List.

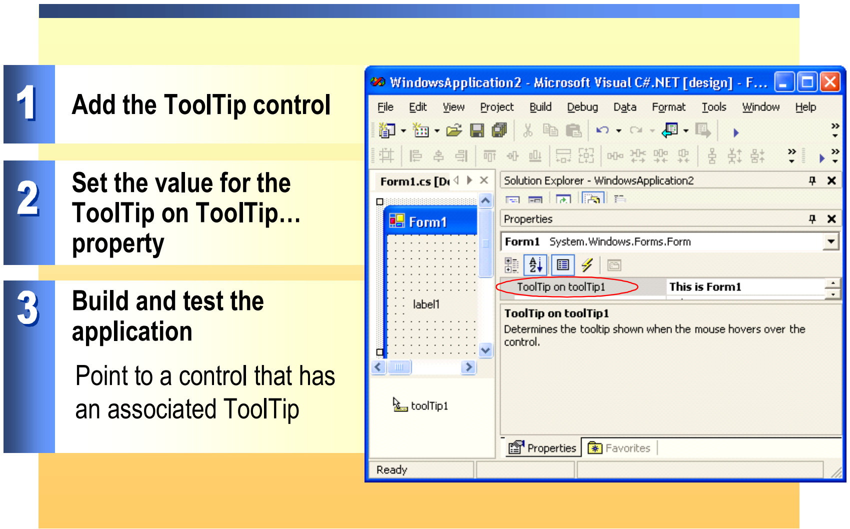
    To show TODO comments, on the **View** menu, point to **Show Tasks**, and then click **All**.

5. Locate the TODO comment in the **Click** event handler for the menu item, and add the following code.

   ```
   // TODO: Call the ShowHelp method to display the help file
   to the user.
   Help.ShowHelp(this,
   UsabilityDemoHelpProvider.HelpNamespace);
   ```
6. Build and run the application.
7. On the **Help** menu, click the **Help** menu item.

   Internet Explorer displays the Help file.

# How to Display Help with the ToolTip Control



1 Add the ToolTip control

2 Set the value for the ToolTip on ToolTip… property

3 Build and test the application

Point to a control that has an associated ToolTip

WindowsApplication2 - Microsoft Visual C#.NET [design] - F...
File Edit View Project Build Debug Data Format Tools Window Help
Form1.cs [De
Form1
label1
toolTip1
Solution Explorer - WindowsApplication2
Properties
Form1 System.Windows.Forms.Form
Properties
Favorites
Ready

ToolTip on toolTip1: This is Form1

**ToolTip on toolTip1**
Determines the tooltip shown when the mouse hovers over the control.

**************************ILLEGAL FOR NON-TRAINER USE******************************

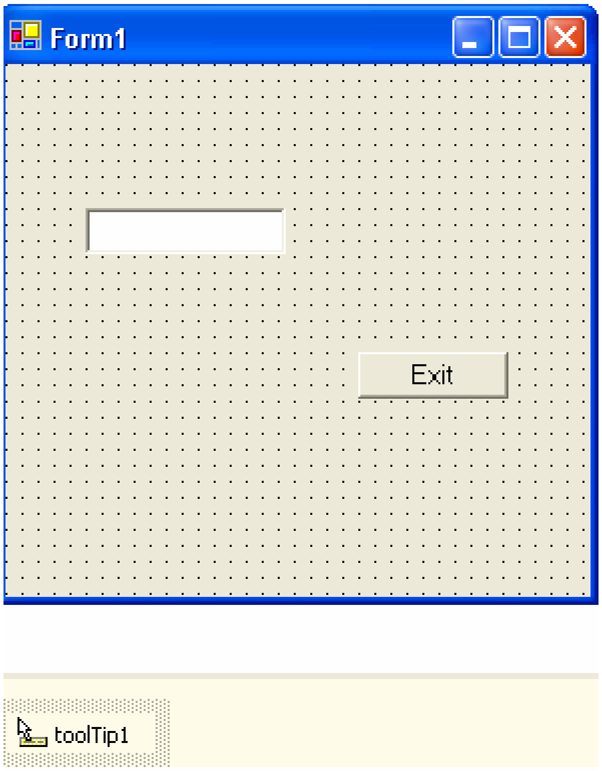
## Introduction

To enable ToolTips in an application, you must add a **ToolTip** control to each form.

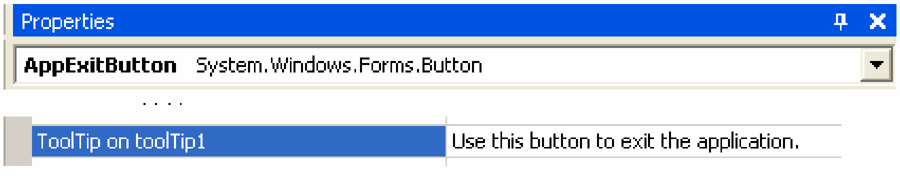
## Procedure: Adding ToolTips to an application

To add ToolTips to an application:

1. In Visual Studio .NET Design view, add a **ToolTip** control to the form.



2. Set the appropriate value for the ToolTip in the **ToolTipControlName** Properties window for each control on the form for which you want a ToolTip to display.



AppExitButton System.Windows.Forms.Button

ToolTip on toolTip1 Use this button to exit the application.

When the user moves the mouse pointer over the control at run time, after a short pause the ToolTip window appears, displaying the text that you set as the value for the **ToolTip on toolTip1** property.

# Practice: Adding ToolTips to an Application

In this practice, you will

- Add the **ToolTip** control to an application
- Set the **ToolTip** property for a control

Begin reviewing the objectives for this practice activity

10 min

## Introduction

In this practice, you will add a **ToolTip** control to the form and set the **ToolTip on UsabilityToolTip** property of the **Show Date/Time** button to a short string. You will then run the application to test whether the ToolTip appears when you point to the button by using the mouse pointer.

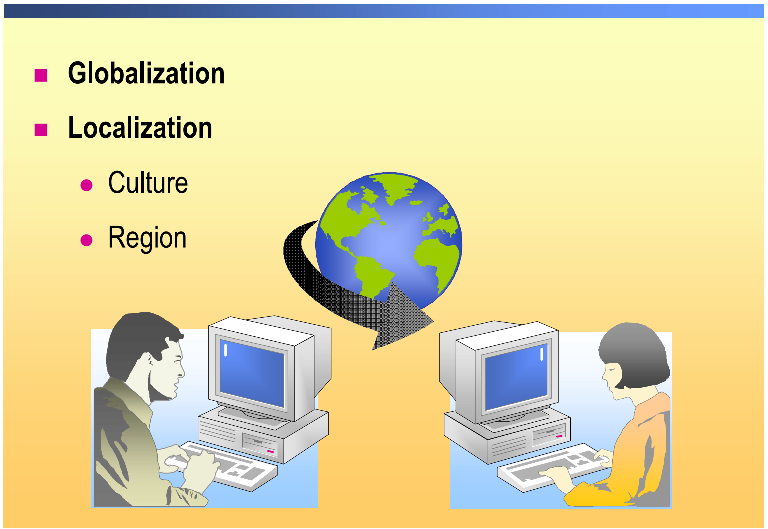
## Instructions

### ► Add ToolTips to an application

1. In Visual Studio .NET, open the UsabilityDemo.sln solution file in *install_folder*\Practices\ Mod08\Mod08_03\Starter.
2. Open the UsabilityDemo form.
3. On the **View** menu, click **ToolBox**.
4. In the ToolBox, double-click the **ToolTip** control to add it to the form.
5. Change the **(Name)** property of the **ToolTip** control to **UsabilityDemoToolTip**.
6. Click the **Show Date/Time** button to display its properties.
7. Set the **ToolTip on UsabilityToolTip** property of the button to **Displays the current date and time**.
8. Build and run the application.
9. Move the mouse pointer over the **Show Date/Time** button.

   A ToolTip window appears and shows the text that you added.

# Lesson: Localizing an Application



- Globalization
- Localization
  - Culture
  - Region

### Introduction

When you want to develop applications for use by people who speak various languages and come from various cultures, there are two aspects to consider:

- Globalization

  You should build the core functionality of the application without sources that will be localized. By doing this, you are designing applications that can be adapted to different cultures.

- Localization

  You then localize an application for specific cultures and regions by creating translations of resources. *Culture* is "who you are," *region* is "where you are." For example, you can localize an application into English for various regions, such as the United States of America (en-US) and the United Kingdom (en-GB).

  Within culture there are two concepts: current *culture* and current *uiculture*. Culture is for formatting numerics and datetime; uiculture is for retrieving resources by the correct localized language.

### Localization support in the .NET Framework

The **CultureInfo** class contains culture-specific information, such as the language, country/region, calendar, and cultural conventions associated with a specific culture. This class also provides the information required for performing culture-specific operations, such as casing, formatting dates and numbers, and comparing strings.

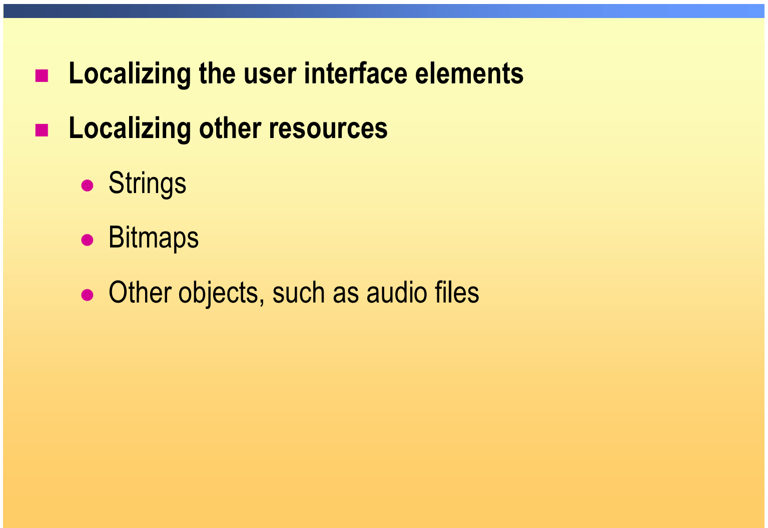
The **CultureInfo** class specifies a unique name for each culture. The name is a combination of a two-letter lowercase culture code associated with a language and, if required, a two-letter uppercase subculture code associated with a country or region. For a list of all the valid culture and region codes, see the topic "CultureInfo Class" in the .NET Framework software development kit (SDK) documentation.

**Lesson objectives**

After completing this lesson, you will be able to:

- Differentiate between the concepts of globalization and localization of an application.
- Use localization properties and resource files to create a localized version of an application.

# Localization in the .NET Framework



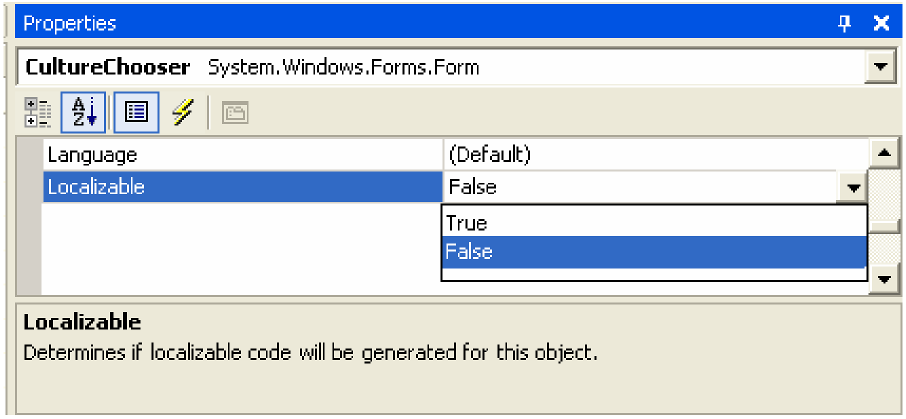
- Localizing the user interface elements
- Localizing other resources
  - Strings
  - Bitmaps
  - Other objects, such as audio files

**Introduction**

Visual Studio .NET provides good support for developers who want to create localized versions of their applications.

**Localizing user interface elements**

You can set properties on forms and controls to create multiple language versions of the user interface of your application.



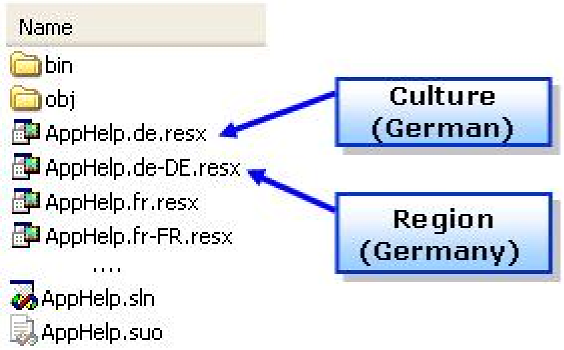
| | |
|---|---|
| Language | (Default) |
| Localizable | False |

True
False

**Localizable**
Determines if localizable code will be generated for this object.

Visual Studio .NET creates and uses resource files to store the information created for each form and control for the different cultures and regions.

When you create multiple language versions of a form, Visual Studio .NET automatically creates resource files to store the localization information. There are two resource files for each language: one for the specific culture and region and one for the overall culture. You can view these files by using Windows Explorer to the view folder for your project.

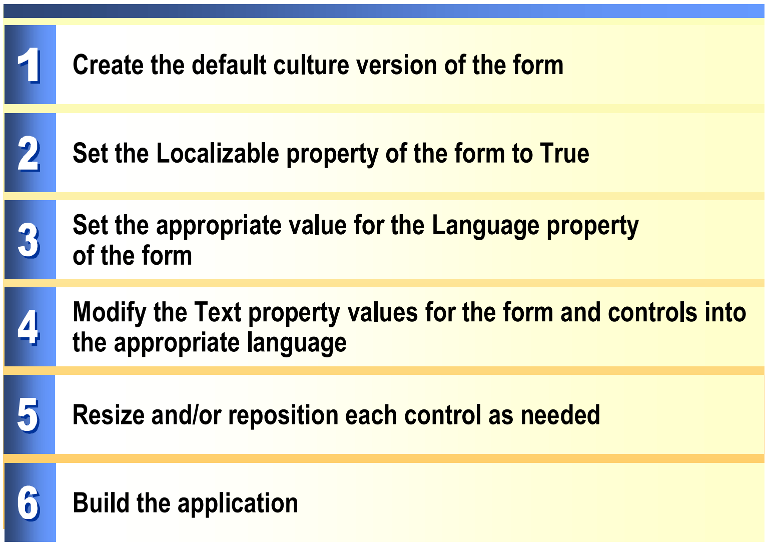


```
Name
bin
obj
AppHelp.de.resx
AppHelp.de-DE.resx
AppHelp.fr.resx
AppHelp.fr-FR.resx
        ....
AppHelp.sln
AppHelp.suo
```

## Localizing other resources

When you localize other resources, such as text strings, bitmaps, and so on, you must create resource files manually. To do this, add a new resource file to the project, name it appropriately, and then use the **ResourceManager** object in the .NET Framework to retrieve the appropriate resources at run time.

# How to Set Localization Properties



1 Create the default culture version of the form

2 Set the Localizable property of the form to True

3 Set the appropriate value for the Language property of the form

4 Modify the Text property values for the form and controls into the appropriate language

5 Resize and/or reposition each control as needed

6 Build the application

### Introduction

Before you localize your application, you must create the initial user interface. Then, test the interface thoroughly for functionality: button clicks, hiding and showing controls, enabling and disabling controls, and so forth. This is the default culture version of the user interface for your application. Set the **Language** property to **(default)** and the **Localizable** property to **False**.

After you have created and tested the UI in the default language, you can then create the other localized versions that your organization's business needs require by setting localization properties and providing new values for the text or other values of the controls.

### Procedure: Setting localization properties

To set localization properties on a form:

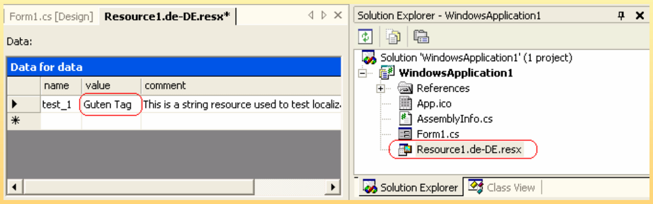
1. In Visual Studio .NET Design view, select the form that you want to localize.
2. Set the **Localizable** property of the form to **True**.
3. Set the **Language** property of the form to the language into which you want to localize the form.
4. Set the **Text** properties for each control with the correction for that culture.
5. Resize and/or reposition each control, as needed, to fit the localized text.
6. Build and test the application in the appropriate culture.

Each time that you choose another culture to localize to, Visual Studio .NET creates a new resource file for that culture. The data stored in the resource file is XML and includes not only the localized data but also changes made to control positioning and size.

# How to Create Localized Resource Files

1 Open an existing project and add an assembly resource file for the appropriate culture

2 Add entries to the resource file with values in the appropriate language for the culture



| | name | value | comment |
|---|---|---|---|
| ▶ | test_1 | Guten Tag | This is a string resource used to test localiz. |
| * | | | |

3 Save the file

4 Build the application

*****************************ILLEGAL FOR NON-TRAINER USE******************************

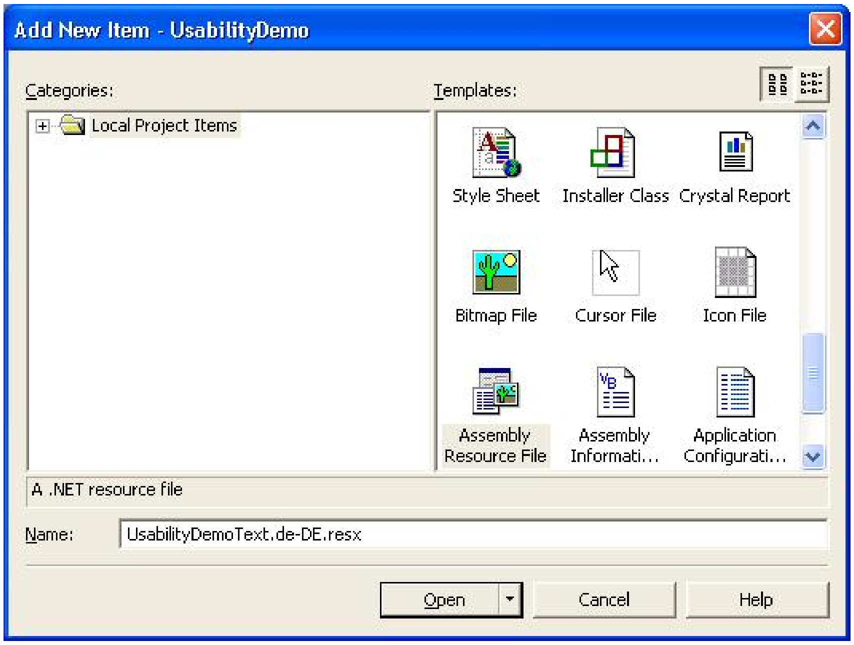
### Introduction

You can localize text strings associated with your application, such as informational or error messages, by using separate resource files for the various culture and region versions of your application.

### Procedure: Creating a localized resource file

To create a localized resource file:

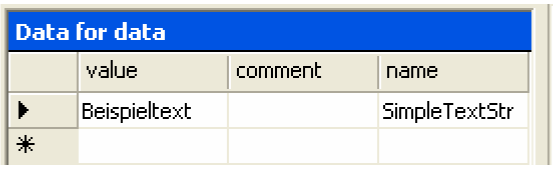
1. In Visual Studio .NET, open the existing project for the application to which you want to add localized strings.
2. On the **Project** menu, click **Add New Item**, and then select the **Assembly Resource File** icon in the Templates pane. Give the file a name that includes the appropriate culture identifier: for example, UsabilityDemoText.de-DE.resx.



A .NET resource file

Name: UsabilityDemoText.de-DE.resx

3. Add entries into the resource file, with values in the appropriate language for the culture.

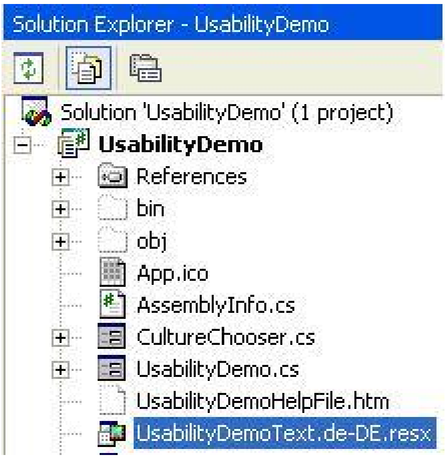


Data for data

| | value | comment | name |
|---|---|---|---|
| ▶ | Beispieltext | | SimpleTextStr |
| * | | | |

4. Save the file.

   When you create resources files in this manner, they will be listed in the Visual Studio .NET Solution window as part of the project.



Solution Explorer - UsabilityDemo

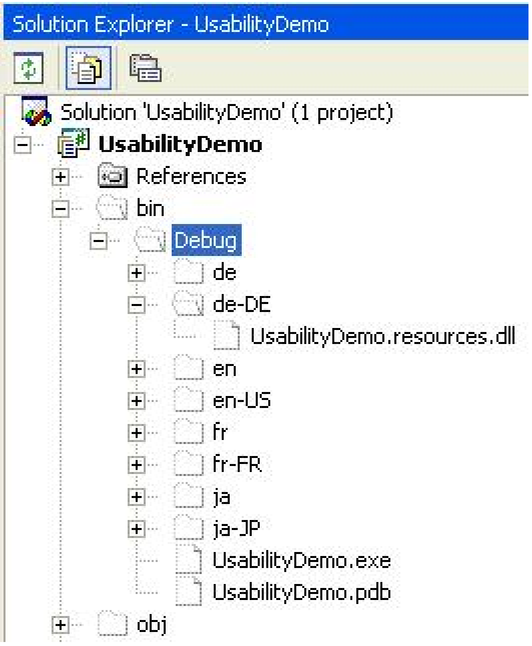
```
Solution 'UsabilityDemo' (1 project)
⊟ UsabilityDemo
    ⊞ References
    ⊞ bin
    ⊞ obj
      App.ico
      AssemblyInfo.cs
    ⊞ CultureChooser.cs
    ⊞ UsabilityDemo.cs
      UsabilityDemoHelpFile.htm
      UsabilityDemoText.de-DE.resx
```

5. Build the application.

   The compiler automatically creates a compiled version (or *satellite assembly*) for each culture.

### Using satellite assembly files

A *satellite assembly* is an assembly that contains only resources. After you build an application that contains multiple resource files for multiple cultures, the resulting satellite assembly names are all the same; however, they are stored in different folders that are named after the culture that they represent. For example, if you build an application called AppHelp with the resource file mentioned in the previous procedure, the resulting satellite assembly would be named UsabilityDemo.resources.dll, and it would be stored in *ApplicationFolder*\bin\Debug\*culture* (for example, UsabilityDemo\bin\Debug\de-DE).

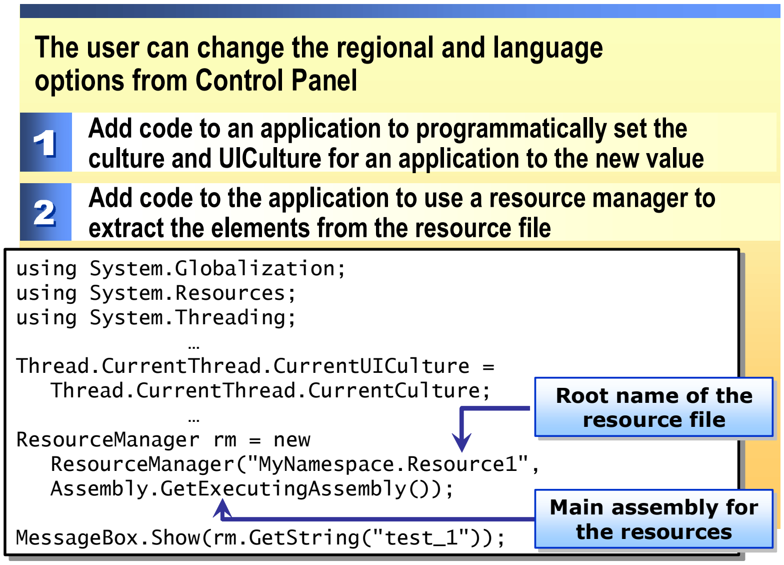

Solution Explorer - UsabilityDemo
Solution 'UsabilityDemo' (1 project)
UsabilityDemo
References
bin
Debug
de
de-DE
UsabilityDemo.resources.dll
en
en-US
fr
fr-FR
ja
ja-JP
UsabilityDemo.exe
UsabilityDemo.pdb
obj

The best reason to use satellite assemblies is the separation of resources from the application. You can build a version of your application with the default culture and release it to the public. At a later time, you can deploy a satellite assembly for each culture that you support. When a culture for your application becomes available, you put it in a separate assembly and make it available to your clients (by using the network, a Web site, or some other media).

Third-party organizations can create resource files by using the Resource File Generator (ResGen.exe) tool that is included with the .NET Framework. You can use this tool to create resource files without having Visual Studio .NET installed. Source files for ResGen must be text files with a file extension of .txt or XML files with the file extension of .resx. ResGen compiles these files into resource files that have the file extension .resource.

For more information about how the .NET Framework searches for resource files, see Module 4, “Assembly Versioning and Satellite Assemblies,” in Course 2350A, *Securing and Deploying Microsoft .NET Assemblies* or see the .NET SDK documentation.

# How to Change the Locale



The user can change the regional and language options from Control Panel

1 Add code to an application to programmatically set the culture and UICulture for an application to the new value

2 Add code to the application to use a resource manager to extract the elements from the resource file

```
using System.Globalization;
using System.Resources;
using System.Threading;
              …
Thread.CurrentThread.CurrentUICulture =
   Thread.CurrentThread.CurrentCulture;
              …
ResourceManager rm = new
   ResourceManager("MyNamespace.Resource1",
   Assembly.GetExecutingAssembly());

MessageBox.Show(rm.GetString("test_1"));
```

Root name of the resource file

Main assembly for the resources

### Introduction

After you have resource files with localized strings for the various versions of your application, you can add the code to set the culture of your application to match changes in locale that the user has selected. Then, you can add code to use a resource manager to display strings as appropriate. The .NET Framework ensures that the correct resource file for the culture is used as the source for your text strings.

### Procedure: Changing the locale to match the Control Panel settings

To change the locale to match the Control Panel settings:

1. Add the following **using** directives, which are needed to work with culture information:

```
using System.Globalization;  // Defines culture-related
          // information, such as
          // the language, the country and
          // region, calendars in use, format
          // pattern for dates, currency, and
          // number, and sort order for strings.

using System.Resources;  // provides classes and interfaces
          // that allow developers to create, store and
          // manage various culture-specific resources
          // used in an application

using System.Threading;
```

2. Set the thread's User Interface culture based on what is set in Control Panel.

```
Thread.CurrentThread.CurrentUICulture =
Thread.CurrentThread.CurrentCulture;
```

In this case, the **Thread.CurrentThread.CurrentCulture** property value is taken from the settings in Control Panel. To make the UI use the correct culture, you set the **CurrentUICulture** property value of the current thread.

**Note** If you want to ensure that your Windows Forms application starts with the culture that is set in the **Regional and Language Options** dialog box in Control Panel, set the **Thread.CurrentThread.CurrentUICulture** property value prior to the call to the **InitializeComponents** method in the startup class constructor.

The resource manager uses the resource fallback process to control how an application retrieves resources. The resource manager uses the **CultureInfo.CurrentUICulture** property to determine which resources to search for and load. If you want to use a different culture than that of the local computer, then you must set this property programmatically in your application. For example, if you set the **Culture.CurrentUICulture** property to **en-GB**, the runtime searches for a satellite assembly that supports the **en-GB** culture. If the **Culture.CurrentUICulture** is not set, the common language runtime sets the culture to that of the local computer.

Because the runtime always searches for the default culture when all other possibilities are exhausted, you should always include a default culture with your applications. The default culture is the only one that is built into the main application assembly. For more information about how the .NET Framework searches for resource files, see Module 4, “Assembly Versioning and Satellite Assemblies,” in Course 2350A, *Securing and Deploying Microsoft .NET Assemblies* or see the .NET SDK documentation.

### Procedure: Accessing localized strings by using a resource manager

To access localized strings by using a resource manager:

1. Add the code to instantiate a resource manager and access resources.

   ```
   ResourceManager rm = new
   ResourceManager("MyNamespace.Resource1",
   Assembly.GetExecutingAssembly());
   ```

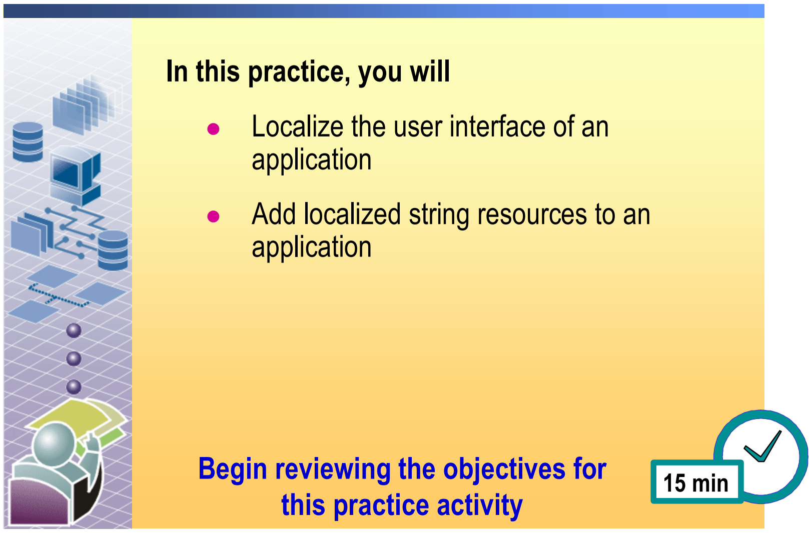
   The first parameter is the root name of the resource file. For example, if the resource file is Resource1.en-EN.resources, the root name is Resource1. The root name of the resource file is qualified with the namespace that contains the resource file. The second parameter is the main assembly for the resources. Because you are in the assembly that is the main assembly, the call to **GetExecutingAssembly()** returns the current assembly’s name.

2. Add the code to access the resource and display the localized string.

   ```
   MessageBox.Show (rm.GetString("test_1"));
   ```

   When you create an instance of the resource manager, it automatically gets the value for the current culture and locates the appropriate resource file. Call **GetString**, passing the name of a specific resource file element, and it will return the value from the correct resource file that matches the current culture setting. If the resource manager cannot locate the appropriate resource file, an exception is thrown. Therefore, any code that uses the resource manager should be wrapped in exception-handling code.

# Practice: Localizing an Application



In this practice, you will

- Localize the user interface of an application
- Add localized string resources to an application

Begin reviewing the objectives for this practice activity

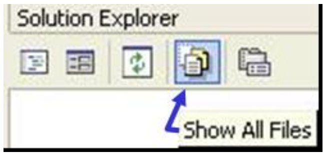
**Introduction**

In this practice, you will localize the user interface of an application and add localized string resources for an application.

**Instructions**

► **Localize a form in a Windows Forms application**

1. In Visual Studio .NET, open the UsabilityDemo.sln solution file in *install_folder*\Practices\ Mod08\Mod08_04\Starter.
2. Open the UsabilityDemo form.
3. Click the **Show All Files** button in the Solution Explorer window.



4. Examine the **Localizable** property of the form. Notice that it is set to **True**, which means that the form can be localized.
5. Set the **Language** property of the form to **French (France)**.

   The English version of the form is still displayed, but it can be localized into French. Notice that a set of new resource files for the form can be found under the UsabilityDemo source file in Solution Explorer. These are UsabilityDemo.fr.resx and UsabilityDemo.fr-FR.resx. Both of these files are required for the resource fallback process.
6. Notice that this form has also been localized into German and Japanese.
7. Set the **Text** property of the form to **Démonstration de l'utilisation**.

**Note** To get the strings for the French version of the form, go to the UsabilityDemoLocalizedStrings.rtf file in *install_folder*\Practices\ Mod08\Mod08_04.

8. Set the **Text** property of the menu and menu item to **Aide**.
9. Set the **Text** property of the **Choose a Culture** button to **Choisir une langue**.
10. Set the **Text** property of the **Show Date/Time** button to **Afficher la date/l'heure**.
11. Set the **Text** property of the **Show Currency** button to **Afficher la devise**.
12. Set the **Text** property of the **Show a String** button to **Afficher une chaîne**.
13. Set the **Text** property of the **Exit** button to **Quitter**.

### ► Add a string resource file to the application

1. In Solution Explorer, right-click the **UsabilityDemo** project, select **Add**, and choose **Add New Item**.
2. In the **Add New Item** window, select the **Assembly Resource File** icon in the **Templates** pane.
3. Set the name of the resource file to **UsabilityDemoText.fr-FR.resx**., and then click **Open**.
4. Set the **Name** of the resource string to **SimpleTextString**.
5. Set the **Value** of the resource string to **Voici du texte**.
6. Save and close the resource file.

### ► Add code to obtain resource strings

1. View the UsabilityDemo source file.
2. Show TODO comments in the Task List. To do this, click the **View** menu, point to **Show Tasks**, and then click **All**.
3. Find the first TODO comment in the code. Add the three **using** directives to enable support for localization in the application.

```
using System.Globalization;
using System.Resources;
using System.Threading;
```

4. Find the next TODO comment. Declare a **private** resource manager variable.

```
private ResourceManager RM;
```

5. Find the next TODO comment. Create an object instance of the resource manager. The code is added to the first constructor in the application.

```
RM = new ResourceManager("UsabilityDemo.UsabilityDemoText",
Assembly.GetExecutingAssembly());
```

6. Find the next TODO comment. Create an object instance of the resource manager. The code is added to the second constructor in the application.

```
RM = new ResourceManager("UsabilityDemo.UsabilityDemoText",
Assembly.GetExecutingAssembly());
```

7. Find the next TODO comment. Add code to use the resource manager to retrieve the text string from the resource file and display it in the text box.

   ```
   OutputTextBox.Text = RM.GetString("SimpleTextString");
   ```

8. Find the next TODO comment. Add code to set the current thread's **Culture** and **UICulture** property values to the same culture that the user requested.

   ```
   Thread.CurrentThread.CurrentUICulture = new
   CultureInfo(ChosenCulture, false);
   Thread.CurrentThread.CurrentCulture = new
   CultureInfo(ChosenCulture, false);
   ```

9. Save your work.

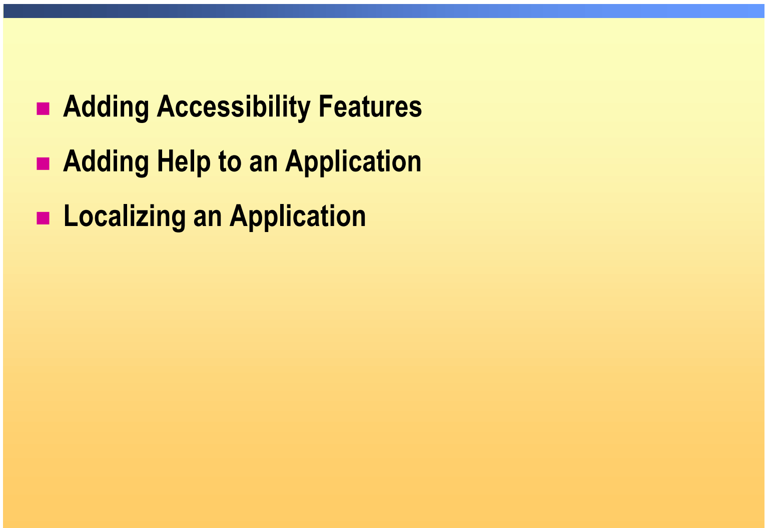
► **Test the application**

1. Build and run the application.
2. Notice that the only thing that you can do when the application starts is either choose a culture or exit the application. After you choose a culture for the application, the other buttons become available.
3. Click **Choose a Culture**.
4. Select one of the cultures.

   If you don't select any of them, the application uses English as the default.

5. When you click **OK** on the **CultureChooser** form, the **CultureChooser** form disappears and the **UsabilityDemo** form reappears in the correct culture.

# Review



- Adding Accessibility Features
- Adding Help to an Application
- Localizing an Application

****************************ILLEGAL FOR NON-TRAINER USE******************************

1. What is an accessibility aid? List the accessibility aids that are included with Windows XP.

   **An accessibility aid is an application that provides alternative input and output methods for people with disabilities, such as people who have hearing, sight, and mobility impairments. Windows XP includes several accessibility aids, including Narrator, Magnifier, and On-Screen Keyboard.**

2. List some guidelines that developers should consider with regard to accessibility when writing an application.

   **The application must be compatible with specific system color, size, font, sound, and input settings. Applications must be compatible with the High Contrast option. The application must provide keyboard access to all features. Applications should convey no information by sound alone.**

3. What control properties are available to enable accessibility of an application through an accessibility aid? Describe their uses.

   **AccessibleName, AccessibleDescription, and AccessibleRole. The AccessibleName property is used to briefly describe the control. For example, if the control is an OK button, the AccessibleName property would be set to something similar to OK. The AccessibleDescription property provides additional information to the user. This property is particularly useful when the control does not have any associated text, such as a button with a picture on it. AccessibleRole describes the role of the control. Typically, the AccessibleRole will be set to (default), but there are times when this must changed for custom and composite controls.**

4. Describe the purpose of the ampersand (&) character with regard to the text for controls.

   **The ampersand character, when included in the Text property of a control, provides an access key to the user so that he or she can use the keyboard—in addition to the mouse—to access the control.**

5. Describe techniques for adding Help to an application.

   **The developer can add Help by using the HelpProvider control, the ToolTip control, a Help menu, and the HelpButton property of a form.**

6. What is the purpose of the **HelpNamespace** property?

   **The HelpNamespace property of the HelpProvider control indicates where a Help file is located. When this property has a value, pressing F1 at run time loads the Help file rather than displaying a pop-up Help window.**

7. When enabling the **HelpButton** property, what must be done to ensure that the **Help** button appears on the form at run time?

   **The MaximizeBox and MinimizeBox properties must be set to False.**

8. Where do the **HelpProvider** and **ToolTip** controls appear when they are added to a form?

   **They appear below the form, not on the form.**

9. What should be localized when building a Windows application?

   **The user interface, strings, graphics, audio files, and any other resources that must be changed between cultures.**

10. What properties must be set to localize a form?

    **The Localizable property must be set the True to indicate that the form can be localized, and the Language property must be set for the appropriate culture and region.**

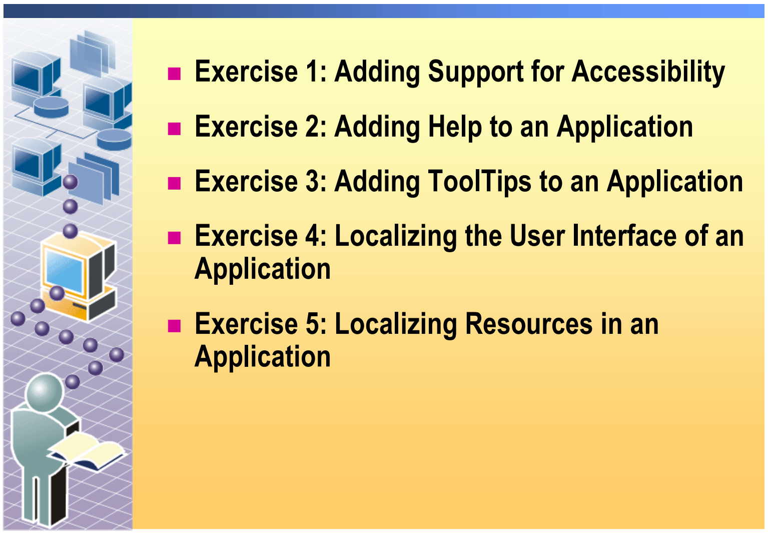
11. What is required to change the locale of an application at run time so that it matches the current settings of the computer?

    **A resource manager reads the localized information from the resource files, and the user interface culture must be changed to the current culture of the thread.**

12. Describe the purpose of resource files.

    **A resource file is used to store the localized versions of resources such as strings, bitmaps, and audio and video files that are not part of the user interface. The resource manager reads the values from a resource file based on the current culture setting in the application.**

# Lab 8.1: Enhancing the Usability of an Application



- Exercise 1: Adding Support for Accessibility
- Exercise 2: Adding Help to an Application
- Exercise 3: Adding ToolTips to an Application
- Exercise 4: Localizing the User Interface of an Application
- Exercise 5: Localizing Resources in an Application

**Objectives**

After completing this lab, you will have demonstrated your ability to:

- Enhance an application by adding accessibility features.
- Provide user Help and ToolTips in an application.
- Localize the user interface of an application.
- Create and access localized resource files.

**Note** This lab focuses on the concepts in Module 8, "Enhancing the Usability of Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. As a result, this lab may not comply with Microsoft security recommendations.

**Prerequisites**

Before working on this lab, you must have:

- The knowledge and skills to develop a simple Windows Forms application by using a Visual Studio .NET–compatible programming language.
- The knowledge and skills to modify the properties of controls.
- The knowledge and skills to add accessibility and Help features and to localize an application's user interface and resources by using Visual Studio .NET.

**Scenario**

The Internal Business Application shell provides a common access point to various internal business applications. The shell must provide accessibility support and Help to users in the organization. In addition, the application must be localized into several languages to support the languages spoken by people of different cultures across the organization.

Supporting accessibility requires that you set control properties and use an accessibility aid such as the Narrator utility included in Windows XP. Including Help as part of the application requires that you use the **HelpProvider** control, set several properties on controls, and add code to the application to provide access to a Help file. To localize the application for specific region and culture combinations requires that each user interface element (form, controls, and so on) and resources found in the source code be localized for the appropriate cultures. In addition, the use of a resource manager is required to retrieve localized resources from resource files.

In this lab, you will add some accessibility support and Help features to the Internal Business Application shell and localize the application for several region and culture combinations.

**Estimated time to complete this lab: 30 minutes**

# Exercise 1
# Adding Support for Accessibility

In this exercise, you will update the Internal Business Application shell by adding support for accessibility and testing the application by using the Narrator accessibility aid in Windows XP.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab08_1\Ex01\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab08_1\Ex01\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open the InternalBusinessApp project in Visual Studio .NET. Browse to *install_folder*\Labfiles\Lab08_1\Ex01\Starter to find the project. | **a.** For more information about opening a project file and starting an application, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. For additional information about starting an application in the Designer, in Index, search by using the phrase **Debugging Windows Applications**. |
| **2.** Open the AppControlForm form and set the form's **AccessibleName** property to **Internal Business Application Control Panel**. | **a.** For more information about how to add accessibility support to an application, see the following resources:<br>• Practice: Adding Accessibility Support to an Application, in Module 8, "Enhancing the Usability of Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Lesson: Adding Accessibility Features in Module 8, "Enhancing the Usability of Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>**b.** For more information about writing accessible applications, see the following resource:<br>• The Visual Studio.NET Help documentation. In Search, select the **Search in titles only** check box, then search by using the phrase **Designing Accessible Applications**. |

<table>
<tr><th>Tasks</th><th>Additional information</th></tr>
<tr><td>

**3.** Set the **AccessibleName** property for the buttons. See the following table for the values to give to the **AccessibleName** property for each button.

<table>
<tr><th>Button</th><th>Value</th></tr>
<tr><td>Make Travel Plans</td><td>Make Travel Plans</td></tr>
<tr><td>Expense Reporting</td><td>Expense Reporting</td></tr>
<tr><td>Procurement</td><td>Procurement</td></tr>
<tr><td>Exit</td><td>Exit the Application</td></tr>
</table>

</td><td>

**a.** For more information about how to add accessibility support to an application, see the following resources:

- Practice: Adding Accessibility Support to an Application, in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- Lesson: Adding Accessibility Features in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.

</td></tr>
<tr><td>

**4.** Enable Narrator.

</td><td>

**a.** For more information about how to enable Microsoft Narrator in Windows XP Professional, see the following:

- Practice: Adding Accessibility Support to an Application, in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- Lesson: Adding Accessibility Features in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.

</td></tr>
<tr><td>

**5.** Run the application to test the accessibility support. You can log on with the user name **mario** and the password **P@ssw0rd**.

</td><td>

**a.** For more information about starting an application in the Designer, see the following resource:

- The Visual Studio.NET Help documentation. In Search, select the **Search in titles only** check box, then search by using the phrase **Debugging Windows Applications**.

</td></tr>
</table>

# Exercise 2
# Adding Help to an Application

In this exercise, you will update the Internal Business Application shell by adding on-screen Help. When the user presses the F1 key, a Help message that is associated with the control that has focus appears.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab08_1\Ex02\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab08_1\Ex02\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open the InternalBusinessApp project in Visual Studio .NET. Browse to *install_folder*\Labfiles\Lab08_1\Ex02\Starter to find the project. | **a.** For more information about opening a project file and starting an application, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. For additional information about starting an application in the Designer, in Index, search by using the phrase **Debugging Windows Applications**. |
| **2.** View the Toolbox, and add the **HelpProvider** control to the AppControlForm form.<br>**a.** Set the **(Name)** property of the **HelpProvider** control to **InternalBusinessAppHelpProvider**.<br>**b.** Set the **HelpNamespace** property to **http://localhost/ InternalBusinessAppHelp.htm**. | **a.** For more information about adding Help to an application, see the following resources:<br>• Practice: Adding Help to an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C # .NET).*<br>• Lesson: Adding Help to an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*<br>• The .NET Framework SDK documentation. In Search, select the **Search in titles only** check box, then search by using the phrase **Introduction to the Windows Forms HelpProvider**. |

<table>
<tr><th>Tasks</th><th>Additional information</th></tr>
<tr>
<td>
<b>3.</b> Select the AppControlForm form. To enable Help on the form, set some of the Help properties. See the following table for the properties and the corresponding values to set.
<table>
<tr><th>Property</th><th>Value</th></tr>
<tr><td>HelpKeyword</td><td>IBA_ControlPanel</td></tr>
<tr><td>HelpNavigator</td><td>Topic</td></tr>
<tr><td>HelpString</td><td>Provides access to the company’s internal applications</td></tr>
</table>
</td>
<td>
<b>a.</b> For more information about adding Help to an application, see the following resources:
<ul>
<li>Practice: Adding Help to an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, <i>Developing Microsoft .NET Applications for Windows (Visual C# .NET)</i>.</li>
<li>Lesson: Adding Help to an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, <i>Developing Microsoft .NET Applications for Windows (Visual C# .NET)</i>.</li>
<li>The .NET Framework SDK documentation. In Search, select the <b>Search in titles only</b> check box, then search by using the phrase <b>Introduction to the Windows Forms HelpProvider</b>.</li>
</ul>
</td>
</tr>
</table>

<table>
<tr><th>Tasks</th><th>Additional information</th></tr>
<tr>
<td>
<b>4.</b> For each of the controls on the AppControlForm form, set the Help properties to enable Help. Use the following tables to set the appropriate properties for the following controls.<br>
<b>a.</b> <b>Make Travel Plans</b> control
<table>
<tr><th>Property</th><th>Value</th></tr>
<tr><td>HelpKeyword</td><td>IBA_Travel</td></tr>
<tr><td>HelpNavigator</td><td>Topic</td></tr>
<tr><td>HelpString</td><td>Plan a business trip.</td></tr>
</table>
<b>b.</b> <b>Expense Reporting</b> control
<table>
<tr><th>Property</th><th>Value</th></tr>
<tr><td>HelpKeyword</td><td>IBA_Expense</td></tr>
<tr><td>HelpNavigator</td><td>Topic</td></tr>
<tr><td>HelpString</td><td>Log expenses for reimbursement.</td></tr>
</table>
<b>c.</b> <b>Procurement</b> control
<table>
<tr><th>Property</th><th>Value</th></tr>
<tr><td>HelpKeyword</td><td>IBA_Procurement</td></tr>
<tr><td>HelpNavigator</td><td>Topic</td></tr>
<tr><td>HelpString</td><td>Internal Purchasing.</td></tr>
</table>
<b>d.</b> <b>Exit</b> control
<table>
<tr><th>Property</th><th>Value</th></tr>
<tr><td>HelpKeyword</td><td>IBA_Exit</td></tr>
<tr><td>HelpNavigator</td><td>Topic</td></tr>
<tr><td>HelpString</td><td>Exit the Internal Business Application.</td></tr>
</table>
<b>e.</b> <b>Status Bar</b> control
<table>
<tr><th>Property</th><th>Value</th></tr>
<tr><td>HelpKeyword</td><td>IBA_Status</td></tr>
<tr><td>HelpNavigator</td><td>Topic</td></tr>
<tr><td>HelpString</td><td>Network Connection Status</td></tr>
</table>
</td>
<td>
<b>a.</b> For more information about adding Help to an application, see the following resources:<br>
• Practice: Adding Help to an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, <i>Developing Microsoft .NET Applications for Windows (Visual C# .NET)</i>.<br>
• Lesson: Adding Help to an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, <i>Developing Microsoft .NET Applications for Windows (Visual C# .NET)</i>.<br>
• The .NET Framework SDK documentation. In Search, select the <b>Search in titles only</b> check box, then search by using the phrase <b>Introduction to the Windows Forms HelpProvider</b>.
</td>
</tr>
</table>

| Tasks | Additional information |
|---|---|
| **5.** Add a **Help** button to the AppControlForm form.<br>**a.** Set the value for the **MaximizeBox** and **MinimizeBox** properties to **False**. | **a.** For more information about adding Help to an application, see the following resources:<br>• Practice: Adding Help to an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Lesson: Adding Help to an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. In Search, select the **Search in titles only** check box, then search by using the phrase **Introduction to the Windows Forms HelpProvider**. |
| **6.** Link the **Help** menu to the Help file. | **a.** For more detailed information about the tasks that you must perform, see the TODO comments in the code.<br>**b.** For more information about adding Help to an application, see the following resources:<br>• Practice: Adding Help to an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Lesson: Adding Help to an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. In Search, select the **Search in titles only** check box, then search by using the phrase **Introduction to the Windows Forms HelpProvider**. |
| **7.** Run the application to test Help. | **a.** For more information about starting an application from within Designer, see the following resource:<br>• The Visual Studio.NET Help documentation. In Index, search by using the phrase **Debugging Windows Applications**. |

# Exercise 3
# Adding ToolTips to an Application

In this exercise, you will update the Internal Business Application shell by adding ToolTips to the AppControlForm form and providing ToolTip text for the controls on the form.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab08_1\Ex03\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab08_1\Ex03\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open the InternalBusinessApp project in Visual Studio .NET. Browse to *install_folder*\Labfiles\Lab08_1\Ex03\Starter to find the project. | **a.** For more information about opening a project file and starting an application, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. For additional information about starting an application in the Designer, in Index, search by using the phrase **Debugging Windows Applications**. |
| **2.** View the AppControlForm form in the form designer.<br>**a.** View the Toolbox, and add the **ToolTip** control to the AppControlForm form.<br>**b.** Set the **(Name)** property of the **ToolTip** control to **ApplicationToolTip**. | **a.** For more information about adding ToolTips to an application, see the following resource:<br>• Practice: Adding ToolTips to an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Lesson: Adding Help to an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. Search by using the phrase **ToolTip Component**. |

<table>
<tr><th>Tasks</th><th>Additional information</th></tr>
<tr>
<td>
**3.** Set the text for the ToolTip for each of the controls on the AppControlForm form.<br>
**a.** Use the following table to set the value for the **ToolTip on ApplicationToolTip** property for each control.
<table>
<tr><th>Property</th><th>Value</th></tr>
<tr><td>Form</td><td>Accesses internal applications.</td></tr>
<tr><td>Make Travel Plans</td><td>Allows you to plan a business trip.</td></tr>
<tr><td>Expense Reporting</td><td>Allows you to submit and check expense reports.</td></tr>
<tr><td>Procurement</td><td>Allows you to purchase items at company rates.</td></tr>
<tr><td>Exit</td><td>Exit the application.</td></tr>
<tr><td>Status bar</td><td>Displays the network connection status.</td></tr>
</table>
</td>
<td>
**a.** For more information about adding ToolTips to an application, see the following resources:<br>
• Practice: Adding ToolTips to an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>
• Lesson: Adding Help to an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>
• The .NET Framework SDK documentation. Search by using the phrase **ToolTip Component**.
</td>
</tr>
<tr>
<td>**4.** Run the application to test the ToolTips.</td>
<td>
**a.** For more information about starting an application in the Designer, see the following resource:<br>
• The Visual Studio.NET Help documentation. In Index, search by using the phrase **Debugging Windows Applications**.
</td>
</tr>
</table>

# Exercise 4
# Localizing the User Interface of an Application

In this exercise, you will update the Internal Business Application shell by localizing the AppControlForm form into French.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab08_1\Ex04\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab08_1\Ex04\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open the InternalBusinessApp project in Visual Studio .NET. Browse to *install_folder*\Labfiles\Lab08_1\Ex04\Starter to find the project. | **a.** For more information about opening a project file and starting an application, see the following resource:<br>• The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. For additional information about starting an application in the Designer, in Index, search by using the phrase **Debugging Windows Applications**. |
| **2.** View the AppControlForm form in the form designer. Set the **Language** property of the form to **French (France)**. | **a.** For more information about localizing an application, see the following resources:<br>• Practice: Localizing an Application in Module 8, "Enhancing the Usability of Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Lesson: Localizing an Application in Module 8, "Enhancing the Usability of Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. Search by using the phrase **Developing World-Ready Applications**. |

<table>
<tr><th>Tasks</th><th>Additional information</th></tr>
<tr>
<td>

**3.** Change the **Text** property of each control from English to its equivalent French translation.

**a.** Use the following table to set the value for **Text** property for each control.

<table>
<tr><th>Control</th><th>Value</th></tr>
<tr><td>Form</td><td>Tableau de bord des applications</td></tr>
<tr><td>HelpMenu</td><td>Aide</td></tr>
<tr><td>HelpMenuItem</td><td>Aide</td></tr>
<tr><td>AboutMenuItem</td><td>À propos de l'application interne d'entreprise</td></tr>
<tr><td>Make Travel Plans</td><td>Préparer un voyage</td></tr>
<tr><td>Expense Reporting</td><td>Notes de frais</td></tr>
<tr><td>Procurement</td><td>Réservation</td></tr>
<tr><td>Exit</td><td>Quitter</td></tr>
</table>

**Note:** You can copy and paste the French strings from the file *install_folder*\Labfiles\Lab08_1\Ex04\Starter\InternalBusinessApplicationLocalizedStrings.rtf.

</td>
<td>

**a.** For more information about localizing an application, see the following resources:

- Practice: Localizing an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- Lesson: Localizing an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- The .NET Framework SDK documentation. Search by using the phrase **Developing World-Ready Applications**.

</td>
</tr>
<tr>
<td>

**4.** Add code to the AppControlForm form to set the culture to match the regional and language information that the user has set in Control Panel.

</td>
<td>

**a.** For more detailed information about the tasks that you must perform, see the TODO comments in the code.

**b.** For more information about localizing an application, see the following resources:

- Practice: Localizing an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- Lesson: Localizing an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.
- The .NET Framework SDK documentation. Search by using the phrase **Developing World-Ready Applications**.

</td>
</tr>
</table>

| Tasks | Additional information |
|---|---|
| **5.** Change the default language for the computer, and test the application.<br>**a.** Open Control Panel in Windows XP Professional:<br>• If you are using Category View in Control Panel, click Date, Time, Language, and Regional Options, and then click Regional and Language Options.<br>• If you are using Classic View in Control Panel, double-click **Regional and Language Options**.<br>**b.** In the **Regional and Language Options** dialog box, on the **Regional Options** tab, in the **Standards and formats** area, in the list of languages, click **French (France)**.<br>**c.** Run the application. | No additional information is required for this task. |

# Exercise 5
# Localizing Resources in an Application

In this exercise, you will update the Internal Business Application shell by localizing strings used in the AppControlForm form.

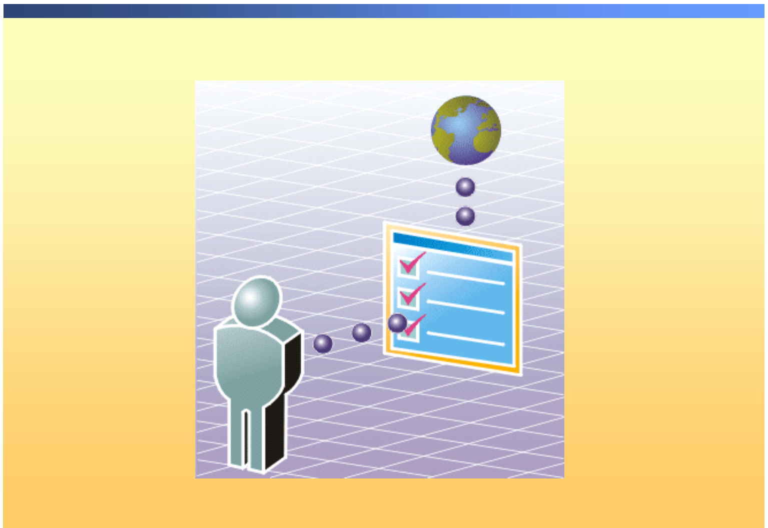
There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab08_1\Ex05\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab08_1\Ex05\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open the InternalBusinessApp project in Visual Studio .NET. Browse to *install_folder*\Labfiles\Lab08_1\Ex05\Starter to find the project. | **a.** For more information about opening a project file, see the following resource:<br>• The Visual Studio.NET Help documentation. Search by using the phrase **Open Project Dialog Box**. |
| **2.** Create a new resource file, and name it AppControlRes.fr-FR.resx. | **a.** For more information about localizing an application, see the following resources:<br>• Practice: Localizing an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• Lesson: Localizing an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. Search by using the phrase **Developing World-Ready Applications**. |

<table>
<tr><th>Tasks</th><th>Additional information</th></tr>
<tr>
<td>
<b>3.</b> Add text in French to the resource file.<br>
<b>a.</b> Use the following table to set the <b>Name</b> and <b>Value</b> elements for each resource string.
<table>
<tr><th>Name</th><th>Value</th></tr>
<tr><td>AppUnavailableMessage</td><td>Application non disponible</td></tr>
<tr><td>PermissionDeniedMessage</td><td>Autorisation refusée</td></tr>
<tr><td>UnauthorizedMessage</td><td>Vous ne disposez pas des autorisations nécessaires pour utiliser cette application</td></tr>
<tr><td>OfflineMode</td><td>Mode hors connexion</td></tr>
<tr><td>OnlineMode</td><td>Mode connexion</td></tr>
</table>
<b>Note:</b> You can copy and paste the French strings from the file <i>install_folder</i>\Labfiles\Lab08_1\Ex05\Starter\InternalBusinessApplicationLocalizedStrings.rtf.
</td>
<td>
<b>a.</b> For more information about localizing an application, see the following resources:<br>
• Practice: Localizing an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, <i>Developing Microsoft .NET Applications for Windows (Visual C# .NET)</i>.<br>
• Lesson: Localizing an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, <i>Developing Microsoft .NET Applications for Windows (Visual C# .NET)</i>.<br>
• The .NET Framework SDK documentation. Search by using the phrase <b>Developing World-Ready Applications</b>.
</td>
</tr>
<tr>
<td>
<b>4.</b> Add code to the AppControlForm form to set the culture to match the regional and language information that the user has set in Control Panel.
</td>
<td>
<b>a.</b> For more detailed information about the tasks that you must perform, see the TODO comments in the code.<br>
<b>b.</b> For more information about localizing an application, see the following resources:<br>
• Lesson: Localizing an Application in Module 8, “Enhancing the Usability of Applications,” in Course 2555A, <i>Developing Microsoft .NET Applications for Windows (Visual C# .NET)</i>.<br>
• The .NET Framework SDK documentation. Search by using the phrase <b>Developing World-Ready Applications</b>.
</td>
</tr>
</table>

| Tasks | Additional information |
|---|---|
| **5.** Make sure that the default language for the computer is set still to **French (France)** and test the application.<br>**a.** To change the default language for the computer, open Control Panel in Windows XP Professional.<br>• If you are using Category View in Control Panel, click **Date, Time, Language, and Regional Options**, and then click **Regional and Language Options**.<br>• If you are using Classic View in Control Panel, double-click **Regional and Language Options**.<br>**b.** In the **Regional and Language Options** dialog box, on the **Regional Options** tab, in the **Standards and formats** area, in the list of languages, click **French (France)**.<br>**c.** Run the application. | No additional information is required for this task. |

# Course Evaluation



Your evaluation of this course will help Microsoft understand the quality of your learning experience.

At a convenient time between now and the end of the course, please complete a course evaluation, which is available at http://www.metricsthatmatter.com/survey.

Microsoft will keep your evaluation strictly confidential and will use your responses to improve your future learning experience.

msdn® training

# Module 9: Deploying Windows Forms Applications

**Contents**

Microsoft®

# Instructor Notes

**Presentation:**
**120 minutes**

**Lab:**
**30 minutes**

In this module, students learn about assemblies and the use of strong-named assemblies and the global assembly cache in the Microsoft® .NET Framework. Students also learn how to configure and deploy Windows Forms applications.

After completing this module, students will be able to:

- Use strong-named assemblies in .NET applications.
- Use application configuration files to configure and use Microsoft Windows® Installer 2.0 to package and deploy .NET applications.

**Required materials**

To teach this module, you need Microsoft PowerPoint® file 2555A_09.ppt.

**Preparation tasks**

To prepare for this module:

- Read all of the materials for this module.
- Complete the demonstrations, practices and lab.

# How to Teach This Module

This section contains information that will help you to teach this module.

- If students are interested in referencing code examples in other languages, point them to "Language Equivalents" in the Help documentation for the Microsoft Visual Studio® .NET development system. This section provides examples in languages such as Microsoft Visual Basic® .NET, C#, and Java.
- Lab 9.1, Deploying an Application is based on the Expense Report application used throughout Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C#™ .NET)* and is intended to simulate a real-world environment in which students will demonstrate what they learned during the lecture and practice portions of the module. The lab does not provide step-by-step detailed instructions; instead, the students are given tasks to complete in the left column and a list of resources that they can use (if they need help) in the right column. Students get the hands-on experience that they need by completing the practice activity in the module.

# Lesson: .NET Assemblies

This section describes the instructional methods for teaching this lesson.

The key points of this lesson are for the student to understand the difference between private and strong-named assemblies and the appropriate scenarios for the use of each.

### How to Install Assemblies into the Global Assembly Cache

Also, be sure to emphasize that the GACUtile.exe utility is for use during application development. For application development, assemblies should be installed in the global assembly cache by using the .NET Framework configuration tool (Mscorcfg.msc) or Windows Installer setup projects.

# Lesson: Deploying Windows Forms Applications

This section describes the instructional methods for teaching this lesson.

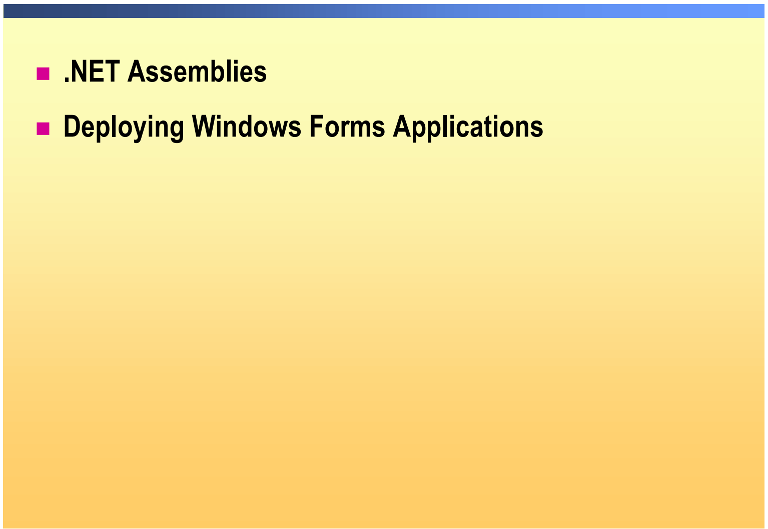
In this lesson, point out to students that the deployment of Windows Forms applications is easier thanks to the support added by the .NET Framework, which avoids the old problem of dynamic link-library (DLL) versioning issues.

# Lab 9.1: Deploying an Application

This section describes the instructional methods for teaching this lab.

If a student opens the InternalBusinessApplication.sln solution file for Exercise 3, Deploying a .NET Application, the work area is initially blank. Tell students that to view the file system editor, in Solution Explorer they must right-click **InternalBusinessApplication**, click **View**, and then click **File System**.

# Overview



- .NET Assemblies
- Deploying Windows Forms Applications

*****************************ILLEGAL FOR NON-TRAINER USE******************************

### Introduction

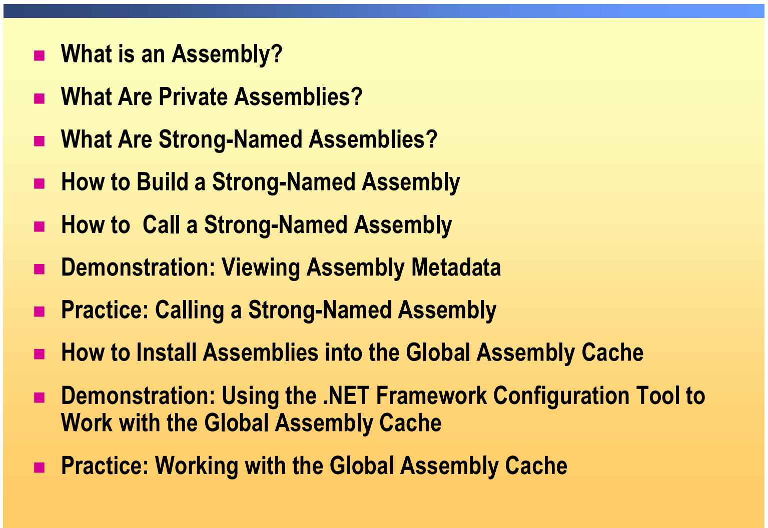
In this module, you will learn about assemblies and the use of strong-named assemblies and the global assembly cache in the Microsoft® .NET Framework. You will also learn how to configure and deploy your .NET applications.

### Objectives

After completing this module, you will be able to:

- Use strong-named assemblies in .NET applications.
- Use application configuration files to configure and use Microsoft Windows® Installer 2.0 to package and deploy .NET applications.

# Lesson: .NET Assemblies



- What is an Assembly?
- What Are Private Assemblies?
- What Are Strong-Named Assemblies?
- How to Build a Strong-Named Assembly
- How to  Call a Strong-Named Assembly
- Demonstration: Viewing Assembly Metadata
- Practice: Calling a Strong-Named Assembly
- How to Install Assemblies into the Global Assembly Cache
- Demonstration: Using the .NET Framework Configuration Tool to Work with the Global Assembly Cache
- Practice: Working with the Global Assembly Cache

**Introduction**

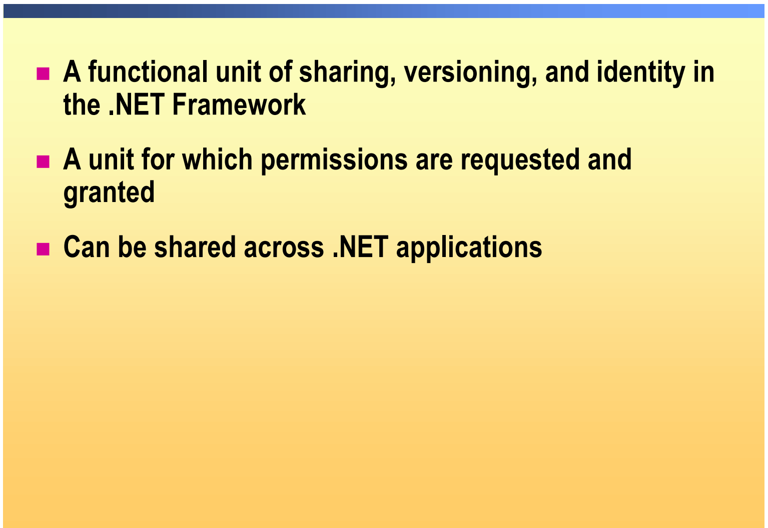
In this lesson, you will learn about the difference between private and strong-named assemblies, how to build and call strong-named assemblies. You will learn how to install strong-named assemblies in the global assembly cache and why you would want to do so. You will also learn how to use the Fusion Log Viewer (FusLogVw utility) to trace the assembly load process to troubleshoot application activation problems.

**Lesson objectives**

After completing this lesson, you will be able to:

- Describe the difference between private and strong-named assemblies.
- Build strong-named assemblies and call them from an application.
- Install strong-named assemblies in the global assembly cache.
- Use the FusLogVw utility to trace the assembly load process to troubleshoot application activation problems.

# What is an Assembly?



- A functional unit of sharing, versioning, and identity in the .NET Framework
- A unit for which permissions are requested and granted
- Can be shared across .NET applications

**Introduction**

Applications in the .NET Framework always consist of one or more assemblies.

**Definition**

An assembly:

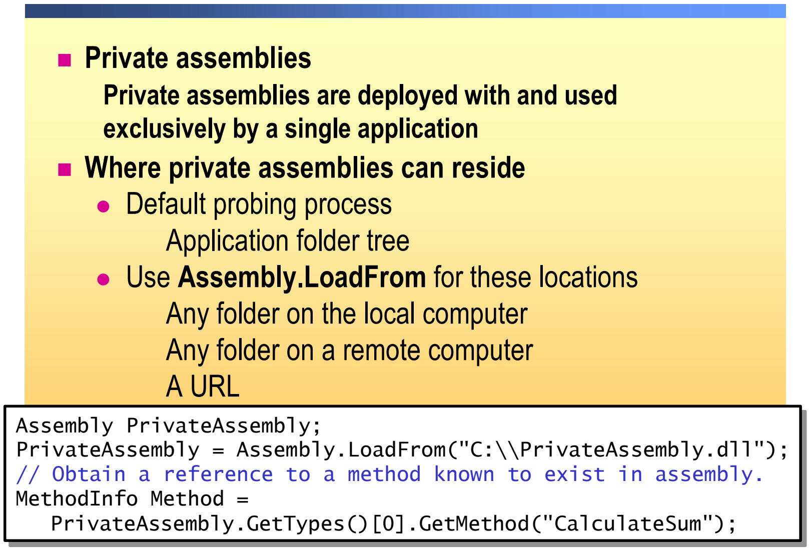
- Is a functional unit of sharing, versioning, and identity in the .NET Framework.
- Is a unit for which permissions are requested and granted.
- Can be shared across .NET applications.

In the simplest case, an application can consist of one assembly that contains one managed module with all the code and resources for the application. In most scenarios, however, an application has multiple assemblies, and each assembly may have multiple files. References between assemblies are not resolved until the code making the call is executed. So, all assemblies of an application need not be present at run time. Assemblies, such as localization resources, can be retrieved on demand.

All assemblies are self-describing. Each assembly contains metadata that includes the identity and version of the assembly and the types implemented by that assembly. You can use the Microsoft intermediate language (MSIL) Disassembler (Ildasm.exe) to view the contents of the assembly file. At the command prompt, type the following command:

```
ildasm <assembly name>
```

# What are Private Assemblies?



- **Private assemblies**

  **Private assemblies are deployed with and used exclusively by a single application**

- **Where private assemblies can reside**
  - Default probing process

    Application folder tree
  - Use **Assembly.LoadFrom** for these locations

    Any folder on the local computer

    Any folder on a remote computer

    A URL

```
Assembly PrivateAssembly;
PrivateAssembly = Assembly.LoadFrom("C:\\PrivateAssembly.dll");
// Obtain a reference to a method known to exist in assembly.
MethodInfo Method =
   PrivateAssembly.GetTypes()[0].GetMethod("CalculateSum");
```

### Introduction

In the .NET Framework, you can create source code by using any programming language that is compatible with the .NET Framework. You then use the corresponding compiler to build a managed module. A managed module is a standard portable executable (PE) file for the Windows operating system that contains MSIL code that is executed by the common language runtime. Managed modules must be packaged in assemblies before they can be executed by the common language runtime, therefore all code written for the .NET Framework must be packaged as an assembly before it can be executed.

### Definition

A private assembly is deployed with and used exclusively by a single application. A private assembly is referenced by its simple name. The simple name is defined in the assembly's metadata.

### Where private assemblies can reside

When the common language runtime searches for a private assembly, it uses the assembly's simple name to locate the referenced assembly.

When the application is installed, the private assembly is also installed in the same root directory of the application, or in a subdirectory that appears under the root directory of the application.

The probing process for private assemblies can be summarized with the following steps:

1. The runtime obtains the private assembly's simple name from the metadata of the application that is referencing the private assembly.
2. The runtime starts probing first in the application's root directory, next in the assembly's named subdirectory, and then in the assembly's culture subdirectory. For example, if the application MyApp is installed in C:\Program Files\MyApp, the following directories may be searched:

   C:\Program Files\MyApp\MyAssembly.dll.

   C:\Program Files\MyApp\MyAssembly\MyAssembly.dll.

   C:\Program Files\MyApp\MyAssembly.exe.

   C:\Program Files\MyApp\MyAssembly\MyAssembly.exe.

   C:\Program Files\MyApp\de\MyAssembly.dll.

   C:\Program Files\MyApp\de\MyAssembly\MyAssembly.dll.

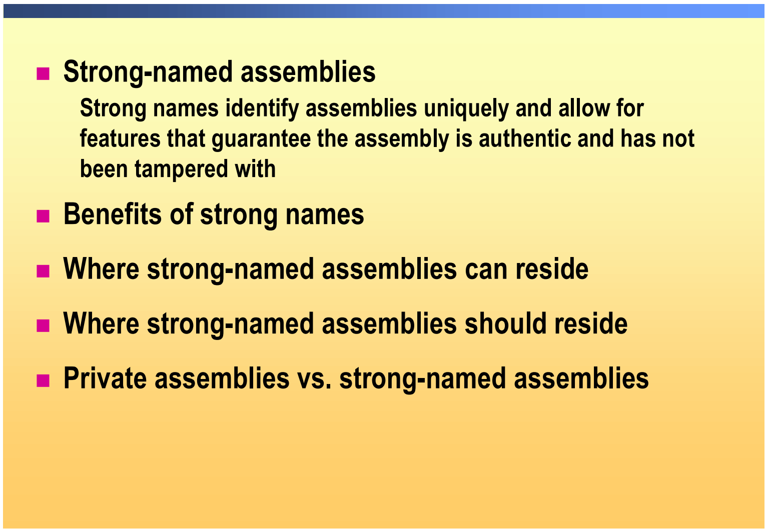
   C:\Program Files\MyApp\de\MyAssembly.exe.

   C:\Program Files\MyApp\de\MyAssembly\MyAssembly.exe.

To access an assembly that is not in the directory tree of the application, you can use the **Assembly.LoadFrom()** method, providing the location where the assembly can be found. To use **Assembly.LoadFrom()**, you must include the **using** directive, **using System.Reflection**. The following code demonstrates how to call **LoadFrom()**:

```
Assembly PrivateAssembly;
PrivateAssembly =
     Assembly.LoadFrom ("C:\\PrivateAssembly.dll");
// Obtain a reference to a method known to
// exist in the assembly.
MethodInfo Method =
     PrivateAssembly.GetTypes()[0].GetMethod("CalculateSum");
```

# What are Strong-Named Assemblies?



- **Strong-named assemblies**

  Strong names identify assemblies uniquely and allow for features that guarantee the assembly is authentic and has not been tampered with
- **Benefits of strong names**
- **Where strong-named assemblies can reside**
- **Where strong-named assemblies should reside**
- **Private assemblies vs. strong-named assemblies**

****************************ILLEGAL FOR NON-TRAINER USE******************************

### Introduction

To get many of the benefits assemblies provide, such as sharing by multiple applications on the system, side-by-side versioning, and better support for security and deployment, you must use strong-named assemblies.

### Definition

The .NET Framework uses strong names to provide a way to identify assemblies uniquely, allowing applications to run with the versions of the strong-named assemblies they were built with. Also, strong-names allow for features to guarantee the assembly is authentic and has not been tampered with.

Strong-named assemblies consist of the assembly's identity, which is:

- The simple text name of the assembly.
- The version number of the assembly.
- The culture information, if it is provided (optional).
- A public key for the client.
- A digital signature.

The tool used to create the assembly generates a hash of the file that contains the assembly's manifest. The private key is used to sign the hash (digital signature). The digital signature is stored in the PE file that contains the manifest for the assembly. The public key is used by the client of the strong-named assembly to decrypt its digital signature.

### Benefits of strong names

Strong names guarantee name uniqueness by relying on unique key pairs. No one can generate the same assembly name that you can, because an assembly generated with one private key has a different name than an assembly generated with another private key.

Strong names protect the version lineage of an assembly. A strong name can ensure that no one can produce a subsequent version of your assembly. Users can be sure that a version of the assembly they are loading comes from the same publisher that created the version the application was built with.

Strong names provide a strong integrity check. Passing the .NET Framework security checks guarantee that the contents of the assembly have not been changed since it was built. Note, however, that strong names in and of themselves do not imply a level of trust, such as that provided by a digital signature and the supporting certificate.

**Note** Strong-named assemblies can only reference other strong-named assemblies.

When you reference a strong-named assembly, you are expecting to get certain benefits, such as versioning and naming protection. If the strong-named assembly you reference then references a private assembly which does not have these benefits, you lose the benefits you would derive from using a strong-named assembly and revert to dynamic link-library (DLL) conflicts.

### Where strong-named assemblies can reside

Strong-named assemblies can reside in:

- An application folder.
- Any folder on the local computer.
- Any folder on a remote computer.
- A URL.
- The global assembly cache.

### Where strong-named assemblies should reside

Assemblies shared by multiple applications should be installed in the global assembly cache, a centralized repository. .NET clients can access the same copy of the assembly, which is signed and installed in the global assembly cache. The global assembly cache can handle multiple versions of an assembly, and it's a secure place where assemblies can be stored. If the assembly is not going to be shared, then the assembly should be installed in the application directory tree. Once a strong-named assembly has been installed in the global assembly cache, it is referred to as a shared assembly.

The only differences between a strong-named assembly and a shared assembly are (a) where they are located and (b) during the integrity/security validation. If the assembly is installed in the global assembly cache, the runtime checks the assembly during the install. If the assembly is not installed in the global assembly cache, the integrity is checked at run time.

### Differences between private and strong-named assemblies

Some key differences between private and strong-named assemblies are listed below.

Private assemblies:

- Can only be installed in an application's directory structure.
- Are referenced only by their simple name.
- Can have version information, but the runtime does not use it.
- Are not installed in the global assembly cache and therefore the runtime will not look in the global assembly cache when probing for the private assembly.

Strong-named assemblies:

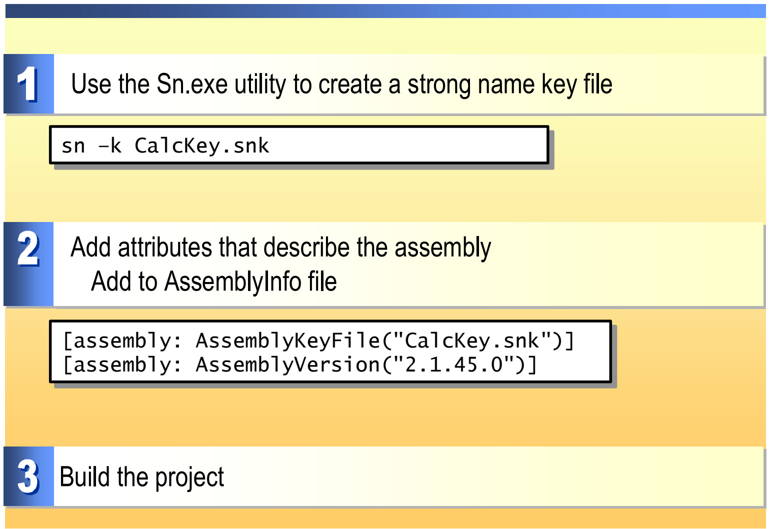
- Can be installed in a number of different locations.
- Are referenced by their simple name, culture, version, and public key.
- Contain version information that the runtime checks when loading the assembly.
- Can be installed in the global assembly cache and therefore the runtime will look in the global assembly cache as part of the probing process.
- Can have multiple versions deployed in a side-by-side manner in the global assembly cache.

In addition to the differences and benefits mentioned above, strong-named assemblies can be used instead of private assemblies to provide the following two benefits:

- When the application is built, it is tied to a specific version of the assembly it references. Therefore, a new version of the assembly cannot simply be installed in place of the original assembly because the application will not run.
- When the application runs, the reference strong-named assembly is verified for tampering prior to being loaded and executed. This prevents any malicious activity that may occur if the assembly being loaded is not the same one that built the application.

For more information about both private and strong-named assemblies, see Course 2350A, *Securing and Deploying Microsoft .NET Applications*.

# How to Build a Strong-Named Assembly



1 Use the Sn.exe utility to create a strong name key file

```
sn -k CalcKey.snk
```

2 Add attributes that describe the assembly

Add to AssemblyInfo file

```
[assembly: AssemblyKeyFile("CalcKey.snk")]
[assembly: AssemblyVersion("2.1.45.0")]
```

3 Build the project

*****************************ILLEGAL FOR NON-TRAINER USE******************************

### Introduction

To create a strong-named assembly, you need to use a private key to sign the assembly. The public key is included with the assembly to allow client applications to verify the strong name. The public key is automatically included in the assembly when it is signed. For more information about signing an assembly, see Module 10, “Securing Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C#™ .NET)*.

### Procedure: Creating a Strong-Named Assembly

To create a strong-named assembly:

1. Use the Sn.exe utility to create a strong name key file.

   The command-line Strong Name tool (Sn.exe) that comes with the .NET Framework is used to build strong-name key files. For example, the following command will build a strong-name key file named CalcKey.snk:

   ```
   sn -k CalcKey.snk
   ```

2. Add the **AssemblyKeyFile** attribute flags and other assembly attributes, as appropriate.

   To create a strong-named assembly, use assembly attributes. An assembly attribute is placed in the AssemblyInfo file that is part of the Microsoft Visual Studio® .NET project. Assembly attributes instruct the compiler to populate certain assembly metadata settings or perform certain actions as defined by the attribute.

3. Build the application.

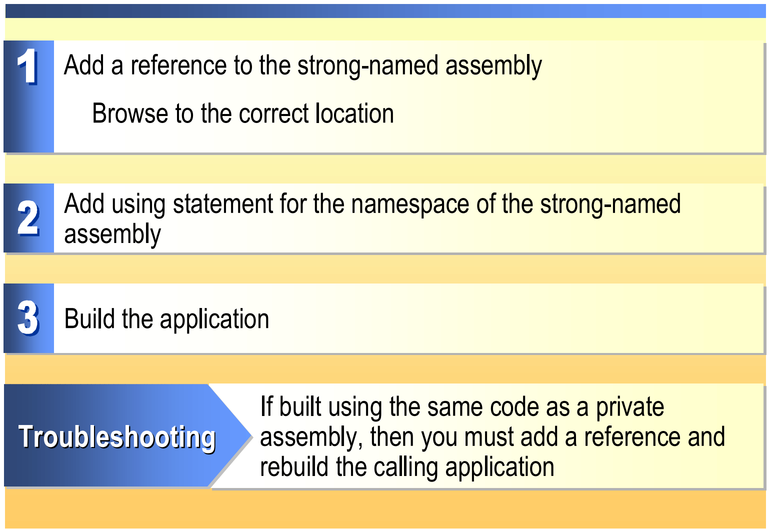
### Examples of assembly attributes

The **AssemblyKeyFile** attribute tells the compiler to create a strong-named assembly and indicates where the key pair can be found. The following example demonstrates the use of the **AssemblyKeyFile** and **AssemblyVersion** attributes (note that the **AssemblyVersion** attribute is not used to force the compile to build a strong-named assembly, but is needed to give the assembly a version number):

```
using System.Reflection;
  …
[assembly: AssemblyKeyFile("CalcKey.snk")]
[assembly: AssemblyVersion("2.1.45.0")]
```

There are other assembly attributes that can be added to the AssemblyInfo file. For more information about assembly attributes, search on the phrase **assembly-level attributes** in Visual Studio .NET online Help.

# How to Call a Strong-Named Assembly



1 Add a reference to the strong-named assembly

Browse to the correct location

2 Add using statement for the namespace of the strong-named assembly

3 Build the application

Troubleshooting

If built using the same code as a private assembly, then you must add a reference and rebuild the calling application

*****************************ILLEGAL FOR NON-TRAINER USE******************************

**Introduction**

Now that you have created a strong-named assembly, how do you call it from an application?

**Procedure: Calling a Strong-Named Assembly**

To call a strong-named assembly:

1. Add a reference to the strong-named assembly to your project:
   a. In the Visual Studio .NET Solution Explorer, right-click the **References** node.
   b. Click **Add Reference**.

      The **Add Reference** dialog box appears.
   c. Click the **Browse** button to locate the assembly.
   d. When you have selected the correct assembly, click **OK**.
   e. Check the References node in Solution Explorer, to verify that the reference to the assembly has been added.
2. Add a **using** statement for the Namespace of the strong-named assembly to the source files that require access to the assembly.
3. Build and run the application.

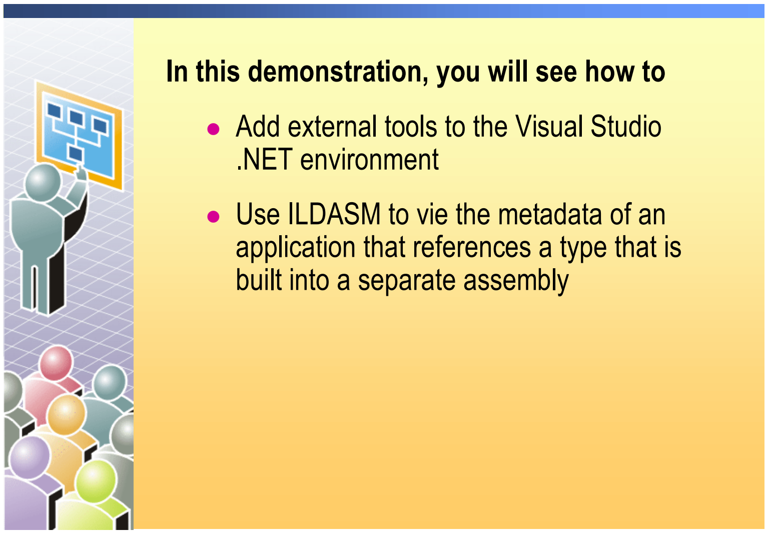
**Troubleshooting**

Default binding policy refers to the process by which the runtime locates an assembly based on the binding information contained in the application. An application will be bound to an assembly it was built and tested with, even if a newer version of the assembly is available. Default binding policy is always used unless it is explicitly overridden.

**Important** If you originally build an application that uses a private assembly and then make a strong-named version of that private assembly, you must add a reference to the strong-named assembly and rebuild the calling application in order for the calling application to use the strong-named version of the assembly.

In the lesson, Deploying Windows Forms Applications, in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Microsoft Windows (Visual C# .NET)*, you will learn how you can use application configuration files to override the default binding policy.

# Demonstration: Viewing Assembly Metadata



**In this demonstration, you will see how to**

- Add external tools to the Visual Studio .NET environment
- Use ILDASM to vie the metadata of an application that references a type that is built into a separate assembly

### Introduction

In this demonstration, you will see how to add external tools to the Visual Studio .NET environment. You will then see how to view the metadata of an application that references a type that is built into a separate assembly.

**Note** If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

### Instructions

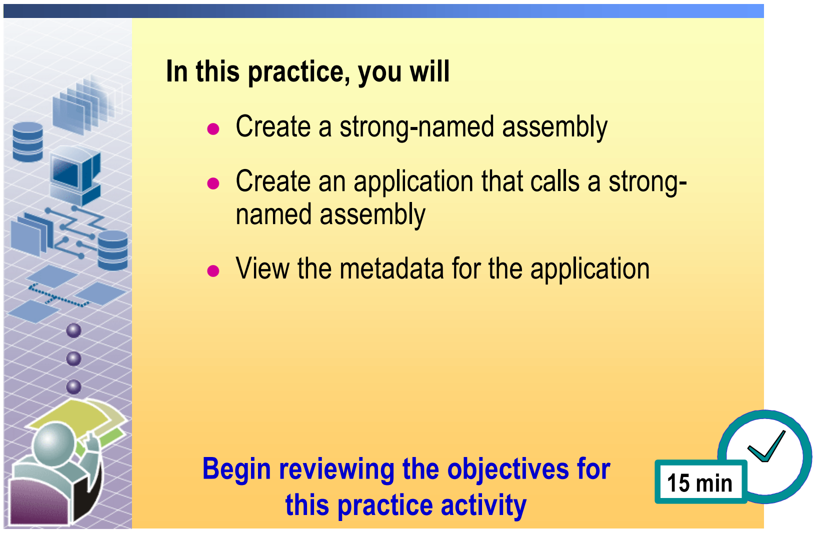
► **To add ILDASM to the Visual Studio .NET environment**

1. Start Visual Studio .NET.
2. From the **Tools** menu, select **External Tools**.
3. In the **External Tools** dialog box, click **Add**.
4. In the **Title** box, type **MSIL Disassembler**
5. Click the **…** button to the right of the **Command** field.
6. In the **Open File** dialog box, browse to C:\Program Files\Microsoft Visual Studio .NET\FrameworkSDK\bin.
7. Select **ildasm.exe** and click **Open**.
8. On the **External Tools** dialog box, click **OK**.
9. View the **Tools** menu again and notice that a menu item for MSIL Disassembler is now available.

### ► To view the metadata of an assembly

1. In Visual Studio .NET, click the **Tools** menu and select **MSIL Disassembler**.
2. When the ILDASM window appears, click the **File** menu and then click **Open**.
3. In the **Open** dialog box, browse to *install_folder*\Democode\Mod09\Mod09_01.
4. Select CalculatorEngine.dll and click Open.
5. Notice that the CalculatorEngine.dll node has two sub-nodes, the MANIFEST and the Calculator namespace.
6. Double-click the MANIFEST sub-node. Notice that there is an **.assembly CalculatorEngine** entry in the manifest. Below this reference you will see the **.ver** entry showing what version of CalculatorEngine.dll was built. CalculatorEngine is not a strong-named assembly because there is no public key entry in the manifest. Therefore, even though a version number has been applied to CalculatorEngine.dll, the runtime will not consider it when probing for this assembly when an application that references it is executed.
7. Close the MANIFEST window and ILDASM.
8. Run ILDASM from the Tools menu again on WindowsCalculator.exe.
9. Notice that the WindowsCalculator.exe node also has two sub nodes, the MANIFEST and the WindowsCalculator namespace.
10. Double-click the MANIFEST sub-node.
11. Notice that there is an entry in the manifest called **.assembly extern CalculatorEngine** that contains the version number. This indicates that this reference is to an EXTERNAL assembly—one that is different than WindowsCalculator.exe.

# Practice: Calling a Strong-Named Assembly



In this practice, you will

- Create a strong-named assembly
- Create an application that calls a strong-named assembly
- View the metadata for the application

Begin reviewing the objectives for this practice activity

15 min

**Introduction**

In this practice, you will create a strong-name key pair file and reference this file in the CalculatorEngine library. You will then create the WindowsCalculator application so that it references the strong-named assembly. In addition, you will use ILDASM to inspect the metadata of both CalculatorEngine.dll and WindowsCalculator.exe.

**Instructions**

► **Creating a strong-named assembly**

1. Open a Command Prompt window, by clicking **Start**, pointing to **All Programs**, pointing to **Microsoft Visual Studio .NET**, pointing to **Visual Studio .NET Tools**, and then clicking **Visual Studio .NET Command Prompt**.
2. At the command prompt, change directory to *install_folder*\Practices\Mod09\Mod09_01\Starter\CalculatorEngine.
3. Enter the command **sn –k CalcKey.snk**. This creates a strong-named key file that we can use to build a strong-named assembly.
4. In Visual Studio .NET, open the CalculatorEngine.sln solution file in *install_folder*\Practices\ Mod09\Mod09_01\Starter\CalculatorEngine.
5. Show TODO comments in the Task List.

   To show TODO comments, click the **View** menu, point to **Show Tasks**, and then click **All**.
6. Open the AssemblyInfo.cs file.
7. Locate the first TODO comment. Change the version number from 2.0.1.1 to **3.0.1.1**.

```
[assembly: AssemblyVersion("3.0.1.1")]
```

8. Locate the next TODO comment. Add a new attribute at the bottom of the file to reference the strong name key pair file.

```
[assembly: AssemblyKeyFile("CalcKey.snk")]
```

9. Open the Calculator.cs source file.
10. Locate the TODO comment. Change the versionInfo string from v2.0.1.1 to v3.0.1.1.

```
private static string versionInfo = "Calculator v3.0.1.1";
```

11. Rebuild the assembly and then close Visual Studio.NET.
12. In a Visual Studio .NET Command Prompt window, change directories to *install_folder*\Practices\Mod09\Mod09_01\Starter\CalculatorEngine\bin\Debug.
13. At the command prompt, run ILDASM on CalculatorEngine.dll.

```
ildasm CalculatorEngine.dll
```

14. Open the MANIFEST sub-node.
15. Notice that under the **.assembly CalculatorEngine** entry, there is a **.publickey** entry. This indicates that CalculatorEngine is a strong-named assembly.
16. Close the MANIFEST window and ILDASM.

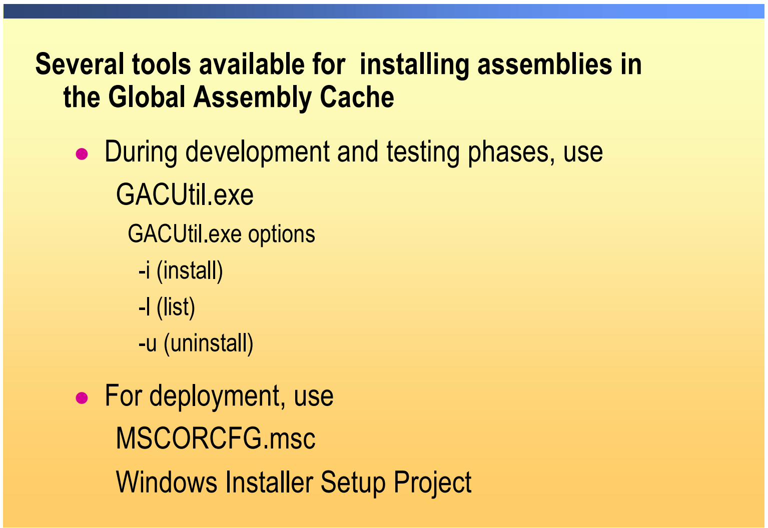
### ► Creating an application that references the strong-named assembly

1. In Visual Studio .NET, open the WindowsCalculator.sln solution file in *install_folder*\Practices\Mod09\Mod09_01\Starter. Rebuild the solution.
2. Run the application and notice that the version number is 3.0.1.1. This is the version number of CalculatorEngine.dll.
3. In a Visual Studio .NET Command Prompt window, change directories to *install_folder*\Practices\ Mod09\Mod09_01\Starter \bin\debug.
4. Run ILDASM on WindowsCalculator.exe by typing the command:

```
ildasm WindowsCalculator.exe
```

5. Open the MANIFEST sub- node. Notice that the **.assembly extern CalculatorEngine** entry has both the version number entry of 3:0:1:1, and a **.publickeytoken** entry. This shows that this application has a reference to an external strong name assembly.

# How to Install Assemblies into the Global Assembly Cache



**Several tools available for installing assemblies in the Global Assembly Cache**

- During development and testing phases, use
  - GACUtil.exe
    - GACUtil.exe options
      - -i (install)
      - -l (list)
      - -u (uninstall)
- For deployment, use
  - MSCORCFG.msc
  - Windows Installer Setup Project

### Introduction

The global assembly cache is a machine-wide code cache used for storing one or more versions of an assembly. Storing multiple versions of an assembly in the global assembly cache is called side-by-side deployment. The global assembly cache stores assemblies that can be shared among many applications on the computer.

Assemblies that are deployed in the global assembly cache must have a strong name. This is because the global assembly cache has stricter naming rules to ensure that side-by-side deployment and execution is possible. As a result, when you install an assembly in the global assembly cache, the common language runtime verifies that the assembly is unique to any other assembly already installed in the global assembly cache and has not been tampered with. The verification process begins by decrypting the signature of the strong name by **using** the public key that is included in the assembly. If the signature is not present or the results of decrypting the signature do not match the contents of the assembly, the installation process is terminated.

To install or uninstall an assembly in the global assembly cache, you need to have administrator-level privileges on the computer on which the cache resides. Assemblies should be installed in the global assembly cache by using Windows Installer or by using the .NET Framework configuration tool, Mscorcfg.msc. During development, you can use the Global Assembly Cache tool or Windows Explorer to install your assembly in the global assembly cache for testing purposes. Both of these tools are available for convenience only and should not be used for production deployment.

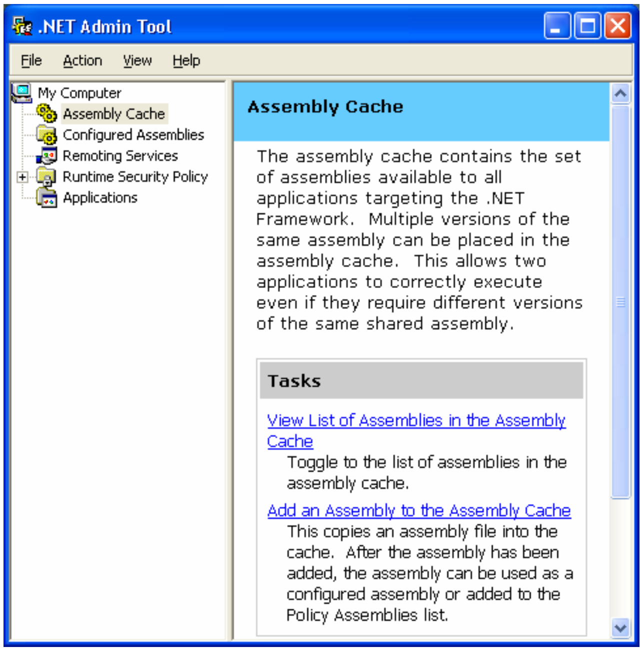
### Using GACUtil.exe

GACUtil.exe is a command line utility that allows you to view and manipulate the contents of the global assembly cache. GACUtil.exe options include:

- /i or –i: installs a strong-named assembly into the global assembly cache.
- /l or –l: lists the contents of the global assembly cache.
- /u or –u: removes one or more assemblies from the global assembly cache.

### Using MSCORCFG.msc

The .NET Framework Configuration tool is a Microsoft Management Console (MMC) snap-in that allows you to manage and configure assemblies in the global assembly cache, adjust code access security policy, and adjust remoting services.



Assembly Cache

The assembly cache contains the set of assemblies available to all applications targeting the .NET Framework. Multiple versions of the same assembly can be placed in the assembly cache. This allows two applications to correctly execute even if they require different versions of the same shared assembly.

Tasks

View List of Assemblies in the Assembly Cache

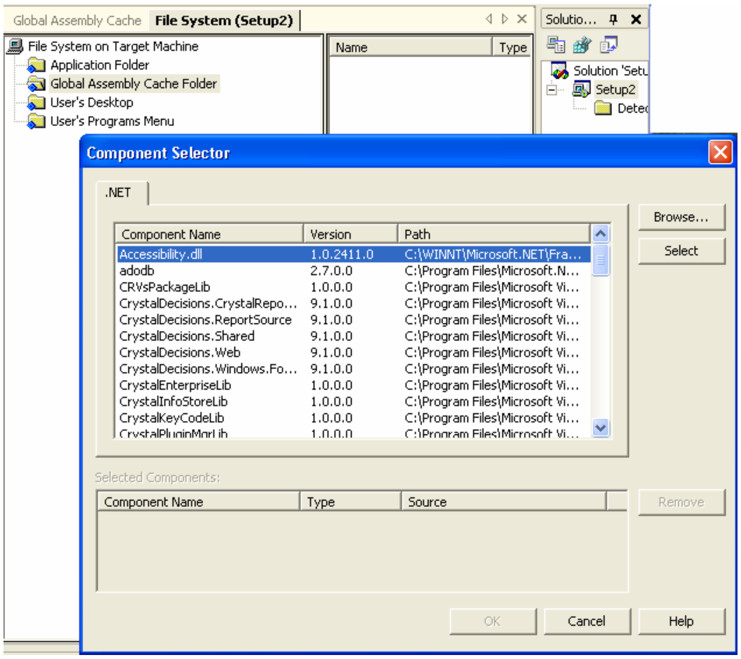
Toggle to the list of assemblies in the assembly cache.

Add an Assembly to the Assembly Cache

This copies an assembly file into the cache. After the assembly has been added, the assembly can be used as a configured assembly or added to the Policy Assemblies list.

### Using a Windows Installer 2.0 setup project

When you create a Microsoft Windows Installer 2.0 setup project for your application, you can add strong-named assemblies to the Global Assembly Cache node of your setup project.

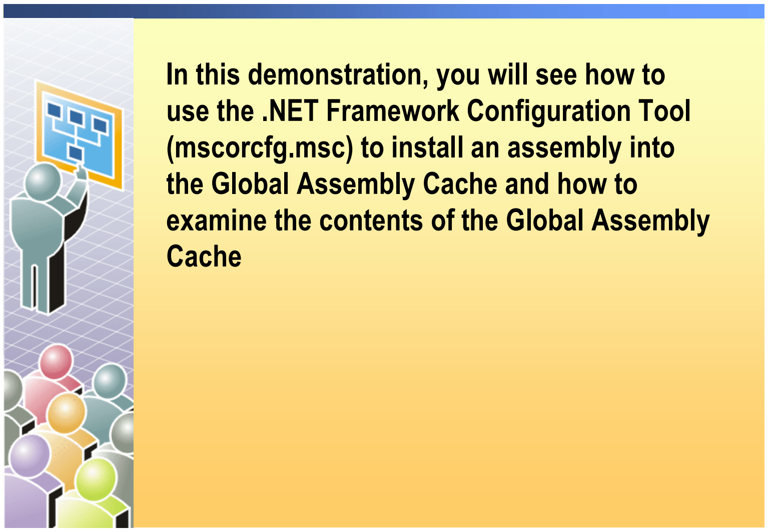


| Component Name | Version | Path |
|---|---|---|
| Accessibility.dll | 1.0.2411.0 | C:\WINNT\Microsoft.NET\Fra... |
| adodb | 2.7.0.0 | C:\Program Files\Microsoft.N... |
| CRVsPackageLib | 1.0.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalDecisions.CrystalRepo... | 9.1.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalDecisions.ReportSource | 9.1.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalDecisions.Shared | 9.1.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalDecisions.Web | 9.1.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalDecisions.Windows.Fo... | 9.1.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalEnterpriseLib | 1.0.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalInfoStoreLib | 1.0.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalKeyCodeLib | 1.0.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalPluginMgrLib | 1.0.0.0 | C:\Program Files\Microsoft Vi... |

| Component Name | Type | Source |
|---|---|---|

When your application is installed by using the Windows Installer, these strong-named assemblies will be installed into the global assembly cache of the computer where the installed project is run. For more information about setup projects see the lesson, Deploying Windows Forms Applications, Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*

# Demonstration: Using the .NET Framework Configuration Tool to Work with the Global Assembly Cache



In this demonstration, you will see how to use the .NET Framework Configuration Tool (mscorcfg.msc) to install an assembly into the Global Assembly Cache and how to examine the contents of the Global Assembly Cache

### Introduction

In this demonstration, you will see how to use the .NET Framework Configuration Tool (mscorcfg.msc) to install an assembly into the global assembly cache, and how to examine the contents of the global assembly cache.

### Instructions

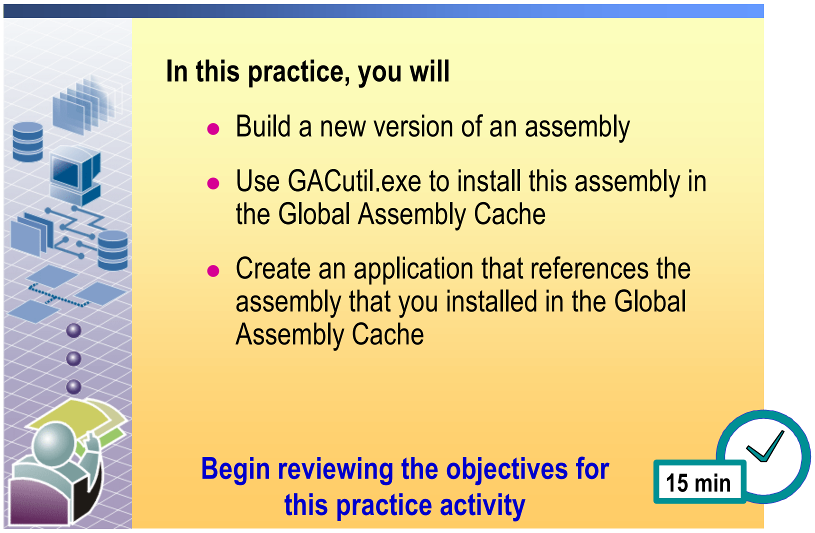
► **To install an assembly into the global assembly cache by using the .NET Framework configuration tool**

1. Run the .NET Framework configuration tool (mscorcfg.msc) by double-clicking the mscorcfg.msc icon on the desktop.
2. In the left pane of the .NET Framework Configuration Tool window, click the **Assembly Cache** node.
3. In the right pane of the .NET Framework Configuration Tool window, click **Add an Assembly to the Assembly Cache**.
4. In the **Add an Assembly** dialog, change folders to *install_path*\Democode\Mod09\Mod09_02.
5. Select CalculatorEngine.dll and click the **Open** button.

► **To use the .NET Framework configuration tool to examine the global assembly cache**

1. In the right pane of the .NET Framework Configuration Tool window, click **View List of Assemblies in the Assembly Cache**.
2. Scroll through the cache list and locate CalculatorEngine. Notice it lists the version, culture, and public key token information as well.

# Practice: Working with the Global Assembly Cache



In this practice, you will

- Build a new version of an assembly
- Use GACutil.exe to install this assembly in the Global Assembly Cache
- Create an application that references the assembly that you installed in the Global Assembly Cache

Begin reviewing the objectives for this practice activity

### Introduction

In this practice, you will install the CalculatorEngine assembly in the global assembly cache. You will then build the WindowsCalculator application. Prior to running the application, you will remove all the local copies of CalculatorEngine.dll. You will then verify that WindowsCalculator still runs because it finds the correct version of CalculatorEngine in the global assembly cache.

**Important** During development, you can use the global assembly cache tool (GACUtil) to install your assembly in the global assembly cache for testing purposes. This tool is available for convenience only and should not be used for production deployment. For production environments, assemblies should be installed in the global assembly cache by using Windows Installer or by using the .NET Framework configuration tool, Mscorcfg.msc.

### Instructions

► **Building a new version of the CalculatorEngine assembly**

1. In Visual Studio .NET, open the CalculatorEngine.sln solution file in *install_folder*\Practices\Mod09\Mod09_02\Starter\CalculatorEngine.
2. Show TODO comments in the Task List.

   To show TODO comments, click the **View** menu, point to **Show Tasks**, and then click **All**.
3. Open the AssemblyInfo.cs file.
4. Locate the first TODO comment. Change the version number from **3.0.1.1** to **4.0.1.1**.

```
[assembly: AssemblyVersion("4.0.1.1")]
```

5. Open the Calculator.cs source file.
6. Locate the TODO comment. Change the versionInfo string from **v3.0.1.1** to **v4.0.1.1**.

```
private static string versionInfo = "Calculator v4.0.1.1";
```

7. Rebuild the library, and then close Visual Studio .NET.

### ► Using GACUtil to install an assembly into the global assembly cache

1. Open the Visual Studio .NET Command Prompt window by clicking **Start** pointing to **All Programs**, pointing to **Microsoft Visual Studio .NET**, pointing to **Visual Studio .NET Tools**, and then clicking **Visual Studio .NET Command Prompt**.
2. Change directories to *install_folder*\Practices\Mod09\Mod09_02\Starter\CalculatorEngine\bin\Debug.
3. Install the CalculatorEngine assembly into the global assembly cache by entering the command:

```
gacutil -i CalculatorEngine.dll
```

   You should receive the message **Assembly successfully added to the cache**.
4. Display the contents of the global assembly cache by entering the command:

```
gacutil -l
```

5. Scroll through the list to find CalculatorEngine. The entry should contain the information “CalculatorEngine, Version=4.0.1.1, Culture-neutral, PublicKeyToken=***, Custom=null”.

### ► Building the application that references the strong-named assembly in the global assembly cache

1. In Visual Studio .NET, open the WindowsCalculator.sln solution file in *install_folder*\Practices\Mod09\Mod09_02\Starter.
2. Expand the References node in the Solution Explorer.
3. Right-click the References node and select **Add Reference**.
4. In the **Add Reference** dialog box, click **Browse** and locate CalculatorEngine.dll in *install_folder*\Practices\Mod09\Mod09_02\Starter\CalculatorEngine\bin\Debug. Click **Open** and then click **OK**.
5. Open the CalcUI.cs source file.
6. Show TODO comments in the Task List.

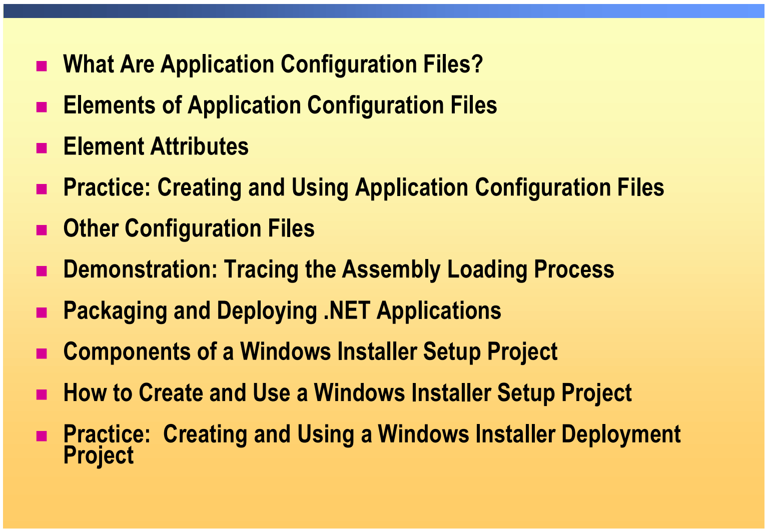
   To show TODO comments, click the **View** menu, point to **Show Tasks**, and then click **All**.
7. Locate the TODO comment. Add a **using** directive to access Calculator.

```
using Calculator;
```

8. Build the WindowsCalculator application.

9. In Windows Explorer, rename all local copies of **CalculatorEngine.dll** to **OLD_Calculator.dll**. This would include *install_folder*\Practices\Mod09\Mod09_02\Starter\bin\debug, *install_folder*\Practices\Mod09\Mod09_02\Starter\CalculatorEngine\bin\Debug, and *install_folder*\Practices\Mod09\Mod09_02\Starter\CalculatorEngine\obj\Debug.
10. In Windows Explorer, browse to *install_folder*\Practices\Mod09\Mod09_02\Starter\bin\debug.
11. Run the application by double-clicking WindowsCalculator.exe. Notice that the WindowsCalculator application still runs even though a local copy of the CalculatorEngine.dll assembly could not be found. This is because the runtime found the correct assembly in the global assembly cache and loaded it.

# Lesson: Deploying Windows Forms Applications



- What Are Application Configuration Files?
- Elements of Application Configuration Files
- Element Attributes
- Practice: Creating and Using Application Configuration Files
- Other Configuration Files
- Demonstration: Tracing the Assembly Loading Process
- Packaging and Deploying .NET Applications
- Components of a Windows Installer Setup Project
- How to Create and Use a Windows Installer Setup Project
- Practice:  Creating and Using a Windows Installer Deployment Project

**Introduction**

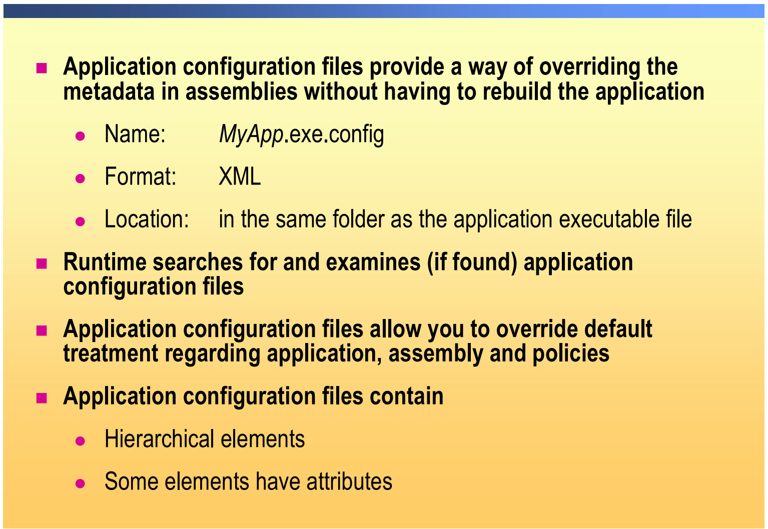
In this lesson, you will learn how to use application configuration files and Windows Installer 2.0 setup projects to deploy your.NET Framework applications.

**Lesson objectives**

After completing this lesson, you will be able to:

- Create and use application configuration files for.NET applications.
- Trace the assembly load process to aid in troubleshooting.
- Use Microsoft Windows Installer 2.0 to package and deploy .NET applications.

# What are Application Configuration Files?



- **Application configuration files provide a way of overriding the metadata in assemblies without having to rebuild the application**
  - Name: *MyApp*.exe.config
  - Format: XML
  - Location: in the same folder as the application executable file
- **Runtime searches for and examines (if found) application configuration files**
- **Application configuration files allow you to override default treatment regarding application, assembly and policies**
- **Application configuration files contain**
  - Hierarchical elements
  - Some elements have attributes

### Introduction

Every assembly and application is versioned in the .NET Framework. Application configuration files provide a way of overriding the metadata in assemblies without having to rebuild the application.

### Definition

When an application runs, the .NET Framework common language runtime searches for dependencies that the application needs, such as other assemblies and resource files. This is known as the probing process. As part of the probing process, the runtime searches for and examines (if found) application configuration files. If a file is found, the runtime modifies its probing behavior based on the contents of the configuration file.

### Definition

Application configuration files are XML files that contain settings specific to an application. This file contains configuration settings that the common language runtime reads (such as assembly binding policy) when it probes for the application's dependencies. Application configuration files allow you to override the application/assembly version, location of dependent assemblies, the probing process, and any publisher policy that a third-party vendor may have provided.

The application configuration file can redirect an application to use a different version of an assembly or look in a different location for an assembly, but the assembly itself must still be built with the same key pair that the application expects. This is because the runtime will check for tampering when a strong-named assembly is referenced. This helps prevent the use of redirection for malicious intentions, because the perpetrator would have to obtain the key pair to accomplish their task.

Application configuration files reside in the same folder as the executable to which they apply. The naming convention for an application configuration file is *executable_name*.config. For example, if your application is named MyApp.exe, the application configuration file must be named MyApp.exe.config.

### Example of an application configuration file

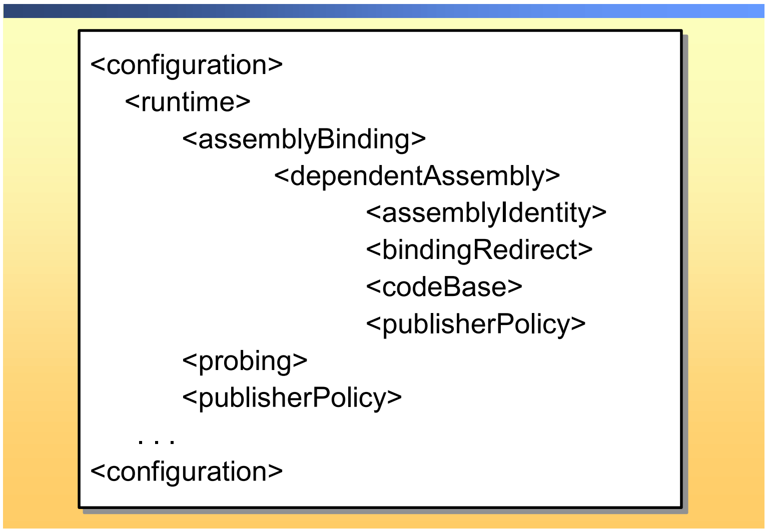
Application configuration files contain hierarchical elements and some of these elements can have attributes. Here is an example of a simple application configuration file.

```
<configuration>
  <runtime>
    <assemblyBinding xmlns="urn:schemas-microsoft-com:asm.v1">
      <dependentAssembly>
        <assemblyIdentity name="CalculatorEngine"
                          publicKeyToken="3a1061701aba2b5b"
                          culture="en-US"/>
          <bindingRedirect oldVersion="3.0.1.1"
                           newVersion="4.0.1.1"/>
          <codeBase version="4.0.1.1"
                    href="file:///C:/Program
Files/MyApplications/WindowsCalculator/CalculatorEngine.dll"/>
      </dependentAssembly>
    </assemblyBinding>
  </runtime>
</configuration>
```

**Note** The path for the <codeBase> element may differ from the one shown in the example configuration file, depending on where the referenced assembly is located.

Other topics in this lesson provide more information about configuration elements and element attributes.

# Elements of Application Configuration Files



```
<configuration>
   <runtime>
        <assemblyBinding>
                <dependentAssembly>
                        <assemblyIdentity>
                        <bindingRedirect>
                        <codeBase>
                        <publisherPolicy>
        <probing>
        <publisherPolicy>
    . . .
<configuration>
```

### Introduction

Configuration files are structured in a hierarchical fashion beginning with the root <configuration> element. Because the elements and sub elements are well-formed XML, they are case-sensitive.

### Application configuration file elements

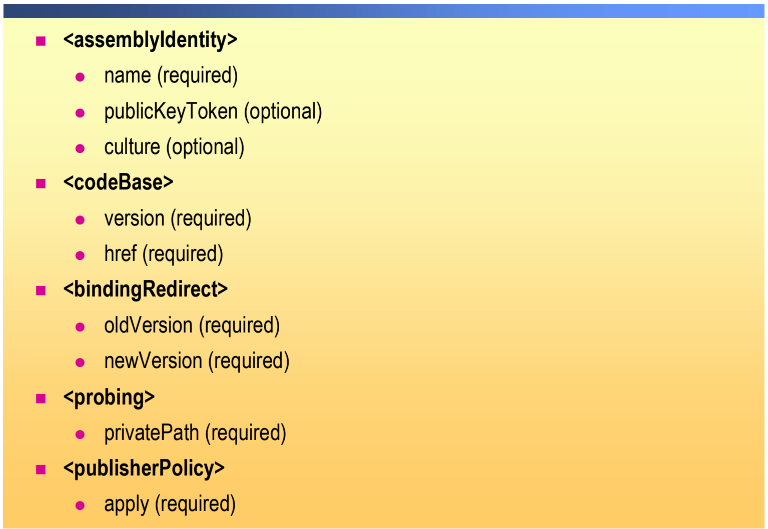
The table describes the elements that may be included in the configuration file.

| Element | Description |
|---|---|
| <configuration> | The root element of the configuration file, <configuration> indicates that the information that follows the element is used to configure the application. All configuration files must contain the starting and ending configuration elements. |
| <runtime> | A sub element of <configuration>, <runtime>contains information about assembly binding at run time. |
| <assemblyBinding> | A sub element of <runtime>, <assemblyBinding> contains information about assembly version redirection and locations of assemblies. |
| <dependentAssembly> | A sub element of <assemblyBinding>, <dependentAssembly> encapsulates binding policy information, such as the assembly name, the version number, and the assembly's location. This element contains the binding rules for an individual assembly. If binding policy is needed for multiple assemblies, use one <dependentAssembly> element for each assembly. |
| <assemblyIdentity> | A sub element of <dependentAssembly>, <assemblyIdentity> contains identifying information about the assembly. The identity information can include the simple name of the assembly, culture information if it is provided, and a token that maps to a public key. |

(*continued*)

| Element | Description |
|---|---|
| <bindingRedirect> | A sub element of <dependentAssembly>, <bindingRedirect> redirects the runtime from one assembly version to another. |
| <codeBase> | A sub element of <dependendAssembly>, <codeBase> specifies where the runtime can find a strong-named assembly. |
| <probing> | Specifies subdirectories of the application's base directory that might contain assemblies. |
| <publisherPolicy> | Specifies whether the runtime applies publisher policy. If placed within the <dependentAssembly> element, the <publisherPolicy> element will apply to the specific assembly. If placed within the <assemblyBinding> element, the <publisherPolicy> element will apply to all assemblies referenced by the application. |

# Element Attributes



- **<assemblyIdentity>**
  - name (required)
  - publicKeyToken (optional)
  - culture (optional)
- **<codeBase>**
  - version (required)
  - href (required)
- **<bindingRedirect>**
  - oldVersion (required)
  - newVersion (required)
- **<probing>**
  - privatePath (required)
- **<publisherPolicy>**
  - apply (required)

### Introduction

Some of the policy configuration elements include attributes that specify additional details for the runtime to use in locating referenced assemblies.

### Configuration file element attributes

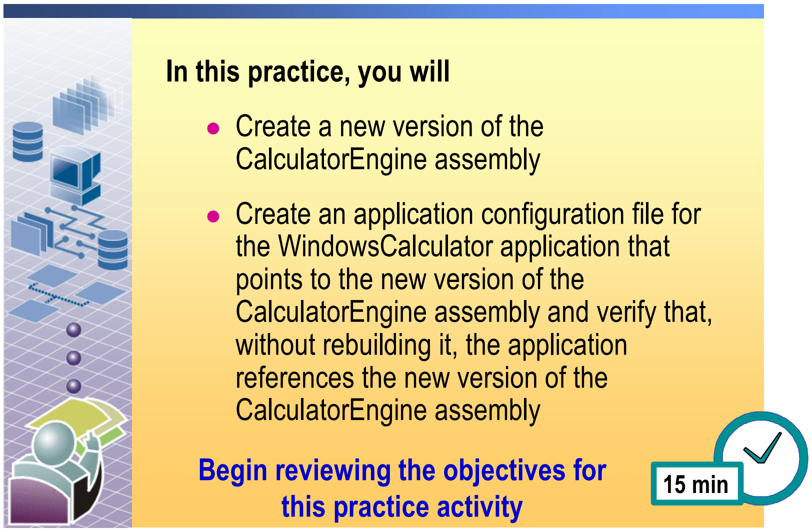
The table describes the attributes that can be used with certain configuration file elements.

| Element | Attribute | Description |
|---|---|---|
| <assemblyIdentity> | name: | (required)<br>The simple name of the assembly. |
| | publicKeyToken | (optional)<br>Value that specifies the strong name of the assembly. |
| | culture | (optional)<br>String that indicates the language and country/region of the assembly. |
| <codeBase> | version | (required)<br>Specifies the version of the assembly the codebase applies to. The format of version numbers in .NET is major.minor.build.revision. |
| | href | (required)<br>Specifies the URL where the assembly is stored. |

(***continued***)

| Element | Attribute | Description |
|---|---|---|
| <bindingRedirect> | oldVersion | (required)<br>Specifies the version or version range of the assembly that was originally requested. |
| | newVersion | (required)<br>Specified the version of the assembly to use instead of the originally requested version. |
| <probing> | privatePath | (required)<br>Contains the directories that the runtime should search for assemblies. The directories specified in privatePath must be subdirectories of the application base directory. |
| <publisherPolicy> | Apply | (required)<br>Indicates whether publisher policy is applied at runtime. Values are yes and no. |

# Practice: Creating and Using Application Configuration Files



**In this practice, you will**

- Create a new version of the CalculatorEngine assembly
- Create an application configuration file for the WindowsCalculator application that points to the new version of the CalculatorEngine assembly and verify that, without rebuilding it, the application references the new version of the CalculatorEngine assembly

Begin reviewing the objectives for this practice activity

15 min

### Introduction

In this practice, you will install a new version of the **CalculatorEngine** assembly in the global assembly cache. You will then build an application configuration file for the WindowsCalculator application that points the application to the new version of **CalculatorEngine**. You will not have to rebuild the WindowsCalculator application for it to use the new version of **CalculculatorEngine**.

### Instructions

► **Install a new version of the CalculatorEngine assembly into the global assembly cache**

1. Open the Visual Studio .NET Command Prompt window by clicking **Start**, pointing to **All Programs**, pointing to **Microsoft Visual Studio .NET**, pointing to **Visual Studio .NET Tools**, and then clicking **Visual Studio .NET Command Prompt**.
2. Change directories to *install_folder*\Practices\Mod09\Mod09_03\Starter.
3. Delete CalculatorEngine.dll.

   This is version 4.0.1.1, which is the assembly that WindowsCalculator currently references.
4. Copy **CalculatorEngine.dll.v5011** to **CalculatorEngine.dll**.
5. Install the **CalculatorEngine** assembly into the global assembly cache by typing the following command:

   ```
   gacutil -i CalculatorEngine.dll
   ```

   You should receive the message **Assembly successfully added to the cache**.

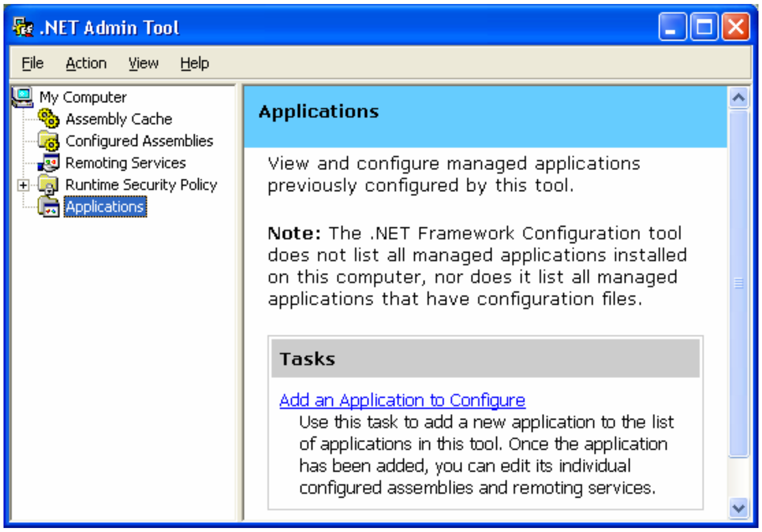
6. Display the contents of the global assembly cache by typing the following command:

```
gacutil -l
```

7. Scroll through the list to find **CalculatorEngine**. There should be two versions of **CalculatorEngine** in the global assembly cache. One version is 4.0.1.1 and the other is 5.0.1.1.

### ► Create an application configuration file for WindowsCalculator

1. Run the .NET Framework configuration tool (Mscorcfg.msc) by double-clicking the Mscorcfg.msc icon on the desktop.
2. Click the Applications node.
3. Click **Add an Application to Configure**.



Applications

View and configure managed applications previously configured by this tool.

**Note:** The .NET Framework Configuration tool does not list all managed applications installed on this computer, nor does it list all managed applications that have configuration files.

Tasks

Add an Application to Configure

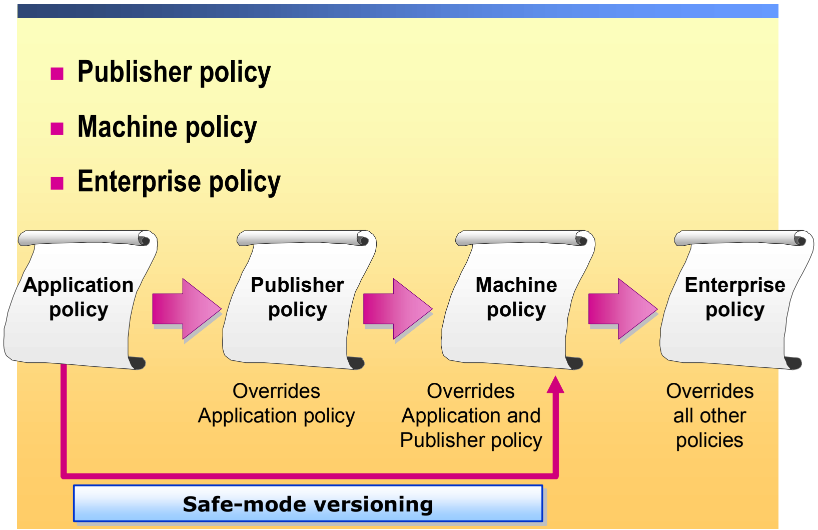
Use this task to add a new application to the list of applications in this tool. Once the application has been added, you can edit its individual configured assemblies and remoting services.

4. In the **Configure an Application** dialog box, click **Other** to locate the assembly.
5. Browse to *install_folder*\Practices\Mod09\Mod09_03\Starter, select **WindowsCalculator.exe**, and then click **Open**.
6. In Mscorcfg.msc, under Applications, expand the WindowsCalculator.exe application node, and then click **Configured Assemblies**.
7. In the right pane, click the **Configure an Assembly** link.
8. In the **Configure an Assembly** dialog box, select **Choose an assembly from the assembly cache**.
9. Click **Choose Assembly**.
10. In the **Choose Assembly from Assembly Cache** dialog box, scroll through the list of assemblies to locate CalculatorEngine version 5.0.1.1. Click **CalculatorEngine**, and then click **Select**.
11. Click **Finish**.

    The **CalculatorEngine Properties** dialog box appears.
12. Click the **Binding Policy** tab.
13. In the **Requested Version** column, type **4.0.1.1**

14. In the **New Version** column, type **5.0.1.1**
15. Click **Apply**, and then click **OK**.
16. Use Notepad to open *install_folder*\Practices\Mod09\Mod09_03\Starter\WindowsCalculator.exe.config and examine the contents. Notice the <bindingRedirect> tag. Close Notepad.
17. Run the application by double-clicking **WindowsCalculator.exe**. Verify that it uses the new version of the **CalculatorEngine** assembly. The version will be 5.0.1.1.

# Other Configuration Files



- Publisher policy
- Machine policy
- Enterprise policy

Application policy

Publisher policy

Machine policy

Enterprise policy

Overrides Application policy

Overrides Application and Publisher policy

Overrides all other policies

Safe-mode versioning

### Introduction

There are several other policy levels that can be applied to your application that can override the default binding policy and settings in the application configuration file.

### Publisher policy configuration files

Publisher policy configuration files are usually explicitly installed as part of a service pack or program update. For example, when using third-party components, the vendor may provide an updated version as a service pack. As part of the service pack, a publisher policy file is included to redirect applications to use the new version of the component. By default, publisher policy overrides application policy and the metadata that is found in the application's metadata. However, safe-mode versioning is allowed by configuring the application configuration file to ignore the publisher policy file. This is accomplished through the <publisherPolicy> element by setting the apply attribute to **no**.

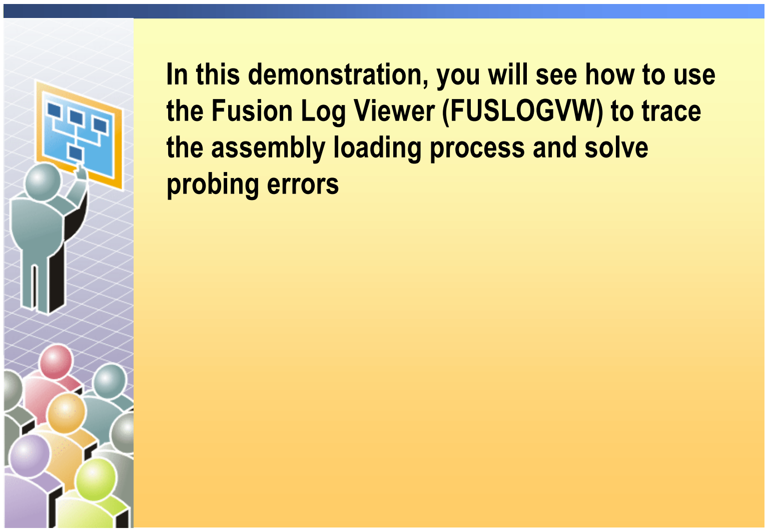
### Machine policy configuration file

The machine policy configuration file, Machine.config, contains settings that apply to an entire computer. This file is located in the %runtime install path%\Config directory. Putting the settings in the machine configuration file makes your system easier to maintain. For example, if you have a third-party component that both your client and server application uses, or that multiple applications use, it is easier to put the settings for that component in one place. In this case, the machine configuration file is the appropriate place for the settings, so you don't have the same settings in multiple different files. By default, machine policy overrides publisher policy and application policy.

### Enterprise policy configuration file

The enterprise policy level affects every computer and user on the network and can only be administered by enterprise or domain administrators. You might consider administering policy on this level when every person in your enterprise uses an application and you want to make sure that it always receives sufficient permissions to run. By default, enterprise policy overrides machine, publisher, and application policy.

# Demonstration: Tracing the Assembly Loading Process



In this demonstration, you will see how to use the Fusion Log Viewer (FUSLOGVW) to trace the assembly loading process and solve probing errors

In this demonstration, you will see how to use the Fusion Log Viewer (FUSLOGVW) to trace the assembly loading process and identify assembly loading problems (probing errors).

**Important** Prior to performing this demonstration, run the Fusion Log Viewer and enable the **Log Failures** checkbox. If you do not do so, the run time will ***not*** log probing errors and the following practice will not work as expected. It is also beneficial to leave tracking of log failures turned on at all times to help you in troubleshooting application problems.

## Instructions

### ► Run the WindowsCalculator application

1. In Windows Explorer, browse to *install_folder*\Democode\Mod09\Mod09_03.
2. Double-click **WindowsCalculator.exe** to run the application. Notice that the version number of the CalculatorEngine is 3.0.1.1.

### ► Using the configuration file

1. In Windows Explorer, delete **CalculatorEngine.dll** from *install_folder*\Democode\Mod09\Mod09_03.
2. Rename HoldWindowsCalculator.exe.config to WindowsCalculator.exe.config.
3. Run the application again.

   You should see a **System.IO.FileNotFoundException**.

### ► To use FUSLOGVW (Fusion Log Viewer) to solve probing errors

1. Open a Visual Studio .NET Command Prompt window.

   To open a Visual Studio .NET Command Prompt window, click **Start**, point to **All Programs**, point to **Microsoft Visual Studio .NET**, point to **Visual Studio .NET Tools**, and then click **Visual Studio .NET Command Prompt**.

2. Run the Fusion Log Viewer by typing the command:

```
fuslogvw
```

3. In the Assembly Binding Log Viewer window, locate the entry that matches the last execution of the WindowsCalculator application and double-click it.

4. Look through the log entries noting the various points at which the runtime attempted to locate the CalculatorEngine assembly and the order in which each attempt was made. The log will be similar to the following:

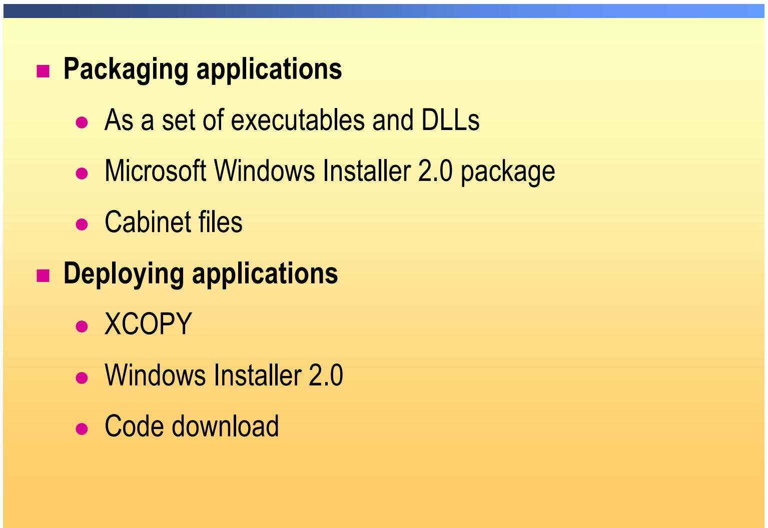
```
LOG: Processing DEVPATH.
LOG: DEVPATH is not set. Falling through to regular bind.
LOG: Publisher policy file is not found.
LOG: Host configuration file not found.
LOG: Using machine configuration file from
C:\WINDOWS\Microsoft.NET\Framework\v1.0.3705\config\
machine.config.
LOG: Post-policy reference: CalculatorEngine,
Version=3.0.1.1, Culture=neutral,
PublicKeyToken=3d117e8f75e2ae71
LOG: Cache Lookup was unsuccessful.
LOG: Attempting download of new URL
file:///C:/Courses/2555A/CD_Trainer/StudentCD/Democode/
Mod09/Demo09_03/CalculatorEngine.DLL.
LOG: Attempting download of new URL
file:///C:/Courses/2555A/CD_Trainer/StudentCD/Democode/
Mod09/Demo09_03/CalculatorEngine/CalculatorEngine.DLL.
LOG: Attempting download of new URL
file:///C:/Courses/2555A/CD_Trainer/StudentCD/Democode/
Mod09/Demo09_03/CalculatorEngine.EXE.
LOG: Attempting download of new URL
file:///C:/Courses/2555A/CD_Trainer/StudentCD/Democode/
Mod09/Demo09_03/CalculatorEngine/CalculatorEngine.EXE.
LOG: All probing URLs attempted and failed.
```

### ► Run the executable

1. Rename **CalculatorEngine.dll.v4.0.1.1** to **CalculatorEngine.dll**.

2. Run the WindowsCalculator application once again. Notice that the version number of **CalculatorEngine** is now 4.0.1.1.

# Packaging and Deploying .NET Applications



- **Packaging applications**
  - As a set of executables and DLLs
  - Microsoft Windows Installer 2.0 package
  - Cabinet files
- **Deploying applications**
  - XCOPY
  - Windows Installer 2.0
  - Code download

**Introduction**

You have several options for how you package and deploy your .NET applications.

**Definitions**

*Packaging* is the act of creating a package that can install your application onto the user's computer.

*Deployment* is the act of distributing a built application to the computer and setting up the application so it will run correctly.

**Packaging Options**

This table compares packaging options available for .NET applications.

| Packaging Option | Description | Comments |
|---|---|---|
| Executables and DLLs | A set of executables and DLLs in the original folder hierarchy in which the application was built. | No packaging is required with .NET applications and assemblies. You can use the EXE and DLL files as they were built. |
| Windows Installer 2.0 | Use Visual Studio .NET to create .msi files for use with the Windows Installer. | This is the standard way of distributing and installing applications that run on the desktop. Other applications can also be deployed by using .msi files. For example, Microsoft ASP.NET applications can be packaged in an .msi file. |

**(*continued*)**

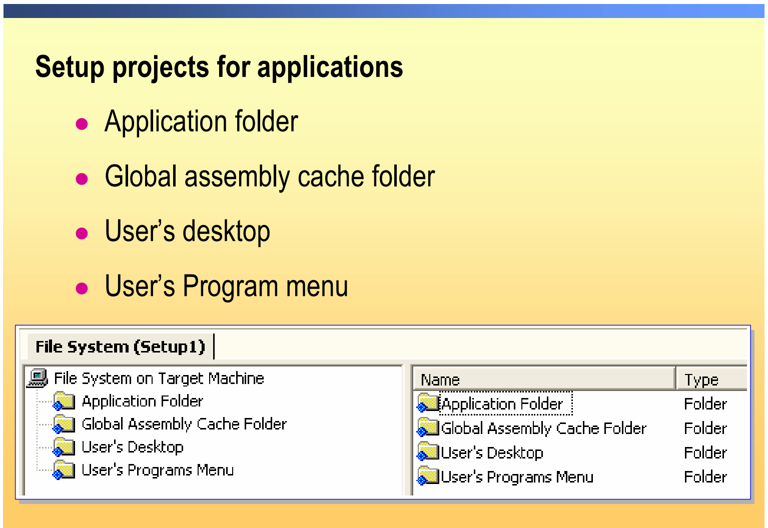
| Packaging Option | Description | Comments |
|---|---|---|
| Cabinet Files | Use Visual Studio .NET to create cabinet (.cab) files for deployment of a .NET Framework application. | The following restrictions apply when creating the .cab file for .NET compatible applications:<br>Only one assembly can be stored in a .cab file. The .cab file must have the same name as the file in the assembly that contains the manifest. For example, if the file containing the manifest in the assembly is called MyClasses.dll, then the .cab file must be named MyClasses.cab.<br>After you have created a .cab file, it can be downloaded by specifying its location by using the <codeBase> element in one of the configuration files. |

## Deployment Options

This table compares the deployment options available for .NET applications.

| Deployment Option | Description | Comments |
|---|---|---|
| XCOPY | Because common language runtime applications are self-describing and require no registry entries, you can use XCOPY to copy the application to an appropriate directory. The application can then be run from that directory. You can also use FTP to deploy your application. | Easiest way to install an application.<br>To remove the application from the computer, just delete for directory structure for the application.<br>Installs private and strong-named assemblies only. No shared assemblies can be installed this way. |
| Windows Installer 2.0 | Windows Installer 2.0 can install, repair, or remove Microsoft .NET Framework assemblies in the GACand in private directories. This is the recommended way to deploy Windows Forms Applications. | Use a Visual Studio .NET setup project to build an .msi file.<br>Installation, repair, and removal of assemblies in the global assembly cache.<br>Install, repair, or remove assemblies in private locations designated for particular applications.<br>Rollback unsuccessful installations, repairs, or removals of assemblies.<br>Install-on-demand strong-named assemblies in the GACand in private locations designated for particular applications. |
| Code Download | If you are distributing your application over the Internet or through a corporate intranet, you can download the code to a computer and run the application. | |

# Components of a Windows Installer Setup Project



**Setup projects for applications**

- Application folder
- Global assembly cache folder
- User's desktop
- User's Program menu

File System (Setup1)

- File System on Target Machine
  - Application Folder
  - Global Assembly Cache Folder
  - User's Desktop
  - User's Programs Menu

| Name | Type |
|---|---|
| Application Folder | Folder |
| Global Assembly Cache Folder | Folder |
| User's Desktop | Folder |
| User's Programs Menu | Folder |

## Introduction

By using a Windows Installer setup project you can customize how the application is deployed on the user's computer.

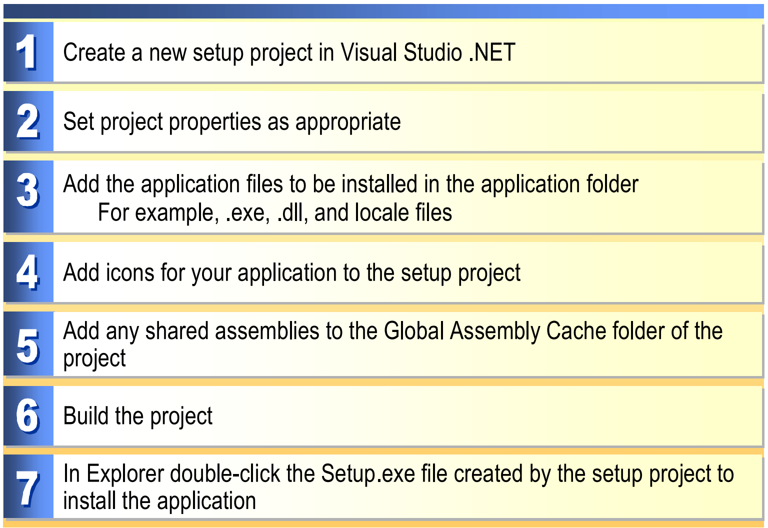
**Note** Remember to add files from a Release build, not a Debug build, to setup projects.

## Components of a Windows Installer setup project

This table describes the uses for the various components of a Windows Installer setup project.

| Setup Component | Description |
|---|---|
| Application folder | The folder used to store the application, for example, C:\Program Files\Manufacturer\Product. |
| Global Assembly Cache Folder | Use this folder to install shared assemblies. When creating a new setup project, the Global Assembly Cache Folder will not appear in the File Editor by default. To show the Global Assembly Cache Folder, right-click **File System on Target Machine**, highlight **Add Special Folder**, and select **Global Assembly Cache Folder**. |
| User's Desktop | Use this folder to install shortcuts on the desktop. |
| User's Program Menu | Use this folder to install a shortcut in the program menu. |

# How to Create and Use a Windows Installer Setup Project



1 Create a new setup project in Visual Studio .NET

2 Set project properties as appropriate

3 Add the application files to be installed in the application folder
For example, .exe, .dll, and locale files

4 Add icons for your application to the setup project

5 Add any shared assemblies to the Global Assembly Cache folder of the project

6 Build the project

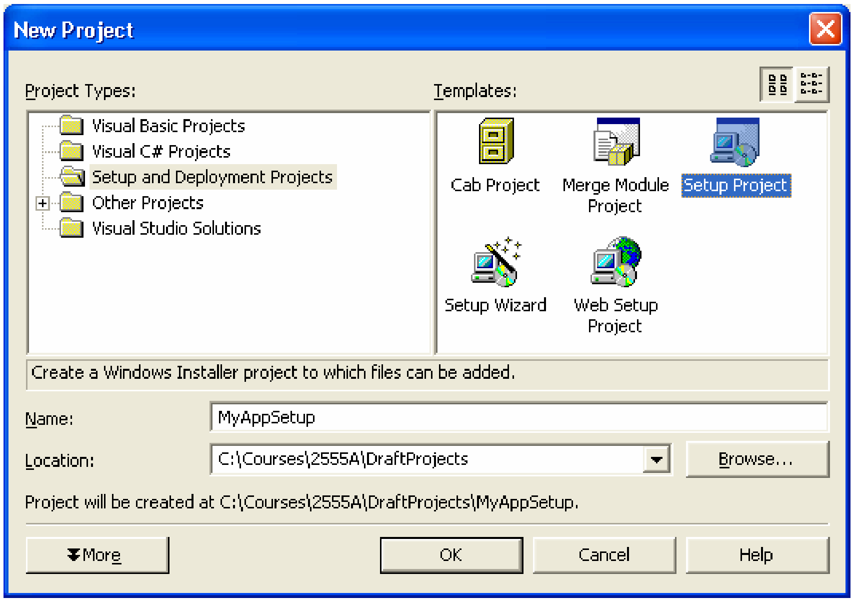
7 In Explorer double-click the Setup.exe file created by the setup project to install the application

### Introduction

You can use Visual Studio .NET to create a Windows Installer setup project for your application. You can then copy the .msi file to the appropriate computer and double-click the .msi file from the setup project to install your application.

### Procedure: Creating and using a Windows Installer setup project

1. Create a new setup project in Visual Studio .NET.



New Project

Project Types: Setup and Deployment Projects

Templates: Setup Project

Create a Windows Installer project to which files can be added.

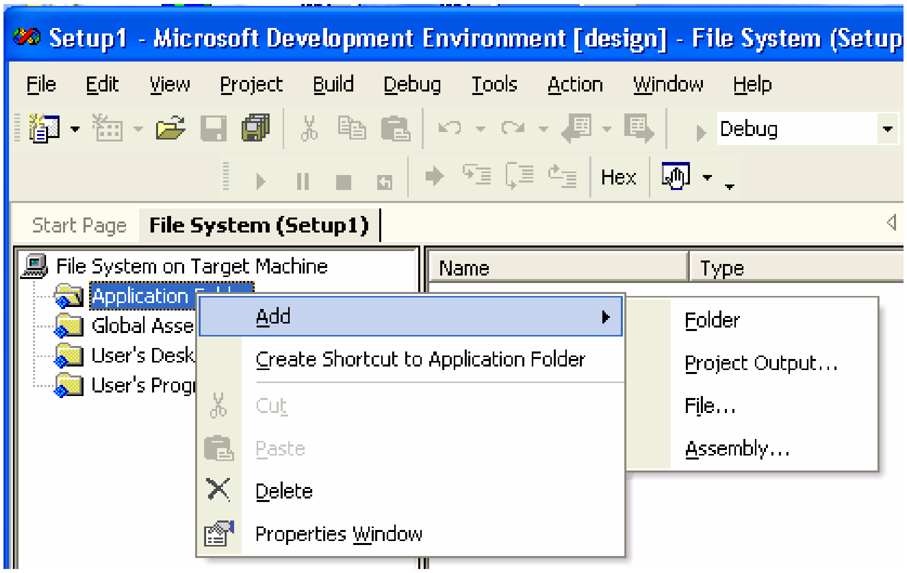
Name: MyAppSetup

Location: C:\Courses\2555A\DraftProjects

Project will be created at C:\Courses\2555A\DraftProjects\MyAppSetup.

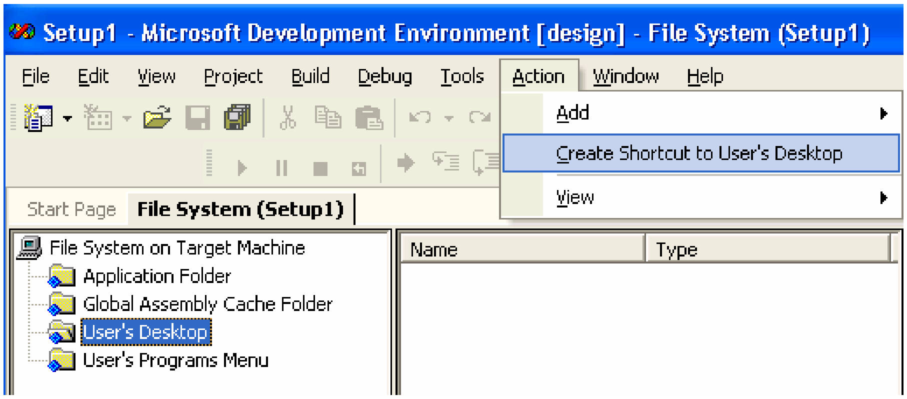
2. Click **OK** and then set project properties such as **Author**, **Description**, **Manufacturer**, **ManufacturerUrl**, **ProductName**, **Title**, and **Version**.

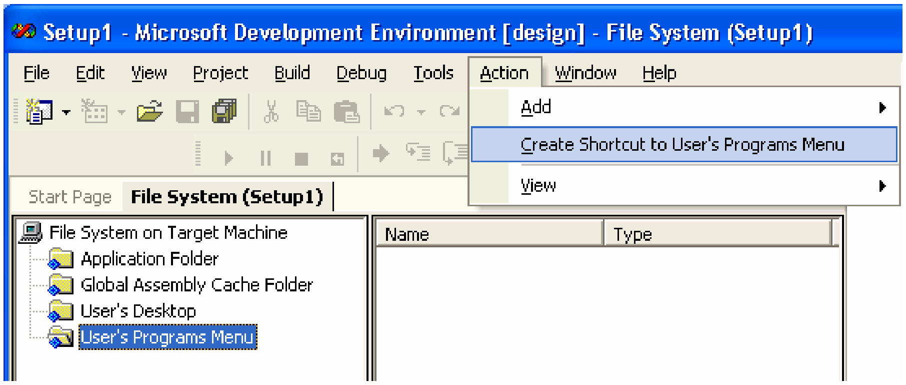
3. To add the application files to be installed in the application folder, in the **File System** tree, right-click the **Application Folder**, click **Add**, and then click **Folder** or **File**, as appropriate.


Setup1 - Microsoft Development Environment [design] - File System (Setup
File Edit View Project Build Debug Tools Action Window Help
Debug
Hex
Start Page File System (Setup1)
File System on Target Machine
Application Folder
Global Asse
User's Desk
User's Progr
Name
Type
Add
Create Shortcut to Application Folder
Cut
Paste
Delete
Properties Window
Folder
Project Output...
File...
Assembly...

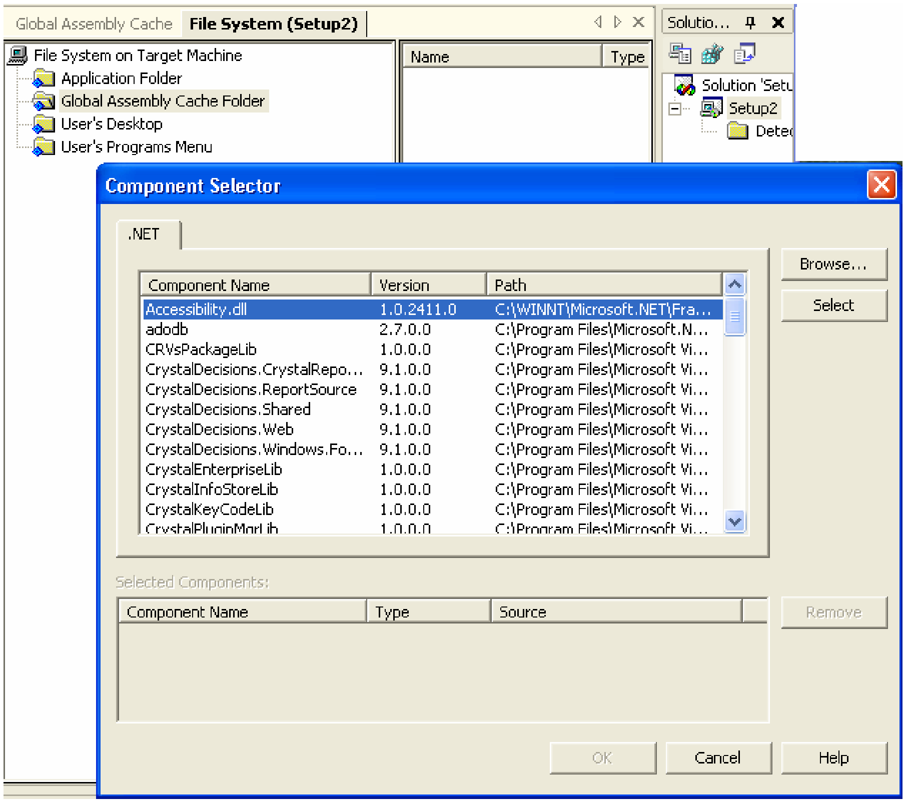
4. Set the **DefaultLocation** property of the Application Folder if you wish to change the location of where the contents of the Application Folder will be installed.

5. To add the icons for your application to the setup project, in the **File System** tree, right-click **User's Desktop** or **User's Program Menu**, and choose **Create Shortcut to User's Desktop** or **Create Shortcut to User's Programs Menu**, as appropriate.


Setup1 - Microsoft Development Environment [design] - File System (Setup1)
File Edit View Project Build Debug Tools Action Window Help
Add
Create Shortcut to User's Desktop
View
Start Page File System (Setup1)
File System on Target Machine
Application Folder
Global Assembly Cache Folder
User's Desktop
User's Programs Menu
Name
Type


Setup1 - Microsoft Development Environment [design] - File System (Setup1)
File Edit View Project Build Debug Tools Action Window Help
Add
Create Shortcut to User's Programs Menu
View
Start Page File System (Setup1)
File System on Target Machine
Application Folder
Global Assembly Cache Folder
User's Desktop
User's Programs Menu
Name
Type

6. To add any shared assemblies to the global assembly cache at setup time (these are strong-named assemblies that will be shared among multiple applications), in the **File System** tree, right-click the **File System on Target Machine** root node, click **Add Special Folder**, and then click **Global Assembly Cache Folder**. Right-click the **Global Assembly Cache Folder**, point to **Add**, and then click **Assembly**. Use the Component Selector to browse and a select any strong-named assemblies you wish to add to the global assembly cache.



| Component Name | Version | Path |
|---|---|---|
| Accessibility.dll | 1.0.2411.0 | C:\WINNT\Microsoft.NET\Fra... |
| adodb | 2.7.0.0 | C:\Program Files\Microsoft.N... |
| CRVsPackageLib | 1.0.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalDecisions.CrystalRepo... | 9.1.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalDecisions.ReportSource | 9.1.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalDecisions.Shared | 9.1.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalDecisions.Web | 9.1.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalDecisions.Windows.Fo... | 9.1.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalEnterpriseLib | 1.0.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalInfoStoreLib | 1.0.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalKeyCodeLib | 1.0.0.0 | C:\Program Files\Microsoft Vi... |
| CrystalPluginMgrLib | 1.0.0.0 | C:\Program Files\Microsoft Vi... |

| Component Name | Type | Source |
|---|---|---|

7. Build the setup project.

   The output is an .msi file.

8. In Windows Explorer, locate and double-click **Setup.exe** to install the application on your computer.

**Important** By default, the .NET framework redistributable library is not included in the deployment project. This library can be added to the deployment project in the event the target machines do not have the .NET Framework already installed. To add the redistributable library, expand the **Detected Dependencies** list in Solution Explorer, right click **dotnetfxredist_x86_enu.msm**, and clear the **Exclude** option on the shortcut menu.

# Practice: Creating and Using a Windows Installer Deployment Project

In this practice, you will create and test a Windows Installer deployment project for the WindowsCalculator application

Begin reviewing the objectives for this practice activity

**Introduction**

In this practice, you will create a Microsoft Installer setup project and add the WindowsCalculator application to the project. You will then test the project to validate that the application installed and works correctly.

**Instructions**

► **Create the setup project**

1. Start Visual Studio .NET.
2. On the **Start** page, click **New Project**.
3. In the New Project window, in the Project Type pane, click **Setup and Deployment Projects**.
4. In the Templates pane, click **Setup Project**.
5. Set the Location to *install_folder*\Practices\Mod09\Mod09_04\Starter.
6. Set the Name to **WinCalc**. Click **OK** to create the project.
7. In the File System tree, right-click **File System on Target Machine**, click **Add Special Folder**, and then click **Global Assembly Cache Folder**.
8. Right-click **Global Assembly Cache Folder**, click **Add**, and then click **File**.
9. In the **Add Files** dialog box, change folder to install_folder\Practices\Mod09\Mod09_04\Starter, and then double-click **CalculatorEngine.dll**.
10. Right-click **Application Folder**, click **Add**, and then click **File**.
11. In the **Add Files** dialog box, double-click **WindowsCalculator.exe**.
12. Right-click **Application Folder**, click **Add**, and then click **File**.
13. In the **Add Files** dialog box, double-click **KEYS03.ICO**.

14. Right-click the **Application Folder**, and then click **Properties Window**. In the Properties window, change the **DefaultLocation** property to **[ProgramFilesFolder]\[ ProductName]** by removing the Manufacturer entry. Also, change the **AlwaysCreate** property to **True**.
15. In Solution Explorer, expand **Detected Dependencies**. In the **Detected Dependencies** folder, right-click **CalculatorEngine.dll**, and then click **Exclude**.
16. To create a shortcut for the User's Desktop, select the **Application Folder** again, and then click **WindowsCalculator.exe**.

    When you reach step 24, you will be directed to repeat steps 16 through 22 to create a second shortcut for the **Program** menu.
17. On the **Action** menu, click **Create Shortcut to WindowsCalculator.exe**. Rename the shortcut to **Windows Calculator**.
18. In the Properties window, make sure the **Target** property is set to **WindowsCalculator**.
19. Click on the **Icon** property and select **Browse**.
20. In the **Icon** dialog box, click **Browse**.
21. Select **Application Folder** from the **Look In** dropdown list and click on **KEYS03.ICO**.
22. Click **OK** and then click **OK** in the **Icon** dialog box.
23. Drag the shortcut from the Application Folder to the User's Desktop node.
24. To create a shortcut for the **Program** menu, repeat steps 16 through 22 again to create another shortcut with the same name.
25. Drag the shortcut to the User's Program Menu node.
26. In Solution Explorer, click the **WinCalc** project. In the Properties window, set the following properties to their associated values.

| Property | Value |
|---|---|
| **Author** | *Your Name* |
| **Manufacturer** | *Your Name* or *Your Company's Name* |
| **ProductName** | WinCalc |
| **Title** | Windows Calculator |

27. Save and build the setup project.

### ► Install and run the application

1. In Windows Explorer, browse to *install_folder*\Practices\Mod09\Mod09_04\Starter\WinCalc\Debug.
2. Double-click **Setup.exe** to start the installation of the Windows Calculator application.
3. Click **Next** three times. The application will be installed on the computer.
4. Click **Close** after the application has been installed.
5. View the computer's desktop and double-click the **Windows Calculator** icon.

   The application should start and show that it is using version 6.0.1.1 of the **CalculatorEngine** assembly.

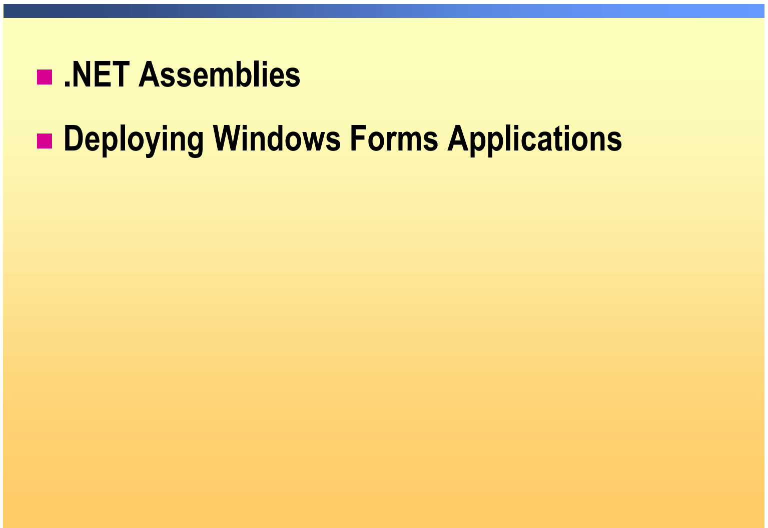
6. Close the Calculator.
7. Click the **Start** button, point to **All Programs**, and then click the **Windows Calculator**.

   The application should run and show that it, too, is using version 6.0.1.1. of the **CalculatorEngine** assembly.
8. Close the Calculator.
9. In Windows Explorer, browse to C:\Program Files\WinCalc.

   You should see the WindowsCalculator.exe file in this folder, but not CalculatorEngine.dll.
10. In Windows Explorer, change directories to C:\WINDOWS\assembly.

    You should see CalculatorEngine version 6.0.1.1 installed in the global assembly cache.

# Review



- .NET Assemblies
- Deploying Windows Forms Applications

1. Describe how private assemblies and strong-named assemblies differ.

   **Private assemblies are referenced exclusively by their simple names. Private assemblies are installed in the same folder hierarchy as the application that references the assembly. Private assemblies are not shared among applications, nor are they versioned.**

   **Strong-named assemblies are referenced by their simple names, versions, cultures, digital signatures, and public keys. Strong-named assemblies can be installed in the same folder hierarchy as the application that references the assembly, in any folder on a computer, on a remote computer, or on a Web site. Strong-named assemblies can be shared across multiple applications and they are versioned.**

2. Describe how strong-named assemblies and shared assemblies differ.

   **Shared assemblies are strong-named assemblies that have been installed in the global assembly cache. Multiple versions of an assembly can be installed in the global assembly cache (side-by-side versioning) and the assembly is verified at installation time rather than at runtime.**

3. What is needed to build a strong-named assembly?

   **A private/public key pair is needed to build a strong-named assembly. For testing purposes, the sn.exe utility can be used to generate this key pair. In addition, an attribute must be added to the AssemblyInfo file to indicate where Visual Studio .NET can locate the key pair.**

4. How is an assembly installed in the global assembly cache?

   **There are three methods for installing an assembly in the global assembly cache.**

   - **GACUtil.exe**
   - **Mscorcfg.msc**
   - **Windows Installer 2.0**

5. Why would you use an application configuration file?

   **A component that an application uses has been upgraded to a newer version. For the application to load the newer version of the component, an application file is necessary fir redirecting the runtime to the newer version, thus overriding the metadata found in the application. An alternative to this is to rebuild the application. Application configuration files allow upgrades without having to rebuild the application.**

6. You want to redirect the runtime to load version 2.3.4.56 of an assembly that an application needs from D:\CustomAssemblies\MyAssembly.dll. What entry needs to be made in the application configuration file?

   **<codeBase version="2.3.4.56" href="file:///D:/CustomAssemblies/MyAssembly.dll"/>**

7. When would a publisher policy configuration file be used?

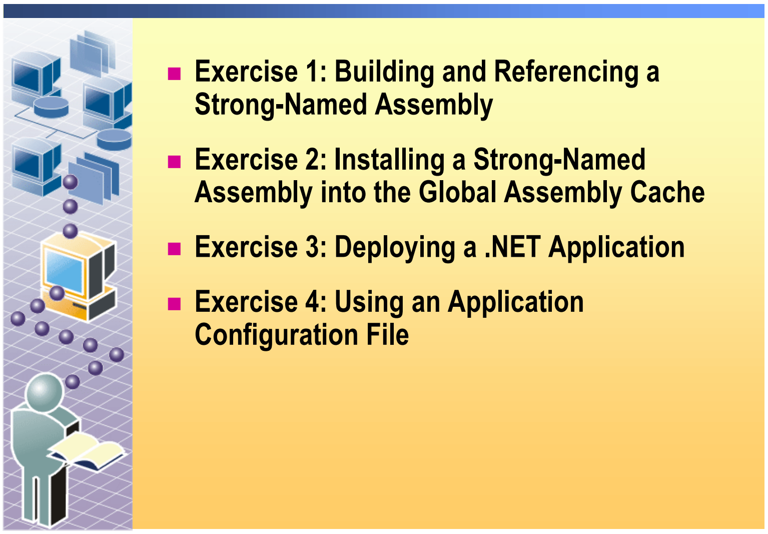
   **Publisher policy configuration files are typically used by third-party vendors when they need to upgrade their component to a newer version. Included in the upgrade is a publisher policy file that redirects the runtime to the newer version of component. Application configuration files can override publisher policy through safe-mode versioning.**

8. List the three methods of deploying an application.

   **Three methods for deploying an application are:**

   - **Copying the application to a folder on the target machine through the use of XCOPY.**
   - **Using Windows Installer.**
   - **Using .cab files.**

# Lab 9.1: Deploying an Application



- Exercise 1: Building and Referencing a Strong-Named Assembly
- Exercise 2: Installing a Strong-Named Assembly into the Global Assembly Cache
- Exercise 3: Deploying a .NET Application
- Exercise 4: Using an Application Configuration File

**Objectives**

After completing this lab, you will have demonstrated your ability to:

- Build a strong-named assembly.
- Build an application that references a strong-named assembly.
- Install a strong-named assembly in the global assembly cache.
- Deploy an application by using Windows Installer.
- Use an application configuration file to cause an application to use a new version of a referenced assembly.

**Prerequisites**

Before working on this lab, you must have:

- The knowledge and skills to develop a simple Windows Forms application by using a Visual Studio .NET–compatible programming language.
- The knowledge and skills to debug an application by using Visual Studio .NET.
- The knowledge and skills to build strong-named assemblies and applications that reference strong-named assemblies.
- The knowledge and skills to create a .NET setup project to deploy an application.
- The knowledge and skills to create an application configuration file for an application.

**Scenario**

Many applications that you create for the .NET Framework will require the use of assemblies that are not included with the application assembly. The referenced assembly may be installed into the global assembly cache, or it may reside in a location other than the application's folder.

In this lab, you will create a strong-named assembly and then reference this assembly from an application. You will then deploy the application by using a Setup and Deployment project to build a Windows Installer file. In addition, you will upgrade the strong-named assembly to a new version and modify the application to use the new assembly through the use of application configuration files.

**Estimated time to complete this lab: 30 minutes**

# Exercise 1
# Building and Referencing a Strong-Named Assembly

In this exercise, you will use the Expense Reporting application to build a strong-named assembly. You will then use the Internal Business Application shell application to reference the strong-named assembly. You will also use ILDASM to verify that ExpenseReport.DLL is a strong-named assembly.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab09_1\Ex01\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab09_1\Ex01\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open the ExpenseReport project in Visual Studio .NET. Browse to *install_folder*\Labfiles\Lab09_1\Ex01\Starter\ExpenseReportApp to find the project files. | **a.** For more information about opening a project file, see the following resource:<br>• The Visual Studio .NET Help documentation. In Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. |
| **2.** Build the expense report project. Keep this instance of Visual Studio .NET open. | **a.** For more information about building an application, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Preparing and Managing Builds**. |
| **3.** Open the InternalBusinessApp project in another instance of Visual Studio.NET. Browse to *install_folder*\Labfiles\Lab09_1\Ex01\Starter\Business Application Shell to find the project files. | **a.** For more information about opening a project file, see the following resource:<br>• The Visual Studio .NET Help documentation. In Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. |
| **4.** Add a reference to the ExpenseReport DLL file. | **a.** For more information about how to add a reference to an assembly by using Visual Studio .NET, see the following resource:<br>• Practice: Working with the Global Assembly Cache in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. |
| **5.** Build and test the internal business application. Keep this instance of Visual Studio .NET open. | No additional information is necessary for this task. |
| **6.** Open a Visual Studio .NET Command Prompt window and change directories to the location of InternalBusinessApp.exe. Browse to *install_folder*\Labfiles\Lab09_1\Ex01\Starter\Business Application Shell\bin\Debug to find the executable. | **a.** For more information about how to open a Visual Studio .NET Command Prompt window, see the following resource:<br>• Practice: Calling a Strong-Named Assembly in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. |

| Tasks | Additional information |
|---|---|
| **7.** Run ILDASM, and view the information for InternalBusinessApp.exe. View the MANIFEST, and note that the reference to the **ExpenseReport** assembly does not include a public key token. Close ILDASM. | **a.** For more information about how to use ILDASM, see the following resources:<br>• Practice: Calling a Strong-Named Assembly in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. Search by using the phrase **MSIL Disassembler (Ildasm.exe)**. |
| **8.** In a Visual Studio .NET Command Prompt window, change directories to where the ExpenseReport project is located. Browse to *install_folder*\Labfiles\Lab09_1\Ex01\Starter\ExpenseReportApp to find the project. | **a.** For more information about how to open a command prompt window in Visual Studio .NET, see the following resource:<br>• Practice: Calling a Strong-Named Assembly in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. |
| **9.** Run the strong-name tool to generate a strong-name key pair file.<br>• Name the strong-name key pair file **ExpenseReport.snk**. | **a.** For more information about SN.EXE, see the following resource:<br>• The .NET Framework SDK documentation. Search by using the phrases **Strong Name Tool (Sn.exe)** and **Creating a Key Pair**. |
| **10.** Switch to the Visual Studio .NET instance in which the ExpenseReport project is open.<br>**a.** Complete the AssemblyKeyFile attribute to reference the ExpenseReport.snk key pair file.<br>**b.** Rebuild the ExpenseReport project. | **a.** For more information about how to use ILDASM, see the following resource:<br>• Practice: Calling a Strong-Named Assembly in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>**b.** For more information about global attributes and the AssemblyInfo file, see the following resource:<br>• The .NET Framework SDK documentation. Search by using the phrase **Global Attributes**. |
| **11.** Switch to the Visual Studio .NET instance in which the InternalBusinessApp project is open.<br>**a.** Rebuild and run the application.<br>**b.** In a Visual Studio .NET Command Prompt window, run ILDASM to view the information for InternalBusinessApp.exe.<br>**c.** View the MANIFEST. Notice that the reference to the ExpenseReport assembly now includes a public key token. | **a.** For more information about how to use ILDASM, see the following resources:<br>• Practice: Calling a Strong-Named Assembly in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. Search by using the phrase **MSIL Disassembler (Ildasm.exe)**. |

# Exercise 2
# Installing a Strong-Named Assembly into the Global Assembly Cache

In this exercise, you will update the **ExpenseReport** assembly to version 2.0.1.1 and install it into the global assembly cache. You will then build and run the Internal Business Application to reference the **ExpenseReport** assembly and use ILDASM to verify that the Internal Business Application is referencing the correct version of the **ExpenseReport** assembly.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab09_1\Ex02\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab09_1\Ex02\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open the ExpenseReport project in Visual Studio .NET. Update the AssemblyVersion attribute to **"2.0.1.1"** and build the expense report project. Browse to *install_folder*\Labfiles\Lab09_1\Ex02\Starter\ExpenseReportApp to find the project. | **a.** For more information about opening a project file, see the following resource:<br>• The Visual Studio .NET Help documentation. In Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. |
| **2.** Open the InternalBusinessApp project in another instance of Visual Studio.NET. Browse to *install*_folder\Labfiles\Lab09_1\Ex02\Starter\Business Application Shell to find the project. | **a.** For more information about opening a project file, see the following resource:<br>• The Visual Studio .NET Help documentation. In Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. |
| **3.** Add a reference to the ExpenseReport DLL file. Browse to *install_folder*\Labfiles\Lab09_1\Ex02\Starter\ExpenseReportApp\bin\Debug to find the DLL. | **a.** For more information about how to add a reference to an assembly using Visual Studio .NET, see the following resource:<br>• Practice: Working with the Global Assembly Cache in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. |
| **4.** In Solution Explorer, beneath **References**, click the **ExpenseReport** entry.<br>**a.** Change the Copy Local property to False. This prevents Visual Studio .NET from copying the ExpenseReport.dll file into the debug folder of the Internal Business Application.<br>**b.** Build the Internal Business Application. | No additional information is necessary for this task. |

| Tasks | Additional information |
|---|---|
| **5.** Open a command prompt window in Visual Studio .NET, and change directories to the location where ExpenseReport.dll is located. Browse to *install_folder*\Labfiles\Lab09_1\ Ex02\Starter\ExpenseReportApp\bin\Debug to find the file. | **a.** For more information about how to open a command prompt window in Visual Studio .NET, see the following resource:<br>• Practice: Calling a Strong-Named Assembly in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. |
| **6.** Run GACUTIL to install ExpenseReport.dll into the global assembly cache.<br>• View the global assembly cache after installing the ExpenseReport.dll assembly. There should be an entry for ExpenseReport in the global assembly cache. | **a.** For more information about how to use GACUTIL, see the following resource:<br>• Practice: Working with the Global Assembly Cache in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. |
| **7.** Run the Internal Business Application. | No additional information is necessary for this task. |
| **8.** Open a Visual Studio .NET Command Prompt window and change directories to the location where InternalBusinessApp.exe is located. Browse to *install_folder*\Labfiles\Lab09_1\ Ex02\Starter\Business Application Shell\bin\Debug to find the executable file. | **a.** For more information about how to open a Visual Studio .NET Command Prompt window, see the following resource:<br>• Practice: Calling a Strong-Named Assembly in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. |
| **9.** Run ILDASM, and view InternalBusinessApp.exe.<br>• View the MANIFEST. Notice that the reference to the **ExpenseReport** assembly is now to version 2.0.1.1. | **a.** For more information about how to use ILDASM, see the following resources:<br>• Practice: Calling a Strong-name Assembly in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK documentation. Search by using the phrase **MSIL Disassembler (Ildasm.exe)**. |

# Exercise 3
# Deploying a .NET Application

In this exercise, you will package the Expense Reporting application into a Windows Installer file by using a Setup and Deployment project. You will then install the application on the local computer.

**Note** The Internal Business Application component of the Expense Reporting application requires that the Expense Report XML Web service be installed on the target computer. This exercise assumes that the Expense Report XML Web service has already been installed on the target computer.

There are solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab09_1\Ex03\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open the InternalBusinessApp project in Visual Studio .NET. Browse to *install_folder*\Labfiles\Lab09_1\Ex03\Starter\Business Application Shell to find the project files. | **a.** For more information about opening a project file, see the following resource:<br>• The Visual Studio .NET Help documentation. In Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. |
| **2.** Build the InternalBusinessApp project. Close the InternalBusinessApp solution, but keep Visual Studio .NET open. | **a.** For more information about building an application, see the following resource:<br>• The Visual Studio .NET Help documentation. Search by using the phrase **Preparing and Managing Builds**. |
| **3.** Create a new Setup and Deployment project. Use the template that will create a Windows Installer setup project.<br>• Set the project name to InternalBusinessApplication.<br>• Set the project location to *install_folder***Labfiles\Lab09_1\Ex03\Starter**. | **a.** For more information about how to create a new Setup and Deployment project, see the following resource:<br>• Topic: How to Create and Use a Windows Installer Setup Project in Module 9, “Deploying Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. |

| Tasks | Additional information |
|---|---|
| **4.** Add the global assembly cache folder to the project. | **a.** For more information about how to add the global assembly cache folder, see the following resource:<br>• Practice: Creating and Using a Windows Installer Deployment Project in Module 9, “Deploying Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*<br>**b.** For more information about deploying applications, see the following resources:<br>• Lesson: Deploying Windows Forms Applications in Module 9, “Deploying Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*<br>• The .NET Framework SDK. Search by using the phrase **Deploying Applications**. |
| **5.** Add the localized folders to the Application folder.<br>**a.** Right-click Application folder, point to **Add**, and select **Folder**.<br>**b.** Create folders with the following names: de, de-DE, en, en-US, fr, fr-FR, ja, and ja-JP. | **a.** For more information about how to add the Application folder, see the following resource:<br>• Practice: Creating and Using a Windows Installer Deployment Project in Module 9, “Deploying Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*<br>**b.** For more information about deploying applications, see the following resources:<br>• Lesson: Deploying Windows Forms Applications in Module 9, “Deploying Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*<br>• The .NET Framework SDK. Search by using the phrase **Deploying Applications**. |
| **6.** Add the InternalBusinessApp.exe file to the Application folder. Browse to *install_folder*\Labfiles\Lab09_1\Ex03\Starter\Business Application Shell\bin\Debug to find the InternalBusinessApp.exe file. | **a.** For more information about deploying applications, see the following resources:<br>• Lesson: Deploying Windows Forms Applications in Module 9, “Deploying Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*<br>• The .NET Framework SDK. Search by using the phrase **Deploying Applications**. |

| Tasks | Additional information |
|---|---|
| **7.** Add the ExpenseReport.dll file to the global assembly cache folder. Browse to *install_folder*\Labfiles\Lab09_1\Ex03\Starter\Business Application Shell to find the ExpenseReport.dll. | **a.** For more information about how to add the global assembly cache folder, see the following resource:<br>- Practice: Creating and Using a Windows Installer Deployment Project in Module 9, “Deploying Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>**b.** For more information about deploying applications, see the following resources:<br>- Lesson: Deploying Windows Forms Applications in Module 9, “Deploying Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>- The .NET Framework SDK. Search by using the phrase **Deploying Applications**. |
| **8.** Add the localized resource assemblies to their associated folders in the setup project. These folders are the localized folders that you created in step 5. Browse to *install_folder*\Labfiles\Lab09\Ex03\Starter\Business Application Shell\bin\Debug to find the folders that contain the localized resource files.<br><br>The localized resource files are named InternalBusinessApp.resources.dll and can be found in the folders named de, de-DE, en, en-US, fr, fr-FR, ja, and ja-JP under *install_folder*\Labfiles\Lab09_1\Ex03\Starter\Business Application Shell\bin\Debug. | **a.** For more information about packaging and deploying resources, see the following resource:<br>- The .NET Framework SDK. Search by using the phrase **Deploying Applications**. |
| **9.** Create the directory for the Help file, and add it to the project.<br>**a.** Name the custom folder IISRoot.<br>**b.** Set the DefaultLocation property of IISRoot to C:\inetpub\wwwroot.<br>**c.** Browse to install_folder\ Labfiles\Lab09_1\Ex03\Starter\Business Application Shell, find the InternetBusinessAppHelp.htm file, and add it to IISRoot. | **a.** For more information about deploying applications, see the following resources:<br>- Lesson: Deploying Windows Forms Applications in Module 9, “Deploying Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>- The .NET Framework SDK. Search by using the phrase **Deploying Applications**. |

| Tasks | Additional information |
|---|---|
| **10.** Set the following project properties.<br>**Property** — **Value**<br>Author — Contoso, Ltd.<br>Manufacturer — Contoso Ltd<br>ProductName — Internal Business Application<br>Title — Internal Business Application<br>Version — 3.0.1<br>**a.** When prompted to change the ProductCode and PackageCode properties, click **Yes**.<br>**b.** Set the AlwaysCreate property for the Application folder to True. | No additional information is necessary for this task. |
| **11.** Create the application shortcuts. | **a.** For more information about deploying applications, see the following resources:<br>• Lesson: Deploying Windows Forms Applications in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK. Search by using the phrase **Deploying Applications**. |
| **Warning:** A new dependency to ExpenseReport.dll may appear under **Detected Dependencies** in Solution Explorer. If this occurs, right-click the **ExpenseReport.dll** dependency, and select **Exclude** from the context menu. If you do not do this, a copy of the ExpenseReport.dll file will be installed in the target application folder in addition to the global assembly cache, and your application will use the .dll file from the application folder instead of the .dll from the global assembly cache. | |
| **12.** Build the project, and install the Internal Business Application. Browse to *install_folder*\ \Labfiles\Lab09_1\Ex03\Starter\ InternalBusinessApplication\Debug to find the Setup.exe file. | **a.** For more information about deploying applications, see the following resources:<br>• Lesson: Deploying Windows Forms Applications in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK. Search by using the phrase **Deploying Applications**. |
| **13.** Test the Internal Business Application.<br>• Test the different region/culture settings that the application supports. | **a.** For more information about deploying applications, see the following resources:<br>• Lesson: Deploying Windows Forms Applications in Module 9, "Deploying Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK. Search by using the phrase **Deploying Applications**. |

# Exercise 4
# Using an Application Configuration File

In this exercise, you will create an application configuration file that will redirect the Internal Business Application to load a new version of the ExpenseReport.dll file.

There are starter and solution files associated with this exercise. Browse to *install_folder*\Labfiles\Lab09_1\Ex04\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab09_1\Ex04\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

| Tasks | Additional information |
|---|---|
| **1.** Open the Visual Studio .NET Command Prompt window and install the ExpenseReport.dll file into the global assembly cache. After the file is installed in the global assembly cache, delete it from the folder. Browse to *install_folder*\ Labfiles\Lab09_1\Ex04\Starter to find the ExpenseReport.dll file. | **a.** For more information about how to create application configuration files, see the following resource:<br>• Practice: Creating and Using Application Configuration Files in Module 9, “Deploying Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>**b.** For more information about configuration files, see the following resources:<br>• Lesson: Deploying Windows Forms Applications in Module 9, “Deploying Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK. Search by using the phrase **Configuration Files**. |
| **2.** Run the .NET Configuration Tool, and create a new application configuration file for the InternalBusinessApp.exe application. Change the **BindingRedirect** entry from **3.0.1.1** to **4.0.1.1**. Browse to *install_folder*\Labfiles\Lab09_1\Ex04\ Starter to find the InternalBusinessApp.exe file. | **a.** For more information about how to create application configuration files, see the following resource:<br>• Practice: Creating and Using Application Configuration Files in Module 9, “Deploying Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>**b.** For more information about configuration files, see the following resources:<br>• Lesson: Deploying Windows Forms Applications in Module 9, “Deploying Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*.<br>• The .NET Framework SDK. Search by using the phrase **Configuration Files**. |

| Tasks | Additional information |
|---|---|
| **3.** Run the application.<br>• Verify that the application still runs with the new version of ExpenseReport.dll. | No additional information is necessary for this task. |

# Module 10: Securing Windows Forms Applications

**Contents**

# Instructor Notes

**Presentation:**
**75 minutes**

**Lab:**
**30 minutes**

This module starts with an overview of the Microsoft® .NET Framework security model. In this module, students learn how to use code access security and role-based security in their applications.

After completing this module, students will be able to:

- Describe the .NET Framework security model.
- Use code access security to secure an application.
- Use role-based security to control access to an application.

**Required materials**

To teach this module, you need the Microsoft PowerPoint® file 2555A_10.ppt.

**Preparation tasks**

To prepare for this module:

- Read all of the materials for this module.
- Review the animation for this module.
- Complete the demonstrations, practice, and lab.

# How to Teach This Module

This section contains information that will help you to teach this module.

- If students are interested in referencing code examples in other languages, point them to "Language Equivalents" in the Microsoft Visual Studio® .NET Help documentation. This section provides examples in languages such as Microsoft Visual Basic® .NET, C#, and Java.
- The lab at the end of this module is based on the Expense Report application in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*, and is intended to simulate a "real world" environment in which students will demonstrate what they learned during the lecture and practice portions of the module. The lab does not provide step-by-step detailed instructions; instead, the students are given tasks to complete in the left column, and a list of resources that they can use (if they need help) in the right column. Students get hands-on experience that they need by completing the practice activity in the module.

# Lesson: Security in the .NET Framework

This section describes the instructional methods for teaching this lesson.

This is an overview lesson. The main point of this lesson is to introduce the key concepts that will help students understand how the .NET Framework security model works. The other two lessons in the module cover the details of how to implement code access and role-based security in applications. Do not go into implementation details in this overview lesson.

**How Does Code Access Security Work?**

This topic defines code access security and presents an overview of how the .NET Framework uses security policy to map evidence about an assembly to a set of permissions for that assembly. Point out the note regarding .NET Framework Service Pack 1.

**Animation: Code Access Security**

In this animation, students see an example of how the permission grant for a Microsoft .NET Framework assembly is determined based on evidence and policy levels. This is a good time to check to make sure that students understand the key elements of code access security.

**What is Role-Based Security?**

This topic defines role-based security and the concepts of identity and principal.

**What is Authentication?**

This topic defines authentication and provides some examples of authentication authorities.

**What is Authorization?**

This topic defines authorization. Emphasize to students that there is an order relationship here: authentication is first, then authorization.

# Lesson: Using Code Access Security

This section describes the instructional methods for teaching this lesson.

### How to Use Code Access Security

This topic covers permission requests and why developers should use them in their applications. Stress to students that the .NET Framework applies the security policy to all applications whether they make permission requests in their applications or not. The motivation for adding permission requests to applications is to document what permissions the application needs and to test them in the different deployment situations. This helps a developer to discover where to add additional exception handling related to permission request failures while the application while is still in the development phase instead of after it has been deployed.

### Demonstration: Administering Security Policy Settings

This demonstration shows students how to use the .NET Framework configuration Microsoft Management Console (MMC) snap-in to view and modify security policy on their computers. This is useful so students can set up test environments for applications they are developing.

### How to Test the Code Access Security of an Application

This topic covers how to use Code Access Security tool (Caspol.exe) which is a command-line based utility for viewing and manipulating security policy. Caspol.exe is an alternative utility that students can use to set up test environments for applications they are developing.

# Lesson: Using Role-Based Security

This section describes the instructional methods for teaching this lesson.

To provide a transition from the previous lesson and this one, you can remind students that code access security controls what resources certain sections of code in their application can access and that role-based security allows developers to limit which users can run certain parts of an application.

The main thing to emphasize in this lesson is the flexibility that role-based security provides and the benefit that much of the infrastructure required for security checking is provided by the .NET Framework, which simplifies the developer's work.

### How Role-Based Security Works

This topic provides more detail about the types of principals and identities that can be used to implement role-based security.

### How to Use Principals and Identities to Control Access to an Application

In this topic, students learn how to use the **IsInRole** method of the **Principal** object to control access to an application.

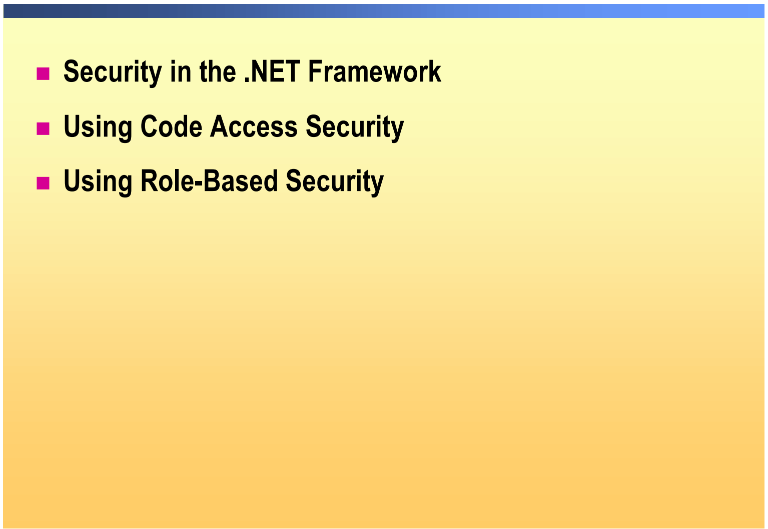
### Demonstration: Using Role-based Security to Control Access to an Application

In this demonstration, students follow the flow of an application in the debugger to see how an application implements role-based security.

# Lab: Adding and Testing Permission Requests

- Ensure that you have demonstrated the lab application Expense Report Application in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*, before students attempt the lab. To see how to demonstrate the lab scenario, see Module 0 in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).*
- The practice exercise will enable students to successfully complete the lab exercise. Therefore, ensure that students have completed the practice exercise.

# Overview



- Security in the .NET Framework
- Using Code Access Security
- Using Role-Based Security

**Introduction**

The Microsoft® .NET Framework provides many new features to secure an application.

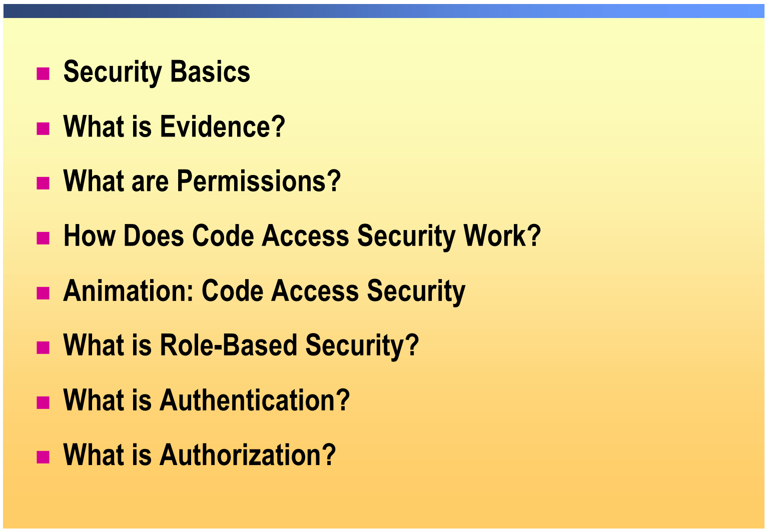
In this module, you will learn about the .NET Framework security model and learn how to use .NET Framework security features in your applications.

**Objectives**

After completing this module, you will be able to:

- Describe the .NET Framework security model.
- Use code access security to secure an application.
- Use role-based security to control access to an application.

# Lesson: Security in the .NET Framework



- Security Basics
- What is Evidence?
- What are Permissions?
- How Does Code Access Security Work?
- Animation: Code Access Security
- What is Role-Based Security?
- What is Authentication?
- What is Authorization?

**Introduction**

The Microsoft Windows® 2000 security model protects a system by preventing unauthorized users from accessing that system. The security model in the .NET Framework protects application code and data from being misused or damaged by other code by enforcing security restrictions on managed code. The .NET Framework security model is based on type safety, code signing, encryption of data, code access security, role-based security, and isolated storage.

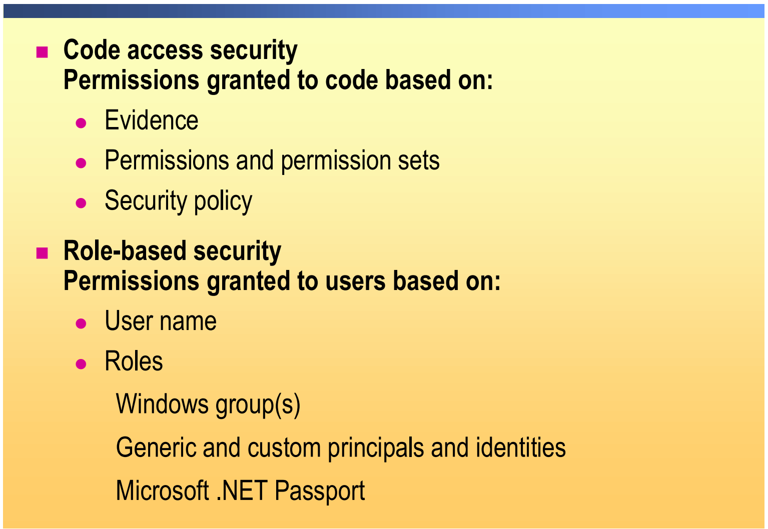
In this lesson, you will learn about the .NET Framework security model.

**Lesson objectives**

After completing this lesson, you will be able to:

- Define evidence and describe its role in the security system in the .NET Framework.
- Define Authentication and Authorization and describe their roles in the security system in the .NET Framework.
- List the major characteristics of code access security and role-based security.
- Describe the .NET Framework security model.

# Security Basics



- **Code access security**
  **Permissions granted to code based on:**
  - Evidence
  - Permissions and permission sets
  - Security policy
- **Role-based security**
  **Permissions granted to users based on:**
  - User name
  - Roles

    Windows group(s)

    Generic and custom principals and identities

    Microsoft .NET Passport

### Introduction

Two important aspects of the .NET Framework security model are code access security, which controls what resources your code can access, and role-based security, which allows developers to limit which users can run certain parts of an application.

### Code access security

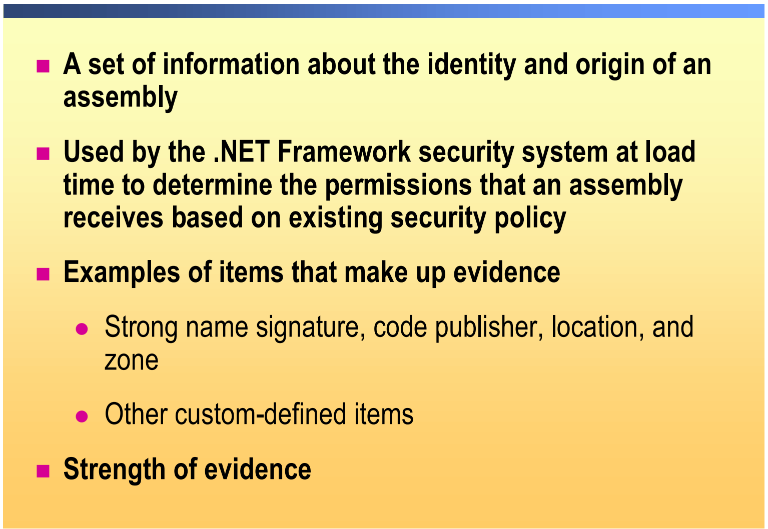
The .NET Framework code access security model allows code to use protected resources only if it has permission to do so. This is implemented by using the concept of permissions, which represent the right for code to access protected resources. Evidence about an assembly is used to grant code a set of permissions based on security policy. When code accesses resources to perform a task, a demand for the permissions it needs is made, and the .NET Framework security system determines whether all of the callers to that code have been granted the permissions being demanded.

### Role-based security

The .NET Framework includes support for role-based security. This allows code to determine the identity and role membership for the user of an application and make access decisions based on identity.

For more information about building secure applications, see *Writing Secure Code*, by Howard, Michael, and David LeBlanc. Redmond, WA: Microsoft Press, 2002.

# What is Evidence?



- A set of information about the identity and origin of an assembly
- Used by the .NET Framework security system at load time to determine the permissions that an assembly receives based on existing security policy
- Examples of items that make up evidence
  - Strong name signature, code publisher, location, and zone
  - Other custom-defined items
- Strength of evidence

**Introduction**

The security system uses this evidence about the assembly to determine which permissions to grant it based upon existing security policy.

**Definition**

*Evidence* is a set of information about the identity and origin of an assembly.

**Examples**

Evidence may include:

- The assembly's strong name, consisting of a unique public key, a simple name, and a version.
- The assembly's publisher, from the Microsoft Authenticode® signature.
- The zone from which the assembly originates, such as the local computer, intranet, or Internet zones.
- The location from which the assembly originates, expressed as a URL, universal naming convention (UNC) path, or local computer folder.
- The cryptographic hash of the assembly.

The creator of an assembly can also include custom evidence with the assembly. This custom evidence is evaluated only if security policy is configured to use the custom evidence.

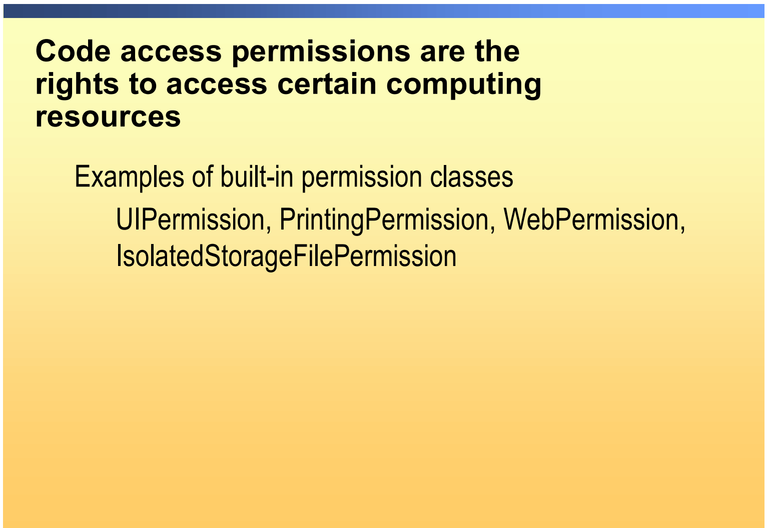
**Strength of evidence**

Some forms of evidence are stronger than others and can therefore be used to make more far-reaching security policy decisions.

Strong names, for instance, provide an extremely reliable form of evidence, because they are very difficult to falsify unless the publisher's private key has been compromised. Authenticode signatures are also strong forms of evidence.

Conversely, evidence such as an assembly's zone or URL is weaker. Web sites can be hacked, and packets can be tampered with over the Internet. It is a risk to grant permissions based heavily on weaker forms of evidence. However, both strong and weak forms of evidence can be combined in an effective security policy.

# What are Permissions?



**Code access permissions are the rights to access certain computing resources**

- Examples of built-in permission classes
  - UIPermission, PrintingPermission, WebPermission, IsolatedStorageFilePermission

****************************ILLEGAL FOR NON-TRAINER USE******************************

**Introduction**

Code access permissions represent rights to access certain computing resources.

**Example**

Actions protected by permissions include reading and writing files on the file system, accessing environment variables, and making calls to ADO.NET for database access.

**Built-in code access permission classes**

The .NET Framework has many built-in code access permission classes that are designed to protect access to system resources. The .NET Framework uses the built-in permission classes to protect system resources. You, as a developer, do not need to take any further action to use system resources, because the .NET Framework will make the appropriate security checks. The built-in permission classes are listed in the following table.

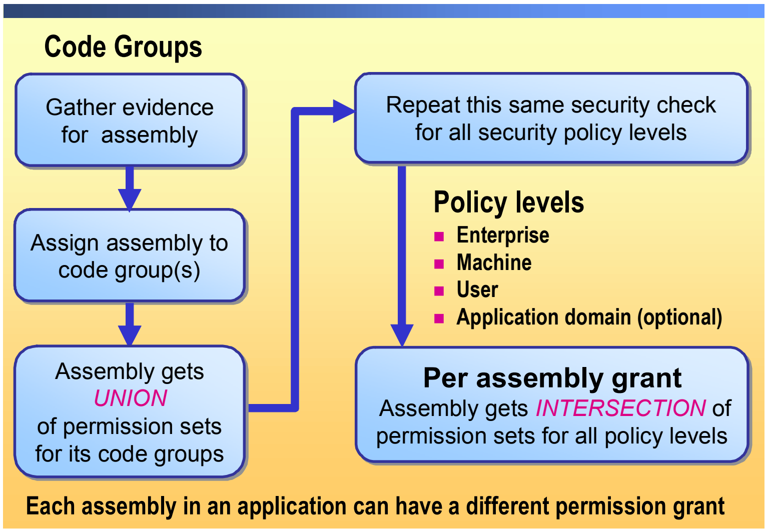
| Code access permission class | Resource protected |
|---|---|
| **DirectoryServicesPermission** | Directory services |
| **DnsPermission** | DNS services |
| **EnvironmentPermission** | Environment variables |
| **EventLogPermission** | Event logs |
| **FileDialogPermission** | File dialog boxes in the UI |
| **FileIOPermission** | Files and folders on the file system |
| **IsolatedStorgeFilePermission** | Isolated storage |
| **MessageQueuePermission** | Message queues |
| **OleDbPermission** | Databases accessed by the OLEDB data access provider |
| **PerformanceCounterPermission** | Performance counters |
| **PrintingPermission** | Printers |
| **ReflectionPermission** | Type information at run time |
| **RegistryPermission** | Registry |

**(*continued*)**

| Code access permission class | Resource protected |
|---|---|
| **SecurityPermission** | Execute code, assert permissions, call unmanaged code, skip verification, and other rights |
| **ServiceControllerPermission** | Running or stopping services |
| **SocketPermission** | Connections to other computers by means of sockets |
| **SqlClientPermission** | Databases accessed by the Microsoft SQL Server™ data access provider |
| **UIPermission** | Windows and other UI elements |
| **WebPermission** | Connections to other computers by means of HTTP |

In addition to the built-in permission classes, developers can add new permissions to the security system by implementing custom permissions.

# How Does Code Access Security Work?



Code Groups

Gather evidence for assembly

Assign assembly to code group(s)

Assembly gets *UNION* of permission sets for its code groups

Repeat this same security check for all security policy levels

Policy levels

- Enterprise
- Machine
- User
- Application domain (optional)

**Per assembly grant**

Assembly gets *INTERSECTION* of permission sets for all policy levels

Each assembly in an application can have a different permission grant

**Introduction**

The .NET Framework enforces security restrictions on managed code. Every assembly that is loaded is granted a set of permissions by the .NET Framework to access various system resources. A group of permissions is called a *permission set*. These permissions are based on security policy. The security policy uses evidence about an assembly to determine which permissions to grant that assembly. In other words, the .NET Framework uses security policy to map evidence about an assembly to a set of permissions for that assembly.

**Code groups**

A code group consists of a membership condition and a set of permissions that an assembly might be granted if it meets that membership condition. The runtime evaluates the membership condition specified in the code group against the evidence about the assembly. If the assembly meets the membership condition, it is eligible to be granted the permission set associated with the code group. When an assembly meets the membership conditions for a code group, it is said to be a member of that code group.

As an example, the membership condition for a code group could specify that the publisher of an assembly is Microsoft. Therefore, any assembly published by Microsoft would satisfy the membership condition and be eligible to receive the permission set associated with the code group. This permission set could represent the right to access the C:\temp directory and the *USERNAME* environment variable, for instance.

An assembly receives the union of permissions for all code groups to which it belongs.

### Security policy levels

Security policy is organized into different policy levels: enterprise policy level, machine policy level, user policy level, and an optional application domain policy level.

| Security Policy Levels | Description |
|---|---|
| Enterprise | Enterprise-level policy is specified by the network administrator, and contains a code group hierarchy that applies to all managed code on the entire network. |
| Machine | Machine-level policy is specified by the local computer administrator, and contains the code group hierarchy that applies to all managed code on the computer. |
| User | User-level policy contains a code group hierarchy that applies to all managed code run by a particular user. Either the local computer administrator or the user sets user policies |
| Application domain (optional) | Application domain policy level is an optional level that provides isolation, unloading, and security boundaries for executing managed code. |

A final permission grant is assigned on a per assembly basis, so each assembly in an application can have a different permission grant.

### Default security policy

The common language runtime provides a default security policy and uses the following named permission sets to implement that policy:

- Nothing

  Provides code with no permissions. Code cannot run. Under default security policy, code in the untrusted zone gets this permission set.

- Execution

  Provides permission for code to run only. Does not allow code to use any protected resources.

- Internet

  Allows code to execute, to create safe top-level windows and file dialog boxes, to make Web connections to the same site that the assembly originates from, and to use isolated storage with a quota. Under default security policy, all code from the Internet and trusted zones receives this permission set.

  **Note** Service Pack 1 of the .NET Framework changes default security policy for code executed from the Internet zone. Under the new policy, code from the Internet receives no permissions by default.

- LocalIntranet

  Allows code to execute, to create user interface elements without restrictions; to use isolated storage with no quota; to use DNS services; to read the USERNAME, TEMP, and TMP environment variables; to make Web connections to the same site the assembly originates from; and to read files in the same folder as the assembly. Under default security policy, all code from the LocalIntranet zone receives this permission set.

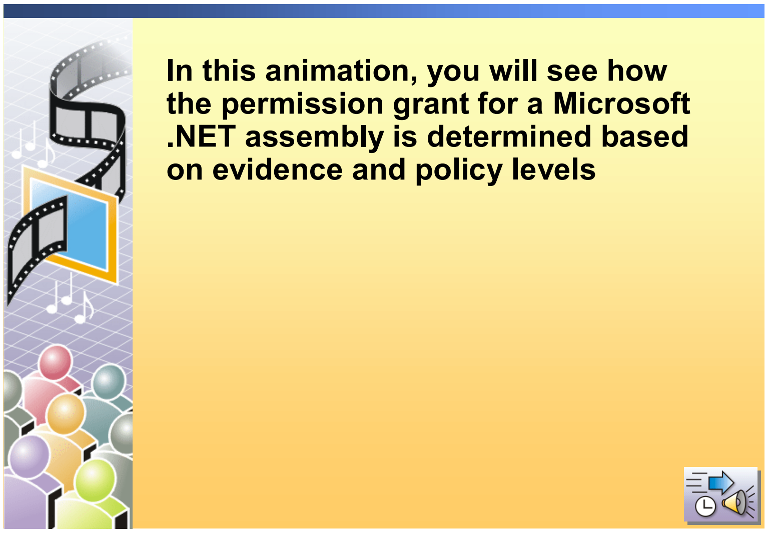
- Everything

  Provides all standard permissions except permission to skip verification.

- FullTrust

  Provides full access to all resources protected by permissions. Under default security policy, all code from the local computer zone receives this permission set.

# Animation: Code Access Security



In this animation, you will see how the permission grant for a Microsoft .NET assembly is determined based on evidence and policy levels

### Introduction

In this animation, you will see how the permission grant for a Microsoft .NET assembly is determined based on evidence and policy levels.

### Summary

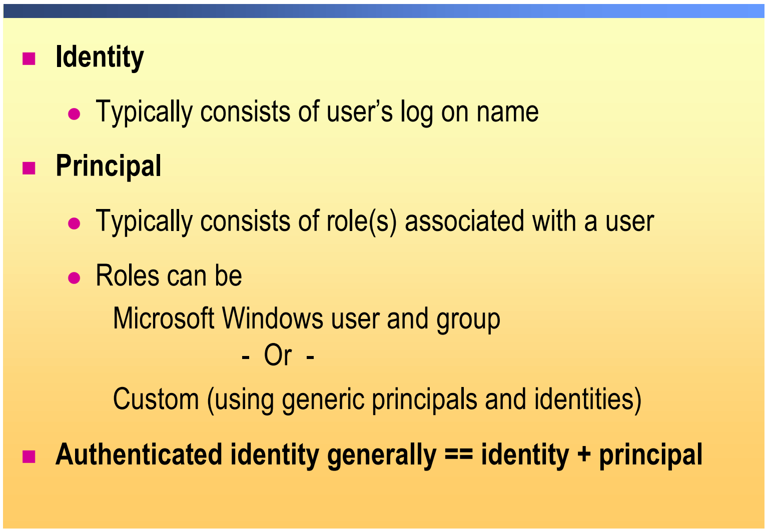
When an assembly is loaded, the common language runtime examines the evidence that the code provides. The permissions that are granted to managed code are determined at the time that the assembly containing that code is loaded.

Policies are represented by a hierarchy of code groups. At the root of the hierarchy of each code group is a group containing all code. This root group has child groups, which in turn may have child groups, and so on. Code that is not a member of a parent code group cannot be a member of any of a child code group in the code group hierarchy. However, code that is a member of a parent code group can also be a member of one or more child code groups.

The runtime uses the evidence about the code to decide which code groups that code belongs to. Code groups specify a set of membership conditions for code. The permission set that the code is assigned at the machine policy level is the union of the permissions that are associated with all of the code groups that the code is a member of.

This procedure is carried out for each policy level and is based on the code's evidence. The runtime computes the intersection of the permission sets that have been assigned to the code at each policy level to determine the final set of permissions that are assigned to the code. For example, if the user policy grants only the permissions that are part of the LocalIntranet permission set, and the machine policy grants the FullTrust permission set, then the final permission set granted is that of LocalIntranet intersected with FullTrust, which is the LocalIntranet permission set.

# What is Role-Based Security?



- **Identity**
  - Typically consists of user’s log on name
- **Principal**
  - Typically consists of role(s) associated with a user
  - Roles can be

    Microsoft Windows user and group

    - Or -

    Custom (using generic principals and identities)
- **Authenticated identity generally == identity + principal**

**Introduction**

A *role-based* security model uses authenticated identity information about the user to make decisions about security authorization. The authenticated identity information typically consists of the user’s login name and the roles associated with the user.

Role-based security in the .NET Framework extensively uses two concepts: identities and principals.

**Identity**

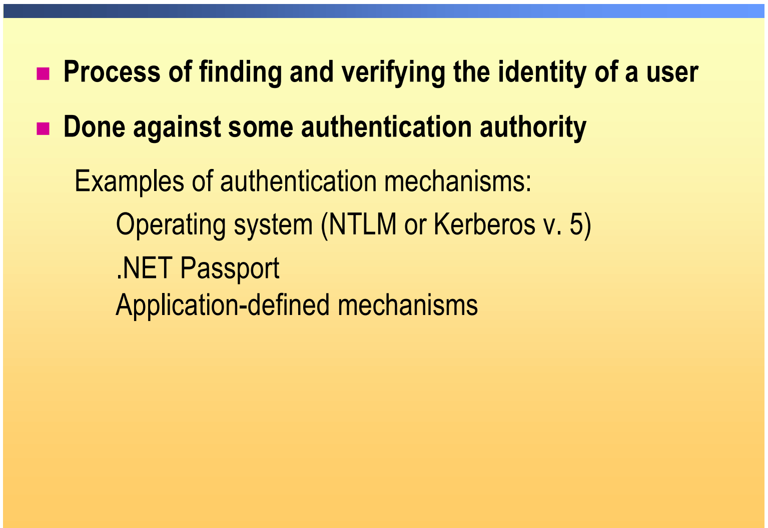
An *identity* encapsulates the user’s login name.

**Principal**

A *principal* encapsulates the user’s role membership information.

Role-based security in the .NET Framework allows developers to use Microsoft Windows user and group information, or to do custom authentication and authorization by using generic principals and identities.

# What is Authentication?



- **Process of finding and verifying the identity of a user**
- **Done against some authentication authority**

  Examples of authentication mechanisms:

  Operating system (NTLM or Kerberos v. 5)<br>
  .NET Passport<br>
  Application-defined mechanisms

**Introduction**

Before you determine what a specific user is authorized to do, you must verify the identity of that user.

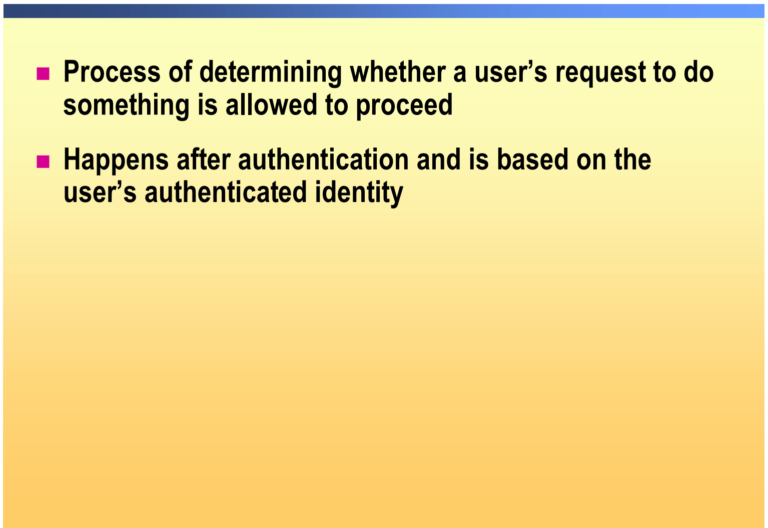
**Definition**

*Authentication* is the process of discovering and verifying the identity of a user by examining the user’s credentials, and then validating those credentials against some authority.

**Authentication mechanisms**

A variety of authentication mechanisms are used today, some of which can be used with .NET Framework role-based security. Examples of commonly used mechanisms include the operating system, Microsoft Passport, and application-defined mechanisms. Specific examples of operating system authentication mechanisms are NTLM authentication and the Kerberos version 5 authentication protocol.

## What is Authorization?



- Process of determining whether a user's request to do something is allowed to proceed
- Happens after authentication and is based on the user's authenticated identity

**Introduction**

The next step after authentication is authorization.

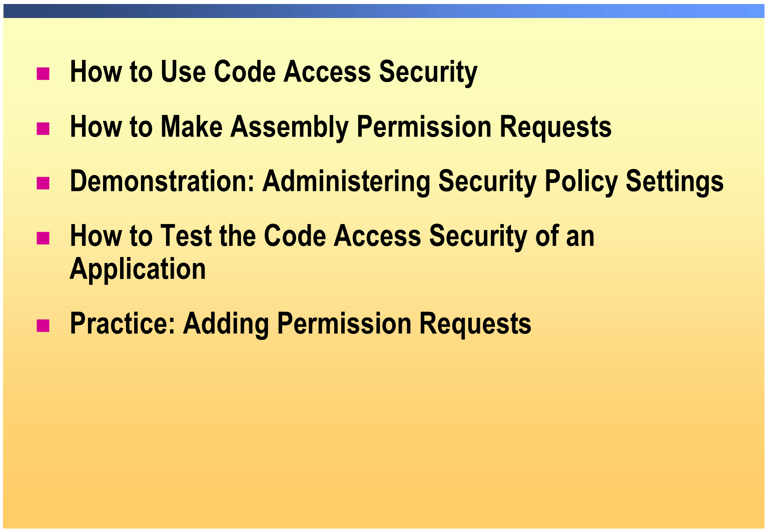
**Definition**

*Authorization* is the process of determining whether a user is allowed to perform a requested action.

Authorization occurs after authentication and uses information about a user's identity and roles to determine what resources that user can access. You can use .NET Framework role-based security to implement authorization.

For more information about how to implement authorization, see Using Role-Based Security in this module.

# Lesson: Using Code Access Security



- How to Use Code Access Security
- How to Make Assembly Permission Requests
- Demonstration: Administering Security Policy Settings
- How to Test the Code Access Security of an Application
- Practice: Adding Permission Requests

**Introduction**

Every application that targets the common language runtime must interact with the runtime's security system. When an application executes, the runtime automatically evaluates it and gives it a set of permissions. Depending on the permissions that the application receives and the resources it uses, it either runs properly or generates a security exception.

The local security settings on a particular computer ultimately determine which permissions code receives. Because these settings can change from computer to computer, you can never be sure that your code will receive sufficient permissions to run. Therefore, every developer must be familiar with code access security concepts to write effective applications targeting the common language runtime.

Interaction with the runtime security system is performed by using imperative and declarative security calls. *Declarative* calls are performed by using attributes; *imperative* calls are performed by using new instances of classes in your code. Some calls can only be performed imperatively, while others can be performed only declaratively. Some calls can be performed in either manner. For more information about imperative security calls and class and method level declarative security calls, see Course 2350A, *Securing and Deploying Microsoft .NET Applications*.

Because most system resources are already protected by built-in permissions, most developers will not need to implement new permissions. Also, the runtime code that accesses these resources already makes the appropriate security checks to make sure calling code has been granted the proper permissions. So, the main task for developers is to document their use of permissions and make sure their code behaves as expected in the security contexts in which it will run. This is done by using permission requests and runtime tools to configure security policy.

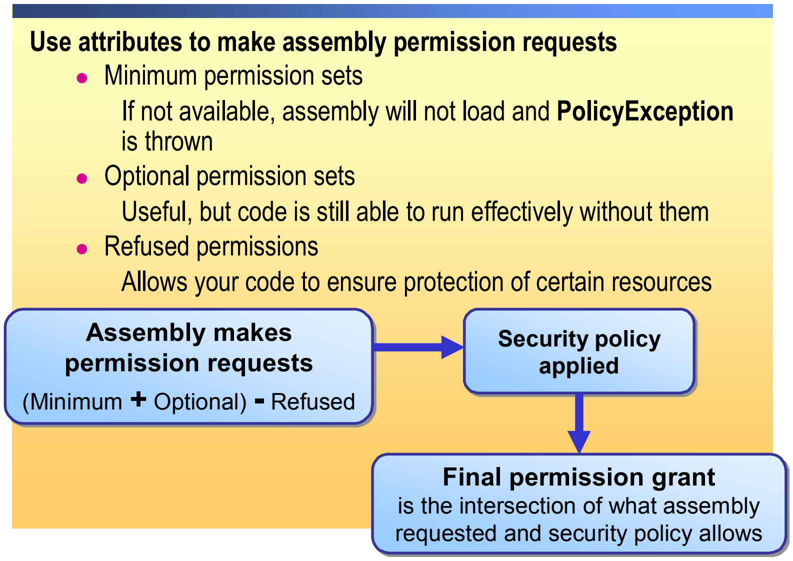
In this lesson, you will learn how to implement permission requests in Windows Forms applications.

**Lesson objectives**

After completing this lesson, you will be able to:

- Make assembly level declarative requests for permissions to access resources.
- Test applications in various security contexts.

## How to Use Code Access Security



**Use attributes to make assembly permission requests**

- Minimum permission sets

  If not available, assembly will not load and **PolicyException** is thrown

- Optional permission sets

  Useful, but code is still able to run effectively without them

- Refused permissions

  Allows your code to ensure protection of certain resources

**Assembly makes permission requests**
(Minimum + Optional) - Refused

**Security policy applied**

**Final permission grant**
is the intersection of what assembly requested and security policy allows

### Introduction

Because the NET Framework protects system resources, the major security task for developers is to make permission requests to document the permissions the code uses. In the development phase, adding permission requests allows you to test if your assembly will be able to run successfully in the security contexts in which you expect the application to be deployed.

To make permission requests for an assembly, add attributes to the AssemblyInfo source file of your project. These attributes are then stored in the metadata of an assembly. The assembly permission requests are examined when an assembly is loaded. The .NET Framework proceeds based on the kind of permission request the assembly makes.

### Kinds of permission requests

The three kinds of permission requests are:

- Minimum permissions (**RequestMinimum**)

  The permissions in these requests represent the minimum set of permissions that an assembly needs to work effectively. If these permissions are not available to an assembly when it is loaded, the .NET Framework does not execute the code in that assembly and throws a **PolicyException**. A minimum permission request for an assembly documents the required permissions for that assembly. Not making a minimum permission request for an assembly is the equivalent of making a minimum permission request of Nothing.

- Optional permissions (**RequestOptional**)

  The permissions in these requests represent permissions that code can use, but the code is still able to run effectively without them. These permissions are granted to an assembly if they are available to that assembly, but if they are not available, the assembly is still allowed to run. Not making an optional permission request for an assembly is the equivalent of making an optional permission request of FullTrust. If an assembly does not need any optional permissions, it is a best practice to make an optional request for no permissions. This prevents your code from getting any additional permissions from security policy, and forces it to run with least privilege.

- Refused permissions (**RequestRefused**)

  The permissions in these requests represent permissions that code is never to be granted, even if security policy allows them to be granted. Not making a refused permission request is the equivalent of making a refused permission request of Nothing.

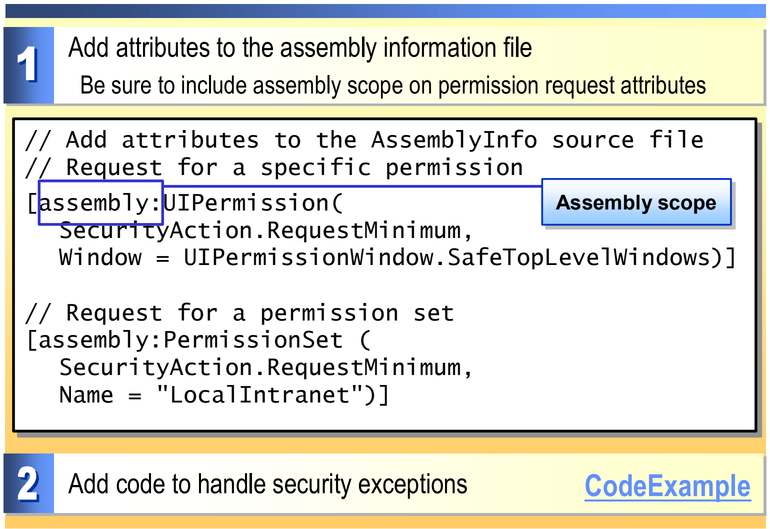
### Final permission grant

The permissions the assembly actually receives are the result of the following operation:

$FG = SP \cap ((M \cup O) - R)$

Where FG is the final grant, SP is the permission set an assembly receives from security policy, $\cap$ is intersection of sets, M is the minimum permission request, $\cup$ is the union of sets, O is the optional permission request, and R is the refused permission request.

# How to Make Assembly Permission Requests



1 Add attributes to the assembly information file

Be sure to include assembly scope on permission request attributes

```
// Add attributes to the AssemblyInfo source file
// Request for a specific permission
[assembly:UIPermission(
   SecurityAction.RequestMinimum,
   Window = UIPermissionWindow.SafeTopLevelWindows)]

// Request for a permission set
[assembly:PermissionSet (
   SecurityAction.RequestMinimum,
   Name = "LocalIntranet")]
```

2 Add code to handle security exceptions

### Introduction

You can request permissions by adding assembly-level attributes to the AssemblyInfo source file. You can make more than one permission request of a certain kind by using multiple attributes. These attributes vary, depending on the permissions you are requesting. The attribute used to make permission requests is scoped with *assembly* and the **SecurityAction** is **RequestMinimum**, **RequestOptional**, or **RequestRefused**.

### Procedure: Making assembly permission requests

To better document your code, place your assembly permission requests in the AssemblyInfo source file.

1. Open the AssemblyInfo.cs source file for the appropriate project and add attributes for permission requests. Be sure to include the assembly scope qualifier to the request, as in the following examples:

### Example: Assembly permission request attributes

```
// Add attributes to the AssemblyInfo source file
// Request for a specific permission
[assembly:UIPermission(
  SecurityAction.RequestMinimum,
  Window = UIPermissionWindow.SafeTopLevelWindows)]

// Request for a permission set
[assembly:PermissionSet(
  SecurityAction.RequestMinimum,
  Name = “LocalIntranet”)]
```

The following example makes an optional permission request for no permissions:

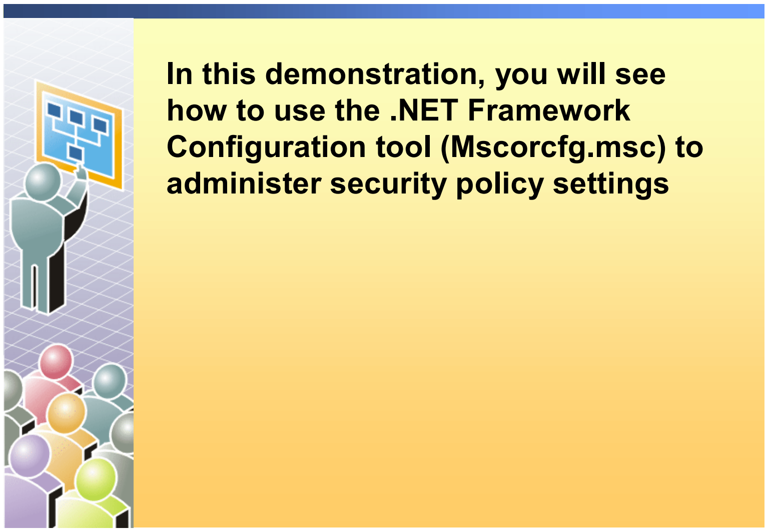
```
[assembly:PermissionSet(
  SecurityAction.RequestOptional,
  Unrestricted = false)]
```

2. If you make any optional permission requests, your code may be granted the permission. However, if it is not, you’ll have to add code to handle security exceptions.

### Example: Handling security exceptions

```
using System.Security;
using System.IO;

FileStream fs = null;

try
{
  // open a file for reading
  fs = new FileStream("C:\\log.txt",
    FileMode.Open,
    FileAccess.Read);
  // read from file
  …
}
catch (SecurityException se)
{
  // display error message
  MessageBox.Show(“ExceptionMessage: “ + se.Message);
}
```

# Demonstration: Administering Security Policy Settings



In this demonstration, you will see how to use the .NET Framework Configuration tool (Mscorcfg.msc) to administer security policy settings

**Introduction**

The .NET Framework configuration Microsoft Management Console (MMC) snap-in provides a graphical interface to modify security policy.

**Instructions**

► **To view current security policy**

1. Run the .NET Framework Configuration tool (Mscorcfg.msc) by double-clicking the Mscorcfg.msc icon on the desktop.
2. In the console tree, expand the following nodes:
   a. **Runtime Security Policy**
   b. **Machine**
   c. **Code Groups**
   d. **All_Code**

   Point out that these code groups are the children of the root (All_Code) code group for the machine policy level.
3. In the console tree, click the **My_Computer_Zone** code group.
   a. In the details pane, point out that the properties for this code group are shown. These include a description of the code group, the membership condition for the code group, and the permission set granted by the code group.
   b. Point out that the last four code groups under All_Code have been added for testing purposes for the practice and lab in this module.
4. In the console tree, expand the **Internet_Zone** node and point out that it has a child code group that grants access to the same Internet site from which the code originated. This code group will only be evaluated if its parent code group's membership condition is satisfied.
5. In the console tree, right-click the **Internet_Zone** code group, and then click **Properties**. Point out that this dialog box shows the properties for the code group, and also lets you edit them.

6. Click the **Membership Condition** tab, and point out that the membership condition for this code group is Internet zone.

    a. Click the **Condition type** box to show the different conditions available for use in a membership condition.

    b. Select the **Strong Name** condition type, and point out that the parameters for the condition type change.

    c. Click the **Condition type** box and change it back to **Zone**.

7. Click the **Permission Set** tab, and point out that the permission set granted for this code group is the Internet permission set.

    a. Click the **Permission set** box to show the different permission sets available for use, but do not change the permission set. This tab also shows the permissions that make up the Internet permission set.

    b. In the **Permission** list, click **User Interface**, and then click the **View Permission** button. Point out that the **Permission Viewer** dialog box displays the user interface-related permissions that are granted as part of the Internet permission set.

    c. Click **Close** to close the **Permission Viewer** dialog box.

    d. Click **Security**, and then click **View Permission**. Point out the security-related permissions that are either granted or denied as part of the Internet permission set.

    e. Click **Close** to close the permission viewer dialog box.

    f. Click **Cancel** to close the **Internet_Zone Properties** dialog box.

8. The default security policy for the user and enterprise policy levels is that all code gets the **FullTrust** permission set.

    a. To show this in the console tree, in the **Runtime Security Policy** node, expand the **Enterprise** and **Code Groups** nodes, and then, in the **Enterprise** hierarchy, click **All_Code**. Point out that it grants FullTrust.

    b. Do the same thing for the **All_Code** node in the **User** hierarchy, pointing out that it looks the same and also grants FullTrust. Explain that code will not get FullTrust based on these policy levels, because the policy grants for the user and enterprise policy levels will be intersected with the policy grant for the machine policy level to determine the final permission grant. Assuming that the machine policy level grants something less than FullTrust when the enterprise and user policy levels are intersected with the machine policy level, an assembly would get the permission set associated with the machine policy level.

    c. Collapse the **Enterprise** and **User** hierarchies before moving to the next step.

9. The snap-in also allows viewing of permission sets.

    a. In the **Machine** node, expand **Permission Sets**. This shows a list of the permission sets for the machine policy level.

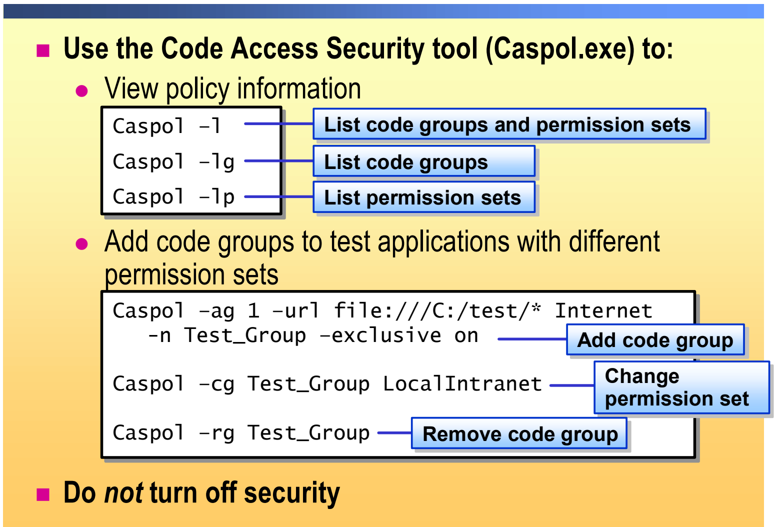
    b. Under the **Permission Sets** list, click **Internet,** and point out that the permissions that make up the Internet permission set are shown in the details pane.

### ► To add a new code group

The snap-in can also be used to add new code groups and permission sets.

1. In the **Machine** and **Code Groups** nodes, right-click the **All_Code** node, and then click **New**. The **Create Code Group** Wizard appears.
2. In the **Create Code Group** Wizard, in the **Name** box, type **New_Test**. In the description box, type **a new test code group** and then click **Next**.
3. In the **Condition Type** box, click **URL**. In the **URL** box, type **file:///C:/testfolder/*** and then click **Next**.
4. In the **Use existing permission set** box, click **LocalIntranet**, and then click **Next**.
5. Click **Finish**.
6. In the console tree, right click the **New_Test** code group, and then click **Properties**. On the **General** tab of the **New_Test Properties** dialog box, select the **This policy level will only have the permissions from the permission set associated with this code group** box to set the **Exclusive** attribute. The code group now has an exclusive URL membership condition of **file:///C:/testfolder/***. Any code run from that folder will get the **LocalIntranet** permission set. Click **OK**.
7. In the console tree, right-click the **New_Test** code group, and then click **Delete**. In the **Remove Code Group** dialog box, click **Yes**. Notice that the code group is gone.
8. Close the MMC snap-in.

# How to Test the Code Access Security of an Application



- Use the Code Access Security tool (Caspol.exe) to:
  - View policy information

```
Caspol -l
Caspol -lg
Caspol -lp
```

List code groups and permission sets

List code groups

List permission sets

  - Add code groups to test applications with different permission sets

```
Caspol -ag 1 -url file:///C:/test/* Internet
   -n Test_Group -exclusive on

Caspol -cg Test_Group LocalIntranet

Caspol -rg Test_Group
```

Add code group

Change permission set

Remove code group

- Do *not* turn off security

## Introduction

The Code Access Security tool (Caspol.exe) is a command-line utility that allows you to view and manipulate security policy. The Code Access Security tool ships with the .NET Framework software development kit (SDK), and it can be used from scripts to configure security.

The Code Access Security tool displays all the available information for the default policy level, which is the machine level. This information includes the code group hierarchy, named permission sets, and assemblies with full trust. To view information about another policy level, you can use a command-line option before any other command-line options to indicate the policy level you wish to view or modify.

The syntax for Caspol.exe is a follows:

```
caspol <option> <args> ...
```

The following table lists the command-line options used in the examples in this topic.

| Command-line option | Action or qualifier | Comments |
|---|---|---|
| -l | List | Lists all available information for the default policy level, which is the machine level. |
| -lg | List code groups | Lists the code groups for the default policy level. This is the most useful option for viewing current security policy. |
| -lp | List permission sets | Lists named permission sets. |
| -ag | Add code group | Add a new code group to the default policy level. |
| -url | URL | The code on the slide uses the URL of the assembly as the condition for the code group. |
| -n | Code group name | The name for the new code group. |
| -exclusive | Set the policy statement Exclusive flag | This is useful for testing purposes. Any assemblies that meet the code group's condition will get the associated permission set, and only the associated permission set, for this policy level. For example, this can be used to prevent code running on the local machine from automatically being granted the FullTrust permission set. |
| -cg | Change code group | Change an existing code group. |
| -rg | Remove code group | Remove an existing code group. |

To get information about the command-line options available for Caspol.exe, type the following command:

```
Caspol -help
```

**Procedure: Using Caspol.exe to view security policy information**

The Code Access Security tool is quick way to view security policy.

1. To list all available information about code groups and permission sets, open the Microsoft Visual Studio® .NET Command Prompt window and type the following command:

   ```
   Caspol -l
   ```

   To open the Visual Studio .NET Command Prompt window, click **Start**, point to **All Programs**, point to **Microsoft Visual Studio .NET**, point to **Visual Studio .NET Tools**, and then click **Visual Studio .NET Command Prompt**.

2. To list code groups, open the Visual Studio .NET Command Prompt window and type the following command:

   ```
   Caspol -lg
   ```

3. To list permission sets, in the Visual Studio .NET Command Prompt window type the following command:

   ```
   Caspol -lp
   ```
4. To reset the current policy level to default security policy, in the Visual Studio .NET Command Prompt window type the following command:

   ```
   Caspol -reset
   ```

By default, all code running on the local computer receives the FullTrust permission set. Often, when testing the security characteristics of your code, it is convenient to have your code run with different permission sets. An easy way to do this is to create a test folder and configure security policy to treat this folder as a different zone.

### Procedure: Testing your application by using Caspol.exe

You can also use the Code Access Security tool to set up security context test environments for your applications.

1. Create a folder named **C:\testfolder**.
2. To create a new code group to test an application that you want to run with the Internet permission set, open the Visual Studio .NET Command Prompt window and type the following command:

   ```
   Caspol -ag 1 -url file:///C:/testfolder/* Internet
   -n Test_Group -exclusive on
   ```
   This adds a new exclusive code group at the machine policy level with a URL membership condition of file:///C:/test/* that grants the Internet permission set and is named Test_Group. The code group must be exclusive. This ensures that other code groups that would normally match your assembly are ignored, as in the case of code whose zone is MyComputer, which would usually be granted FullTrust. You can then copy your application to the test directory, and when you run it, it will get the Internet permission set.

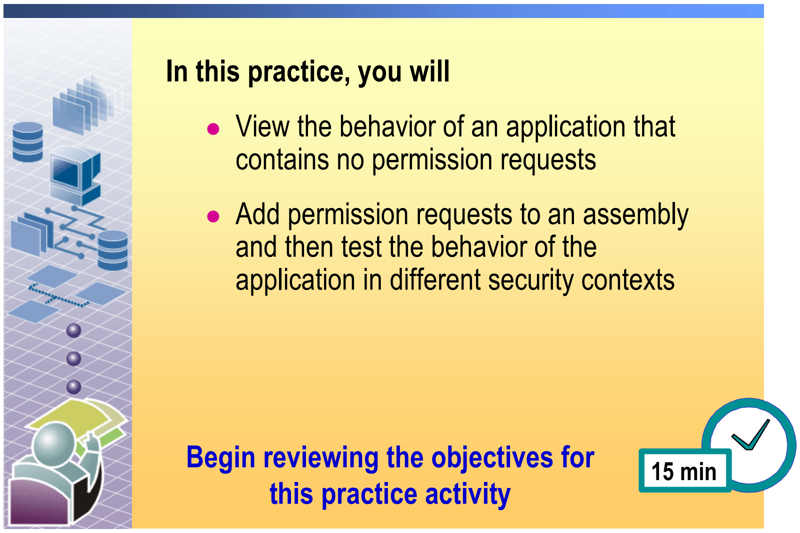
   To open the Visual Studio .NET Command Prompt window, click **Start**, point to **All Programs**, point to **Microsoft Visual Studio .NET**, point to **Visual Studio .NET Tools**, and then click **Visual Studio .NET Command Prompt**.
3. To change this code group to test your application against the LocalIntranet permission set, type the following command:

   ```
   Caspol -cg Test_Group LocalIntranet
   ```
4. To remove your test code group by typing the following command:

   ```
   Caspol -rg Test_Group
   ```

**Important** Do *not* turn off security. The Code Access Security tool has an option that allows you to turn off security. Avoid using this option if at all possible.

# Practice: Adding Permission Requests



**In this practice, you will**

- View the behavior of an application that contains no permission requests
- Add permission requests to an assembly and then test the behavior of the application in different security contexts

**Begin reviewing the objectives for this practice activity**

### Introduction

In this practice, you will compare the behavior of an application that tries to access various resources under different permission request situations and in different security contexts.

### Instructions

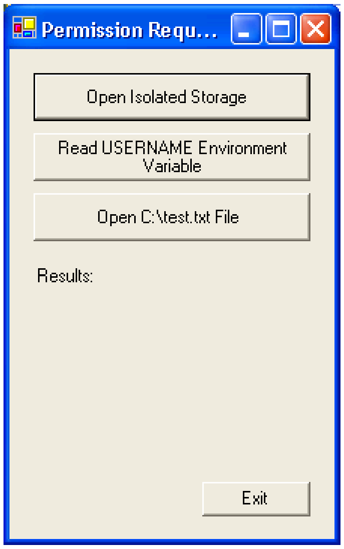
► **View application behavior without permission requests**

1. In Visual Studio .NET, open the PermissionRequests.sln solution file in *install_folder*\Practices\Mod10\Mod10_01\Starter.

**Note** If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

2. Open the PermReqForm.cs form and view its code.
3. Build the project and run the application.

4. Click each of the three buttons at the top of the form and notice that all of the operations succeed.



This happens because the code is running with the **FullTrust** permission set, with no permission request restrictions.

5. Click **Exit** to quit the application.

### ► Add permission requests to the application

1. Open the AssemblyInfo.cs file and find the TODO comment at the bottom of the file. Notice that a minimum permission request for a **UIPermission** has already been added.

2. Add a minimum permission request for the required **IsolatedStorageFilePermission**.

```
[assembly:IsolatedStorageFilePermission(
    SecurityAction.RequestMinimum,
    UsageAllowed =
        IsolatedStorageContainment.DomainIsolationByUser)]
```

3. Add a minimum permission request for the required **EnvironmentVariablePermission**.

```
[assembly:EnvironmentPermission(
    SecurityAction.RequestMinimum,
    Read = "USERNAME")]
```

4. Add an optional permission request for no permissions. This prevents the assembly from getting any additional permissions that security policy may allow.

```
[assembly:PermissionSet(SecurityAction.RequestOptional,
    Unrestricted = false)]
```

5. Build the project and run the application.

6. Click **Open Isolated Storage**, and then click **Read USERNAME Environment Variable**. Notice that these two operations succeed. Then click **Open C:\test.txt File**. That operation fails with a security exception, even though the code would normally get the FullTrust permission set. This happens because the optional permission request for no permissions was added to the assembly.

7. Exit the application.
8. In the **AssemblyInfo** file, add a minimum permission request for the required **FileIOPermission**.

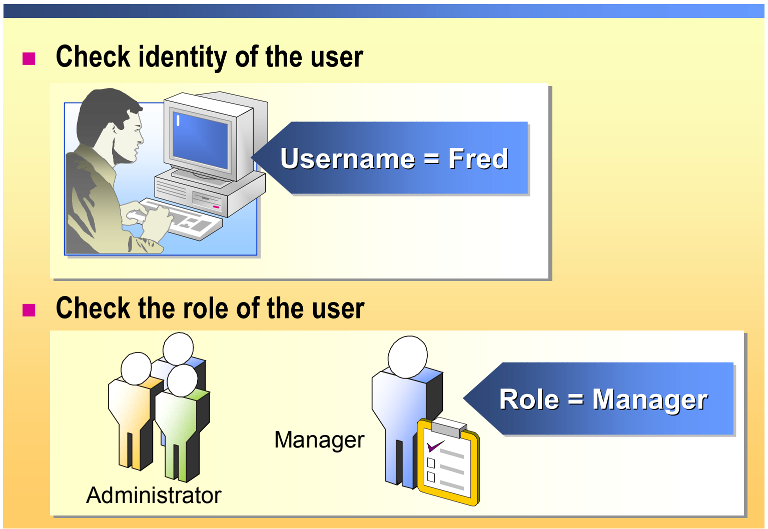
```
[assembly:FileIOPermission(
    SecurityAction.RequestMinimum,
    All = "C:\\test.txt")]
```

9. Build the project and run the application. Try all three operations that you tried in Step 6. Notice that all three operations succeed now. Click **Exit** to quit the application.

► **Test application behavior in other security contexts**

1. In Windows Explorer, copy the PermissionRequests.exe file from *install_folder*\Practices\Mod10\Mod10_01\PermissionRequests\bin\debug to C:\test\LocalIntranet.
2. In the C:\test\LocalIntranet folder, double-click the **PermissionRequests.exe** file to run the application. Notice that an exception window appears, notifying you of a **System.Security.Policy.PolicyException** exception. The application was not able to run because it makes a minimum permission request from permissions that are not granted as part of the LocalIntranet permission set. Specifically, the request for permission to read the C:\Test.txt file is not allowed in the LocalIntranet permission set.
3. Click **No** to avoid debugging the application.

# Lesson: Using Role-Based Security



- Check identity of the user

- Check the role of the user

**Introduction**

Role-based security allows developers to control access to applications based on the user’s identity. In this lesson, you will learn how to use the .NET Framework role-based features in your applications.

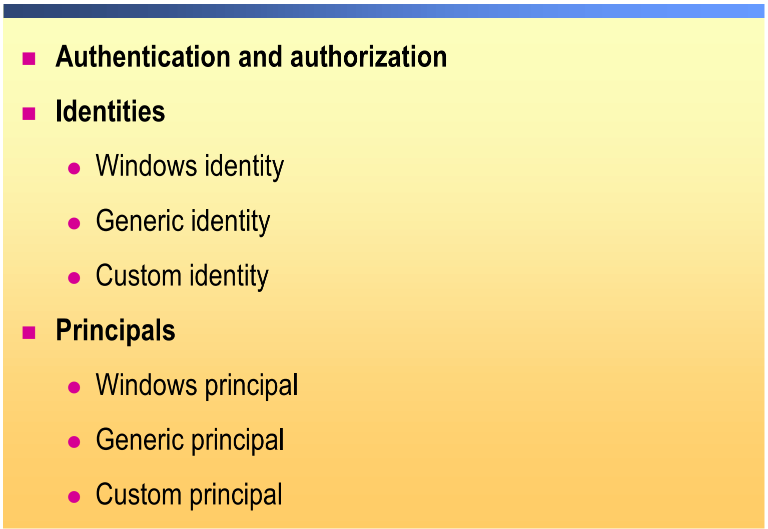
For more information about role-based security calls, see Course 2350B, *Securing and Deploying Microsoft .NET Applications*.

**Lesson objectives**

After completing this lesson, you will be able to:

- Use Windows principals and identities to control access to an application.
- Implement custom authentication by using generic principals and identities.

# How Role-Based Security Works



- **Authentication and authorization**
- **Identities**
  - Windows identity
  - Generic identity
  - Custom identity
- **Principals**
  - Windows principal
  - Generic principal
  - Custom principal

**Introduction**

Principal and identity objects are used to access information about the user. You can access information about Windows users and the Windows groups that they belong to by using the **WindowsPrincipal** and **WindowsIdentity** objects. To access user information based on your own custom authentication scheme, you can use the **GenericPrincipal** and **GenericIdentity** objects or implement your own custom principal and identity objects.

**Authentication**

Before you determine what a specific user is authorized to do, you must verify the identity of that user. *Authentication* is the process of discovering and verifying the identity of a user by examining the user's credentials and then validating those credentials against some authority.

**Authorization**

The next step after authentication is authorization. *Authorization* is the process of determining whether a user is allowed to perform a requested action. Authorization occurs after authentication and uses information about a user's identity and roles to determine what resources that user can access.

**Identities**

An identity object encapsulates information about the user or entity being validated, such as the user name and authentication type. The .NET Framework provides three kinds of identity objects:

- Windows identity

  Represents the identity of the user based on a method of authentication that is supported by the Windows operating system. A Windows identity provides the ability to impersonate another user, so resources can be accessed on behalf of that other user. The **WindowsIdentity** class implements this kind of identity.

- Generic identity

  Represents the identity of the user based on a custom authentication method, which is defined by the application. The **GenericIdentity** class implements this kind of identity.

- Custom identity

  Represents an identity that encapsulates custom user information. Any custom identity class must implement the **IIdentity** interface.

All identity classes must implement the **IIdentity** interface. The **IIdentity** interface has three public properties, listed in the following table.

| Property | Description |
|---|---|
| **Name** | The name of the current user, represented as a string. |
| **IsAuthenticated** | A Boolean value indicating whether the user has been authenticated. |
| **AuthenticationType** | The type of authentication used, represented as a string. |

## Principals

A principal object represents the security context under which code is running. This includes the identity of the user, as represented by an associated identity object, and the roles associated with the user.

A role defines a group of related users of an application. For example, a banking application might impose limits on the withdrawal amounts that can be transacted, based on role. In this scenario, tellers might be authorized to process withdrawals that are less than a specified amount, while managers might be allowed to process withdrawals above the specified amount.

Role-based security in the .NET Framework supports three kinds of principals:

- Windows principal

  Represents Windows users and their roles. The roles are the Windows groups that the user is a member of. The **WindowsPrincipal** class implements this kind of principal.

- Generic principal

  Represents users and roles that are independent of Windows users and their roles. Essentially, the generic principal is a simple solution for application authentication and authorization. The **GenericPrincipal** class implements this kind of principal.

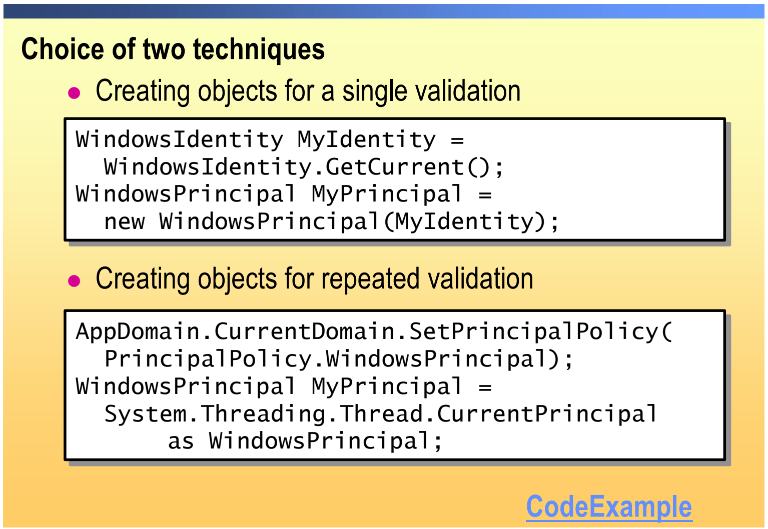
- Custom principal

  Represents application-specific role information. Any custom principal class must implement the **IPrincipal** interface.

All principal classes implement the **IPrincipal** interface. The **IPrincipal** interface has an **Identity** property that stores the identity object related to the current principal and an **IsInRole** method that determines whether the current principal belongs to the specified role.

A principal object is bound to a call context object in an application domain object. A default call context is always created for each new application domain, so there is always a call context available. When a new thread is created, a call context object is also created for that thread. The principal object reference is automatically copied from the creating thread to the new thread's call context.

# How to Create WindowsPrincipal and WindowsIdentity Objects



**Choice of two techniques**

- Creating objects for a single validation

```
WindowsIdentity MyIdentity =
  WindowsIdentity.GetCurrent();
WindowsPrincipal MyPrincipal =
  new WindowsPrincipal(MyIdentity);
```

- Creating objects for repeated validation

```
AppDomain.CurrentDomain.SetPrincipalPolicy(
  PrincipalPolicy.WindowsPrincipal);
WindowsPrincipal MyPrincipal =
  System.Threading.Thread.CurrentPrincipal
       as WindowsPrincipal;
```

CodeExample

### Introduction

There are two ways to create **WindowsPrincipal** objects, depending on whether code must repeatedly perform role-based validation or must perform it only once.

### Procedure: Creating a WindowsPrincipal object for a single validation

When you must perform role-based validation once, you can call the **WindowsIdentity.GetCurrent** method.

1. Initialize a new instance of the **WindowsIdentity** class by calling the static **WindowsIdentity.GetCurrent** method.

   In the first example shown on the slide, the **GetCurrent** method queries the current Windows account and places information about that account into the identity object.

2. Create a new instance of the **WindowsPrincipal** class and pass the value of a **WindowsIdentity** object to it.

### Procedure: Creating a WindowsPrincipal object for repeated validation

When you must perform role-based validation repeatedly, you can call the Set**PrincipalPolicy** method on the **System.AppDomain** object.

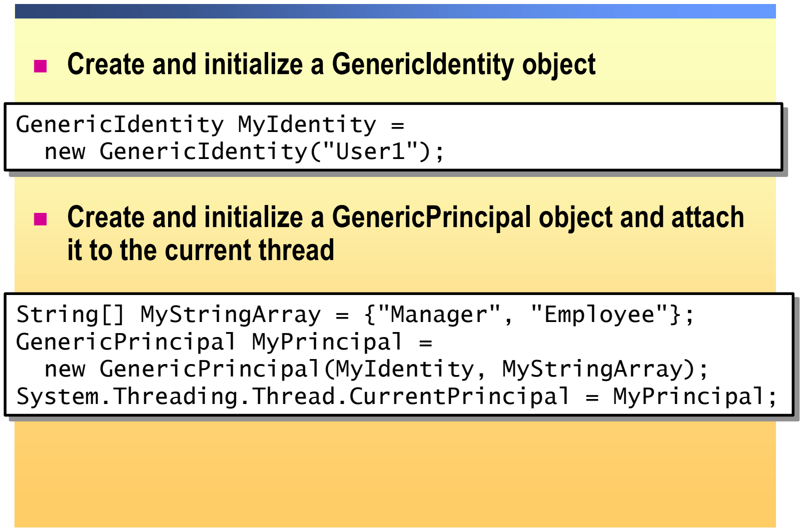
1. Call the static **SetPrincipalPolicy** method on the **System.AppDomain** object, passing it a **PrincipalPolicy** enumeration value that indicates what the policy should be. Supported values are **NoPrincipal**, **UnauthenticatedPrincipal**, and **WindowsPrincipal**.

2. With the policy set, use the **Thread.CurrentPrincipal** property to retrieve the principal that encapsulates the current Windows user.

The following example creates an instance of a **WindowsIdentity** object and a **WindowsPrincipal** object, and gathers identity and principal information from the objects:

```
using System;
using System.Threading;
using System.Security.Principal;
public class Class1
{
  public static int Main(string[] args)
  {
      WindowsIdentity MyIdentity =
              WindowsIdentity.GetCurrent();
      WindowsPrincipal MyPrincipal = new
       WindowsPrincipal(MyIdentity);

      //Principal values.
      string Name = MyPrincipal.Identity.Name;
      string Type = MyPrincipal.Identity.AuthenticationType;
      string Auth =
          MyPrincipal.Identity.IsAuthenticated.ToString();
      //Identity values.
      string IdentName = MyIdentity.Name;
      string IdentType = MyIdentity.AuthenticationType;
      string IdentIsAuth =
                    MyIdentity.IsAuthenticated.ToString();
      string ISAnon = MyIdentity.IsAnonymous.ToString();
      string IsG = MyIdentity.IsGuest.ToString();
      string IsSys = MyIdentity.IsSystem.ToString();
      string Token = MyIdentity.Token.ToString();
      …
      return 0;
  }
}
```

# How to Create GenericPrincipal and GenericIdentity Objects



- Create and initialize a GenericIdentity object

```
GenericIdentity MyIdentity =
  new GenericIdentity("User1");
```

- Create and initialize a GenericPrincipal object and attach it to the current thread

```
String[] MyStringArray = {"Manager", "Employee"};
GenericPrincipal MyPrincipal =
  new GenericPrincipal(MyIdentity, MyStringArray);
System.Threading.Thread.CurrentPrincipal = MyPrincipal;
```

### Introduction

You can use the **GenericIdentity** class in conjunction with the **GenericPrincipal** class to implement role-based security that is independent of the Windows security system. For example, you can prompt a user for a name and password, check them against a database or a Lightweight Directory Access Protocol (LDAP) directory, and then create identity and principal objects based on the values in the database.

### Procedure: Implementing role-based security by using GenericIdentity and GenericPrincipal objects

There are three mains steps for implementing role-based security by using **GenericIdentity** and **GenericPrincipal** objects.

1. Create a new instance of the **GenericIdentity** class and initialize it with the name that you want it to hold.
2. Create a new instance of the **GenericPrincipal** class and initialize it with the previously created **GenericIdentity** object and an array of strings that represent the roles that you want associated with this principal. The second example on the slide specifies an array of strings that represent a Manager role and a Teller role. The **GenericPrincipal** object is then initialized with the previous **GenericIdentity** object and the string array.
3. Attach the principal to the current thread, as shown on the slide. Attaching the principal to the current thread is valuable in situations where the principal must be validated several times, it must be validated by other code running in your application, or it must be validated by a **PrincipalPermission** object.

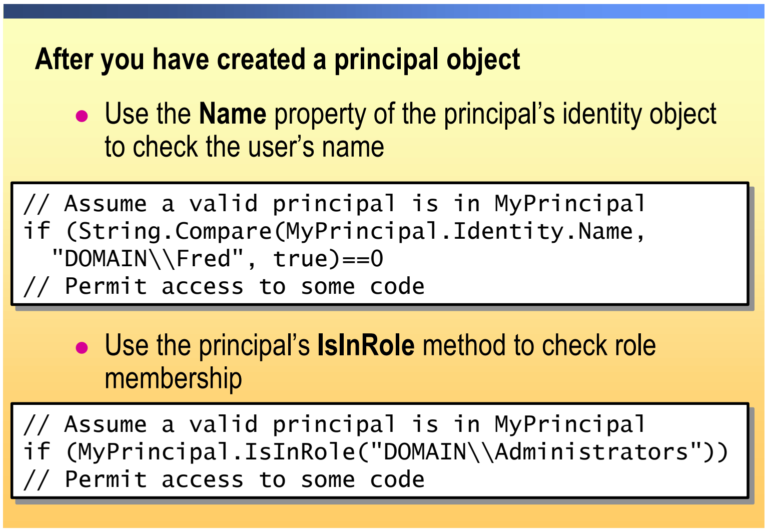
**Important** The code attaching the principal to the current thread must have been granted the **ControlPrincipal** member of the **SecurityPermission** class to successfully attach the principal to the thread.

**Example**

The following example creates an instance of a **GenericPrincipal** and a **GenericIdentity**:

```
using System;
using System.Security.Principal;
using System.Threading;
public class Class1
{
  public static int Main(string[] args)
  {
      // Prompt user for username and password, and then
      // authenticate those credentials against a database
      // or other authority. For this sample, the username
      // is hard-coded.
      String Username = "Joe";

      // Assuming authentication succeeds, create the
      // GenericIdentity representing the user.
      GenericIdentity MyIdentity = new
                                GenericIdentity(Username);

      // Look up the roles the user belongs to in a database
      // or other location. For this sample, the roles are
      // hard-coded.
      String[] MyRoles = {"Manager", "Teller"};

      // Create the GenericPrincipal representing the user
      // and associated roles.
      GenericPrincipal MyPrincipal = new
                 GenericPrincipal(MyIdentity, MyRoles);

      // Store the principal on the thread, so it can be
      // retrieved and used by other code.
      Thread.CurrentPrincipal = MyPrincipal;
  }
}
```

# How to Use Principals and Identities to Control Access to an Application



**After you have created a principal object**

- Use the **Name** property of the principal's identity object to check the user's name

```
// Assume a valid principal is in MyPrincipal
if (String.Compare(MyPrincipal.Identity.Name,
  "DOMAIN\\Fred", true)==0
// Permit access to some code
```

- Use the principal's **IsInRole** method to check role membership

```
// Assume a valid principal is in MyPrincipal
if (MyPrincipal.IsInRole("DOMAIN\\Administrators"))
// Permit access to some code
```

**Introduction**

After you have created a principal object, you can add code to your application to control access to your application based on elements of a user's identity.

**Example**

You can control access to code based on the user's name, as shown in the first example on the slide. The example the following code uses a case-insensitive string comparison to see if the user's name is DOMAIN\Fred. The first two parameters to the static **String.Compare** method are the strings to be compared, and the third parameter tells the method to do a case-insensitive comparison.

```
// Assume a valid principal is in MyPrincipal
if (String.Compare(MyPrincipal.Identity.Name, “DOMAIN\\Fred”,
true)==0
// Permit access to some code
```

For **WindowsIdentity** objects, the name represents the user's login name, including the domain.

**Example**

You can check role membership by calling the **IsInRole** method on the principal object, as shown in the second example on the slide. The example on the slide checks if the user belongs to the DOMAIN\Administrators role.

For **WindowsPrincipal** objects, a role maps to a Windows group, including the domain. When checking for membership in built-in Windows groups, you can use the **WindowsBuiltInRole** enumeration. The following example uses a hard-coded string in the call to determine if the user is a member of the built-in Administrators role:

```
MyPrincipal.IsInRole("BUILTIN\\Administrators");
```

The code works, but it is not easily localized. The following example uses the **WindowsBuiltInRole** enumeration instead, and is more easily localized:

```
MyPrincipal.IsInRole(WindowsBuiltInRole.Administrator);
```

**Note** .NET Framework role-based security does not provide access to COM+ roles with these mechanisms. Classes in the **System.EnterpriseServices** namespace must be used to gain access to COM+ role-based security information.

# Demonstration: Using Role-based Security to Control Access to an Application

**In this demonstration, you will see how an application uses role-based security**

**Introduction**

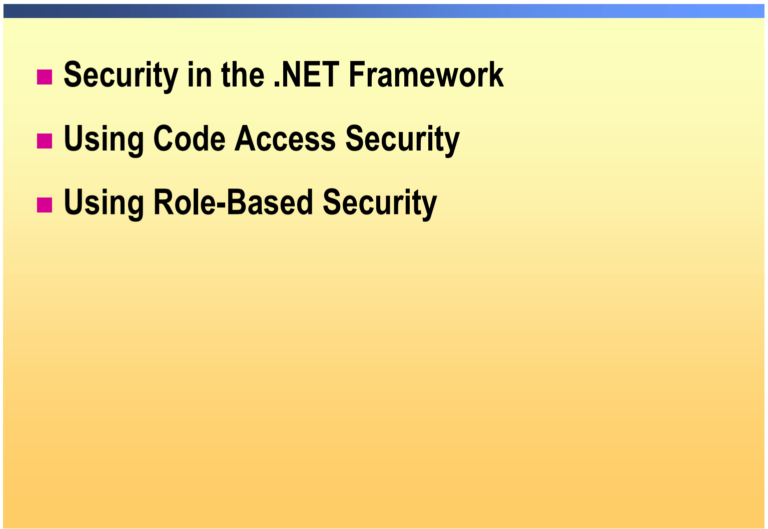
In this demonstration, you will see how an application uses role-based security.

**Instructions**

► **To view code that uses role-based security**

1. Open the RoleBasedSecurity.sln solution file in *install_folder*\Democode\Mod10\Mod10_01\RoleBasedSecurity.
2. Open the RBSForm.cs form and view its code.
3. Note how the current **WindowsPrincipal** is obtained in the **RBSForm** constructor.
4. Set breakpoints in the **CheckUserBtn_Click** and **CheckRoleBtn_Click** methods.
5. Run the application.
6. In the **User Name** box, type your user name, making sure to include the domain, for example, ***computername*\myuser**.
7. Click **Check User Name**.
8. Step through the code in the event handler and note how the user name check is performed. When you are done with the code in this method, press F5 to continue execution of the application.
9. In the **Role** box, type a role that you are a member of, such as **BUILTIN\Users**.
10. Click **Check Role**.
11. Step through the code in the event handler and note out how the role membership check is performed. Then click **Exit** to quit the application.

# Review



- Security in the .NET Framework
- Using Code Access Security
- Using Role-Based Security

1. What are the two major areas of security in the .NET Framework?

   **The two major areas of security in the .NET Framework are code access security and role-based security.**

2. How are evidence, security policy, and permissions related?

   **The runtime uses security policy to map evidence to permissions.**

3. What are the three types of permission requests that you can make?

   **You can make minimum, optional, and refused permission requests.**

4. What kind of permissions should you include in a minimum permission request?

   **Include in a minimum permission request any permission that is absolutely necessary for your code to run successfully.**

5. What are the two ways to configure the security policy to test an application?

   **You can use the .NET Framework Configuration tool (Mscorcfg.msc) or the Code Access Security tool (Caspol.exe) to configure the security policy.**

6. What are authentication and authorization?

   ***Authentication* is the process of validating a user's credentials. *Authorization* is the process of deciding whether an authenticated user is allowed to access a resource.**

7. Describe when you would use a **WindowsPrincipal** object and when you would use a **CustomPrincipal** object to implement role-based security.

   **Use a WindowsPrincipal when your role-based security decisions are based on Windows users and groups. Use a CustomPrincipal when your role-based security decisions are based on another authentication mechanism, such as a SQL Server database.**

8. What method of the **Principal** class do you use to perform role check?

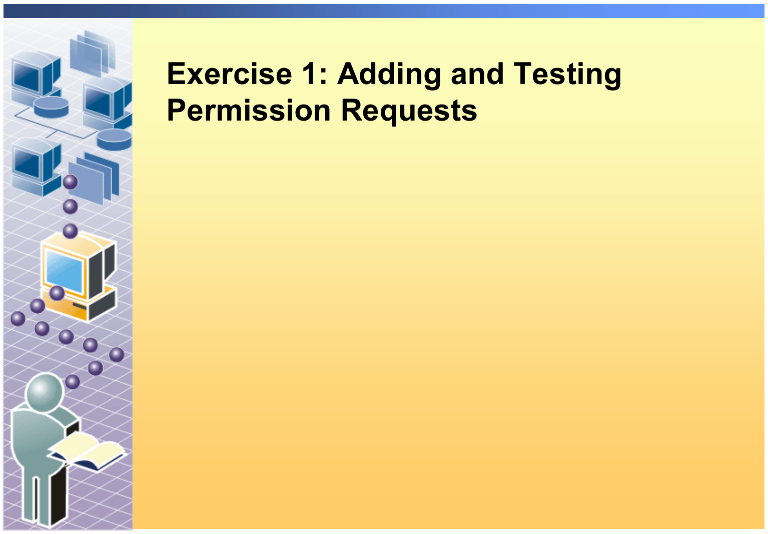
   **You use the IsInRole method of the Principal class.**

9. What are the three main steps to implement role-based security with **GenericIdentity** and **GenericPrincipal** objects in your application?

   **The three main steps to implement role-based security with GenericIdentity and GenericPrincipal objects are:**

   - **Create a new instance of the GenericIdentity class and initialize it with the name you want it to hold.**
   - **Create a new instance of the GenericPrincipal class and initialize it with the previously created GenericIdentity object and an array of strings that represent the roles that you want associated with this principal.**
   - **Attach the principal to the current thread. Attaching the principal to the current thread is valuable in situations where the principal must be validated several times, it must be validated by other code running in your application, or it must be validated by a PrincipalPermission object.**

# Lab 10.1: Adding and Testing Permission Requests



Exercise 1: Adding and Testing Permission Requests

**Objectives**

After completing this lab, you will have demonstrated your ability to:

- Add assembly permission requests.
- Test an application by running it in different security contexts.

---

**Note** This lab focuses on the concepts in Module 10, “Securing Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. As a result, this lab may not comply with Microsoft security recommendations.

---

**Prerequisites**

Before working on this lab, you must have:

- The knowledge and skills to develop a simple Windows Forms application by using a Visual Studio .NET–compatible programming language.
- The knowledge and skills to debug an application by using Visual Studio .NET.
- The knowledge and skills to add permission requests.

**Scenario**

To follow best practices, you want to document the permissions that your application requires and to run it with the fewest privileges possible. To do this with the .NET Framework security system, you will use permission requests. This will also help you to configure the security policy if you decide to deploy your application in other security contexts.

In this lab, you will add minimum permission requests for the permissions that you need and an optional permission request for no permissions to prevent any additional permissions from being granted to your application.

**Lab setup**

There are starter and solution files associated with this lab. Browse to *install_folder*\Labfiles\Lab10_1\Ex01\Starter to find the starter files, and browse to *install_folder*\Labfiles\Lab10_1\Ex01\Solution to find the solution files. If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

**Estimated time to complete this lab: 30 minutes**

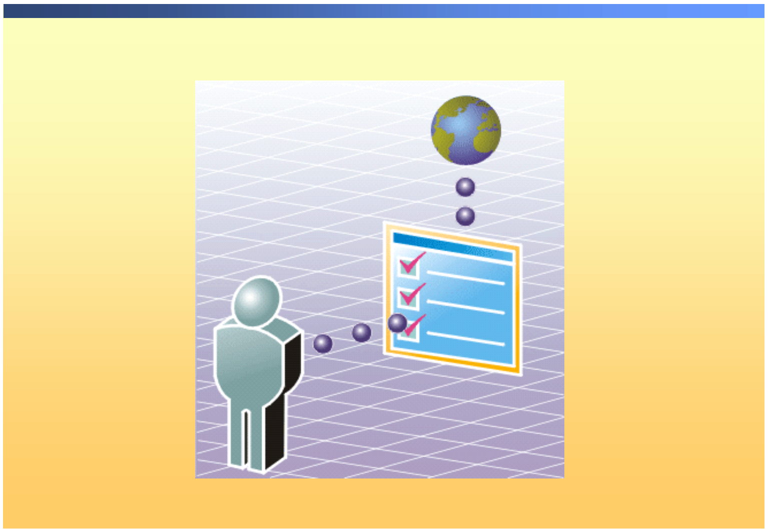
# Exercise 1
# Adding and Testing Permission Requests

In this exercise, you will add some minimum permission requests to the Expense Report application. You will also test the Expense Report application in different security contexts.

| Tasks | Additional information |
|---|---|
| **1.** Open the ExpenseReport.sln solution file in Visual Studio .NET. Browse to *install_folder*\Labfiles\Lab10_1\Ex01\Starter to find the project files. | **a.** For more information about opening a project file and starting an application, see the following resource:<br>- The Visual Studio .NET Help documentation. For additional information about opening a project file, in Search, select the **Search in titles only** check box, then search by using the phrase **Open Project Dialog Box**. For additional information about starting an application in the Designer, in Index, search by using the phrase **Debugging Windows Applications**. |
| **2.** Open the AssemblyInfo.cs file. Add minimum permission requests for the following permissions:<br>- Use of all windows.<br>- Use of Isolated Storage, isolated by domain and user. Use the following syntax:<br>`[assembly: IsolatedStorageFilePermission( SecurityAction.RequestMinimum, UsageAllowed = IsolatedStorageContainment.DomainIsolationByUser)]`<br>- Use of Web access for the Expense Report Web service. Use the following syntax:<br>`[assembly:WebPermission( SecurityAction.RequestMinimum, Connect = "http://localhost/ExpenseReportWebService/ExpenseReportWebService.asmx")]`<br>- Add an optional permission request for no permissions. | **a.** For more information about permission requests, see the following resources:<br>- Practice: Adding Permission Requests in Module 10, “Securing Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).* This practice contains information about how to add the minimum permission request for the use of all windows and the optional permission request for no permissions.<br>- Lesson: Using Code Access Security in Module 10, “Securing Windows Forms Applications,” in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET).* This lesson contains information about how to make permission requests.<br>- The .NET Framework SDK documentation. In Search, select the **Search in titles only** check box, then search by using the phrase **Requesting Permissions**. |

| Tasks | Additional information |
|---|---|
| **3.** Test the permission requests.<br>**a.** Build and run the application in Visual Studio to make sure it still works.<br>**b.** Copy the ExpenseReport.exe file from the bin(\debug) folder of the project to the C:\Test\LocalIntranet folder.<br>**c.** Double-click the application and attempt to run it. Notice that you get a policy exception. | **a.** In step 3.c., the application will not run because the LocalIntranet permission set limits Web access to the same site that the assembly was downloaded from. If the application were installed on localhost, it would run in the LocalIntranet permission set.<br>**b.** For more information about the default security policy, see the following resources:<br>• Lesson: Using Code Access Security in Module 10, "Securing Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows (Visual C# .NET)*. This lesson contains information about configuring the security policy.<br>• The .NET Framework SDK documentation. In Search, select the **Search in titles only** check box, then search by using the phrase **Default Security Policy**. |

# Course Evaluation



Your evaluation of this course will help Microsoft understand the quality of your learning experience.

To complete a course evaluation, go to http://www.metricsthatmatter.com/survey.

Microsoft will keep your evaluation strictly confidential and will use your responses to improve your future learning experience.

msdn® training

# Appendix A: Using Filled Shapes and Images

**Contents**

Microsoft®

Information in this document, including URL and other Internet Web site references, is subject to change without notice. Unless otherwise noted, the example companies, organizations, products, domain names, e-mail addresses, logos, people, places, and events depicted herein are fictitious, and no association with any real company, organization, product, domain name, e-mail address, logo, person, place or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Microsoft Corporation.

Microsoft may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any written license agreement from Microsoft, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property.

# Instructor Notes

**Presentation:**
**60 minutes**

**Lab:**
**00 minutes**

This appendix provides students with an overview of how to create filled shapes and images by using GDI+. Students learn how to use solid, hatch, texture, linear gradient, path gradient, and transparent brushes to fill shapes. They will also learn how to load, skew, rotate, reflect, crop, and scale images.

After completing this module, students will be able to:

- Create and use textured, hatched, and gradient brushes to fill shapes.
- Manipulate bitmap images and use them in a Windows Forms application.

**Required materials**

To teach this appendix, you need the Microsoft® PowerPoint® file 2555A_XA.ppt.

**Preparation tasks**

To prepare for this module:

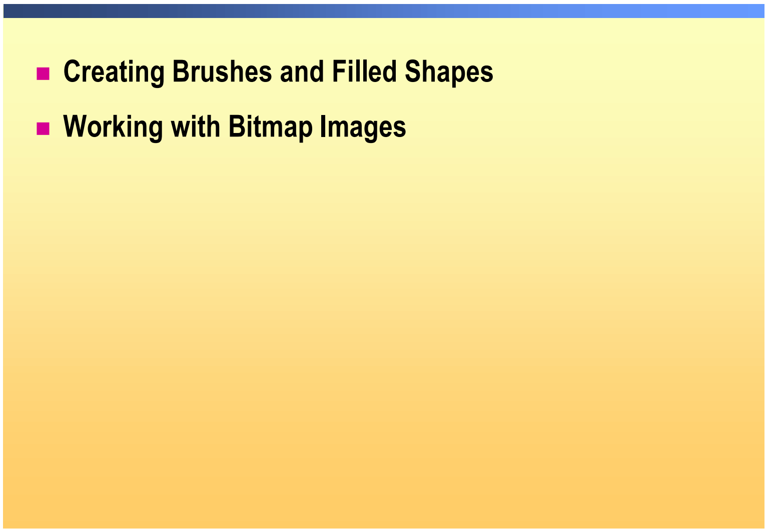
- Read all of the materials for this appendix.
- Complete the practices.

# How to Teach This Appendix

This section contains information that will help you to teach this appendix.

- This is an optional teachable appendix. Find out the interest of students in GDI+. Depending on time and the interest level of students, you may teach this appendix.
- The appendix consists of two practices that will help reinforce the concepts taught in the lessons.

# Overview



- Creating Brushes and Filled Shapes
- Working with Bitmap Images

### Introduction

The Microsoft® .NET Framework common language runtime uses an advanced implementation of the Microsoft Windows® graphics design interface (GDI) called GDI+. GDI+ allows you to create graphics, draw text, and manipulate graphical images as objects. GDI+ is designed to offer performance as well as ease of use. You can use GDI+ to render graphical images on Windows Forms and controls as well as on any Graphics object. GDI+ has fully replaced GDI and is now the only way to render graphics programmatically in Windows Forms applications.

Module 6, "Reporting and Printing in Windows Forms Applications," in Course 2555A, *Developing Microsoft .NET Applications for Windows Applications (Visual C#™ .NET)*, introduces the basics of GDI+ and how to draw text and shapes by using the **Graphics** object methods.

In this appendix, you will learn about some more advanced tasks that you can perform with GDI+, such as how to create and use additional brush types and manipulate images.

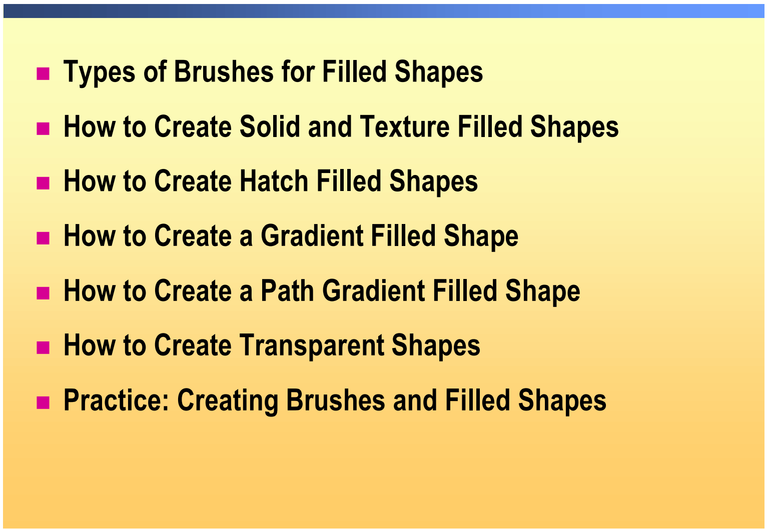
For more information about GDI+ beyond what is included in this appendix, see the Microsoft .NET Framework software development kit (SDK) documentation and *Programming Microsoft Windows with C#,* by Charles Petzold.

### Objectives

After completing this appendix, you will be able to:

- Create and use textured, hatched, and gradient brushes to fill shapes.
- Manipulate bitmap images and use them in a Windows Forms application.

# Lesson: Creating Brushes and Filled Shapes



- Types of Brushes for Filled Shapes
- How to Create Solid and Texture Filled Shapes
- How to Create Hatch Filled Shapes
- How to Create a Gradient Filled Shape
- How to Create a Path Gradient Filled Shape
- How to Create Transparent Shapes
- Practice: Creating Brushes and Filled Shapes

**Introduction**

Two-dimensional (2-D) shapes are vector objects that can be drawn as an outline by using a pen and, in the case of a closed shape, filled by using a brush. Rectangles and ellipses are good examples of closed shapes because they consist of an outline and an interior. Closed shapes can be drawn as either an outline by using a **Draw*xxx*** method or a filled shape by using a **Fill*xxx*** method.

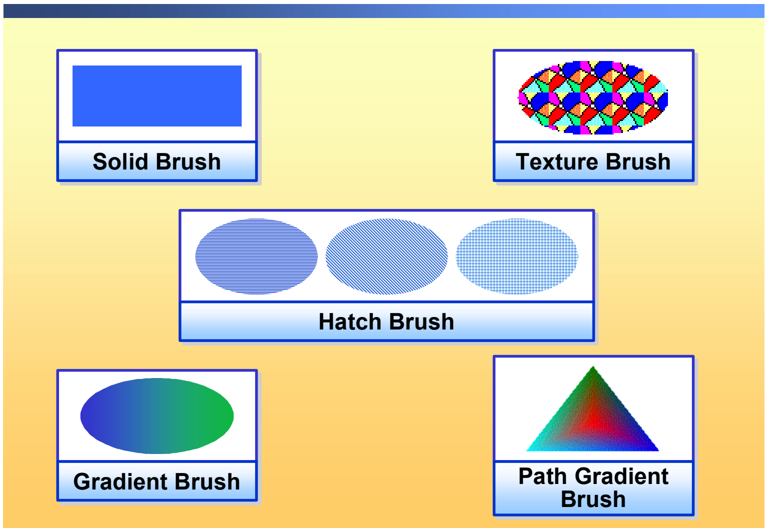
In this lesson, you will learn how to draw filled shapes by using each of the different types of brushes supported by GDI+.

**Lesson objectives**

After completing this lesson, you will be able to:

- Draw filled shapes by using GDI+ methods.
- Create hatch filled shapes.
- Create texture filled shapes.
- Create gradient filled shapes.
- Create path gradient filled shapes.
- Create transparent shapes.

# Types of Brushes for Filled Shapes



Solid Brush

Texture Brush

Hatch Brush

Gradient Brush

Path Gradient Brush

## Introduction

Applications use filled shapes for a variety of tasks. Spreadsheet applications, for example, use filled shapes to construct charts and graphs, and drawing and painting applications use filled shapes to enable the user to draw vector objects.

GDI+ provides five brush classes for filling the interiors of closed shapes. The five brush types are **SolidBrush**, **TextureBrush**, **HatchBrush**, **LinearGradientBrush**, and **PathGradientBrush**. All of these classes inherit the members of the **Brush** class.

## Solid brushes

The **SolidBrush** is the simplest form of brush and is used to fill a shape with solid color. **SolidBrush** is found in the **System.Drawing** namespace.

## Texture brushes

The **TextureBrush** is similar to a solid brush but uses an image rather than a solid color to fill a shape. With a texture brush, you can fill a shape with any pattern that is available in a bitmap form. **TextureBrush** is found in the **System.Drawing** namespace.

## Hatch brushes

The **HatchBrush** enables you to select from more than 50 preset hatch style patterns (enumerations of **HatchStyle**) to fill in a shape. When you fill a shape with a hatch brush, you specify a line color, a background color, and a hatch style. **HatchBrush** is found in the **System.Drawing.Drawing2D** namespace.

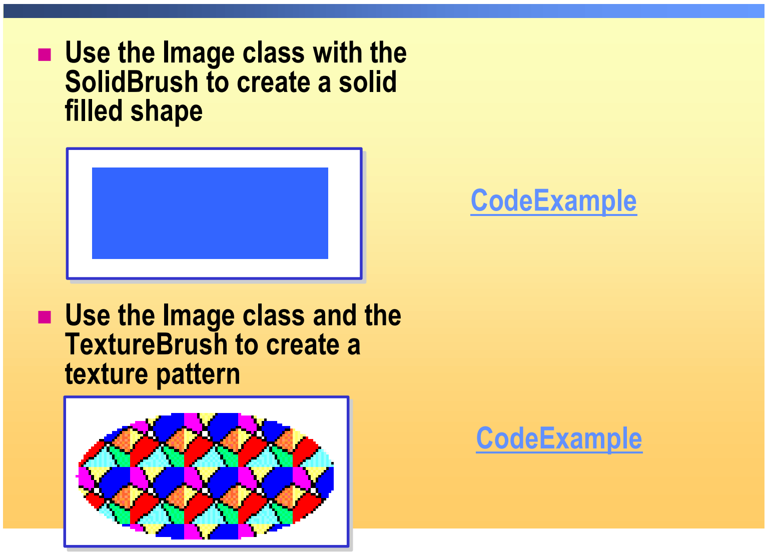
## Linear gradient brushes

You can use a linear gradient brush to fill a shape with color that changes gradually as you move across the shape horizontally, vertically, or diagonally. **LinearGradientBrush** is found in the **System.Drawing.Drawing2D** namespace.

## Path gradient brushes

A path gradient brush provides you with many additional capabilities beyond those of a linear gradient brush. You use a path gradient brush when you want to create a shape that begins with a single color on the inside and gradually changes to one or more other colors along the perimeter of the shape. **PathGradientBrush** is found in the **System.Drawing.Drawing2D** namespace.

# How to Create Solid and Texture Filled Shapes



- Use the Image class with the SolidBrush to create a solid filled shape

CodeExample

- Use the Image class and the TextureBrush to create a texture pattern

CodeExample

### Introduction

You can fill a closed shape with a solid color or a texture by using the **Image** class and the **SolidBrush**/**TextureBrush** class. Both **SolidBrush** and **TextureBrush** are included in the **System.Drawing** namespace.

### Procedure: How to create a solid filled shape

The following example shows how to draw a solid blue ellipse.

```
SolidBrush myBrush = new SolidBrush(Color.Blue);
e.Graphics.FillRectangle(myBrush, 0, 0, 120, 40);
```

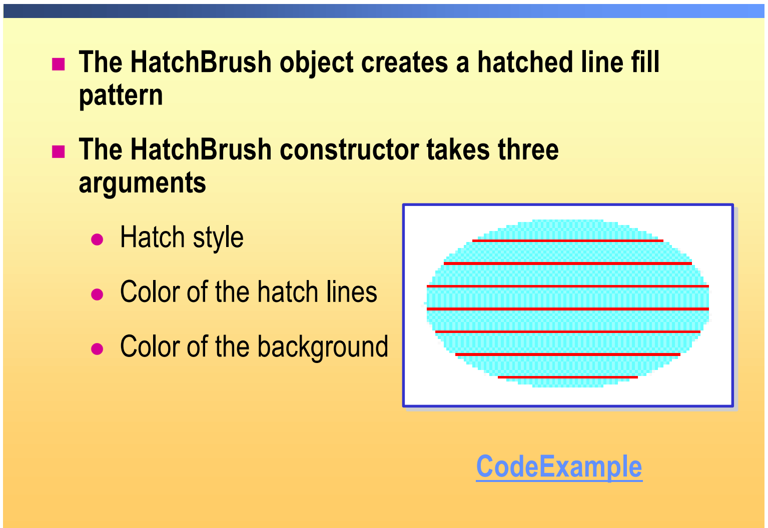
### Procedure: How to create a texture filled shape

The following example fills an ellipse with an image. The code constructs an **Image** object, and then passes the address of that **Image** object as an argument to a **TextureBrush** constructor. The final statement fills the ellipse with repeated copies of the image.

```
imageFileName = "C:\Images\Image01.jpg";
Image brushImage = new Bitmap(imageFileName);
TextureBrush myImageBrush = new TextureBrush(brushImage);
e.Graphics.FillEllipse(myImageBrush, 0, 0, 135, 55);
```

**Note** You can also use a brush to draw lines by constructing a pen based on the brush rather than a solid color. For more information about using a brush as the source of a line color, see "Drawing a Line Filled with a Texture" in the .NET Framework SDK documentation.

# How to Create Hatch Filled Shapes



- The HatchBrush object creates a hatched line fill pattern
- The HatchBrush constructor takes three arguments
  - Hatch style
  - Color of the hatch lines
  - Color of the background

CodeExample

**Introduction**

A hatch brush allows you to select from a large variety of preset hatched line patterns to fill in a shape.

Each hatch pattern is made from two colors: one for the background and one for the lines that form the pattern over the background. To fill a closed shape with a hatch pattern, use a **HatchBrush** object. **HatchBrush** and **HatchStyle** are included in the **System.Drawing.Drawing2D** namespace.

**Procedure**

The **HatchBrush** constructor takes three arguments:

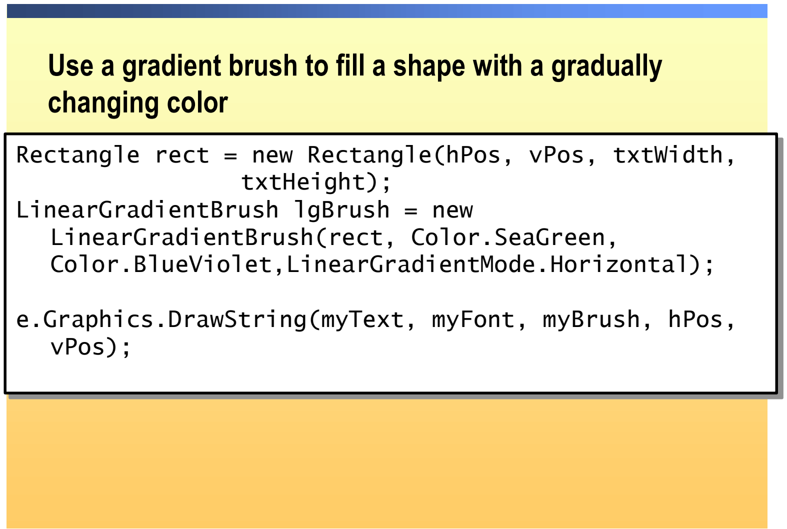
- Hatch style
- Color of the hatch lines
- Color of the background

The hatch style argument can be any of the more than 50 values from the **HatchStyle** enumeration, such as **Cross**, **Horizontal**, **OutlinedDiamond**, **Sphere**, **Vertical**, and **ZigZag**.

The following example demonstrates how to fill an ellipse with a horizontal line hatch pattern of red on a cyan background:

```
Imports System.Drawing.Drawing2D
…
HatchBrush myHatchBrush = new
  HatchBrush(HatchStyle.Horizontal, Color.Red, Color.Cyan);

e.Graphics.FillEllipse(myHatchBrush, 0, 200, 120, 40);
```

# How to Create a Gradient Filled Shape



**Use a gradient brush to fill a shape with a gradually changing color**

```
Rectangle rect = new Rectangle(hPos, vPos, txtWidth,
                 txtHeight);
LinearGradientBrush lgBrush = new
   LinearGradientBrush(rect, Color.SeaGreen,
   Color.BlueViolet,LinearGradientMode.Horizontal);

e.Graphics.DrawString(myText, myFont, myBrush, hPos,
   vPos);
```

### Introduction

You can fill a shape with a gradually changing color by using a gradient brush. For example, you can use a horizontal gradient to fill a shape with color that changes gradually as you move from the left edge of the shape to the right edge.

A linear gradient changes color as you move horizontally, vertically, or diagonally across a closed shape. For creating linear gradients, GDI+ provides a **LinearGradientBrush** which inherits from the **Brush** class and is included in the **System.Drawing.Drawing2D** namespace.

The orientation of a **LinearGradientBrush** (horizontal, vertical, or diagonal) is determined by the orientation of a line connecting the two points used in the construction of the brush. When the brush is constructed by using a rectangle rather than two points, the orientation is determined by the value of either the **LinearGradientMode** parameter or the **angle** parameter.

By default, the color in a linear gradient changes uniformly from one endpoint to another. However, you can customize a linear gradient in many ways. For example, you can create a nonuniform linear gradient or a gradient that begins with one color in the middle and blends into a second color at both ends.

For more information about customizing the appearance of a **LinearGradientBrush**, see “Creating a Linear Gradient and Filling Shapes with a Gradient Brush” in the .NET Framework SDK documentation, or see “LinearGradientBrush Members” in the .NET Framework Class Library.

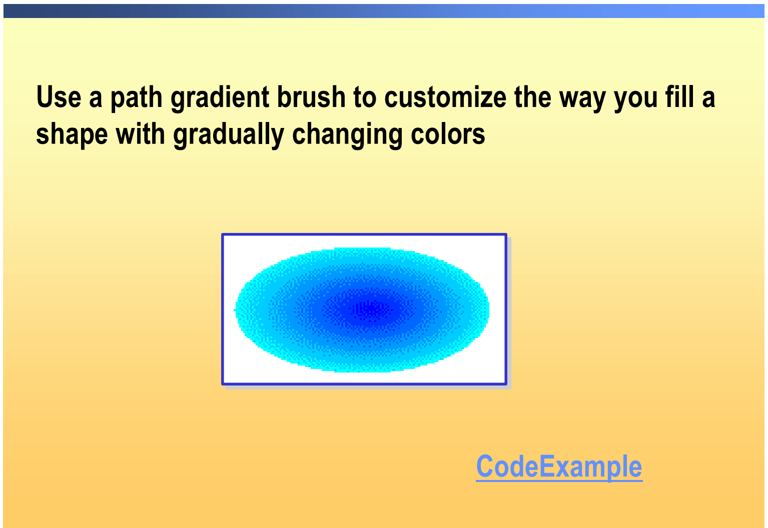
### Procedure: How to create a horizontal linear gradient

The following example draws a string (myText) by using a linear gradient brush. The size of the brush is determined by a rectangle, which is constructed to be the same size as the string when it is drawn on the graphics object.

```
Rectangle rect = new Rectangle(hPos, vPos, txtWidth,
                               txtHeight);
LinearGradientBrush lgBrush = new LinearGradientBrush(rect,
                        Color.SeaGreen, Color.BlueViolet,
                        LinearGradientMode.Horizontal);
e.Graphics.DrawString(myText, myFont, myBrush, hPos, vPos);
```

**Note** If the object being drawn with a linear gradient brush extends beyond the size of the brush, the color gradient repeats itself until the entire object is filled.

# How to Create a Path Gradient Filled Shape



Use a path gradient brush to customize the way you fill a shape with gradually changing colors

### Introduction

The **PathGradientBrush** class enables you to create gradients with more than two colors and provides additional methods for customizing a gradient fill. For example, you can specify one color for the center of a path and another color for the outside edge. You can also specify separate colors for each of several points along the perimeter of a path.

### Procedure: How to fill an ellipse with a path gradient

The following example fills shapes with a path gradient brush. The center color is set to white and the color of the outside edge is set to a predefined color.

```
PathGradientBrush pgBrush = new PathGradientBrush(pathShape1);
pgBrush.CenterColor = Color.White;
pgBrush.SurroundColors = colorsShape1;

e.Graphics.FillPath(pgBrush, pathShape1);
```

By default, a path gradient brush does not extend outside the boundary of the path. If you use the path gradient brush to fill a figure that extends beyond the boundary of the path, the area of the object that is outside the path is not filled.

For more information about customizing a PathGradientBrush, see “Creating a Path Gradient” in the .NET Framework SDK documentation.

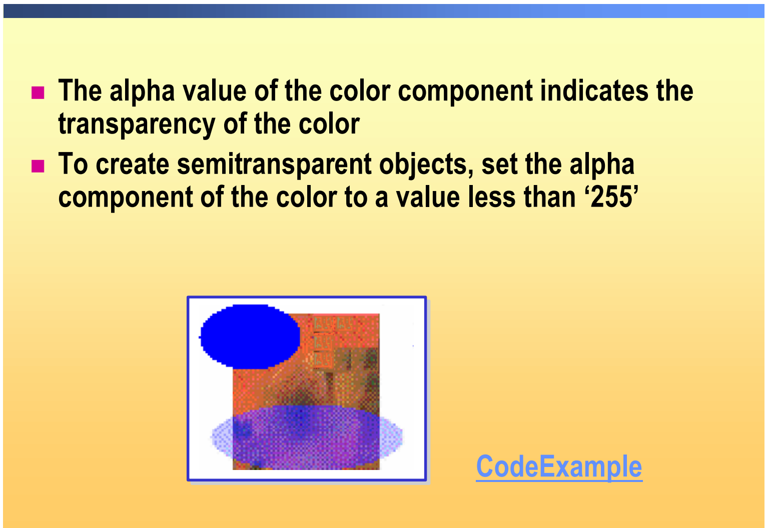
### Procedure: How to set the center point

By default, the center point of a path gradient brush is at the centroid of the path used to construct the brush. You can change the location of the center point by setting the **CenterPoint** property of the **PathGradientBrush** class.

The following example creates a path gradient brush based on an ellipse. The center of the ellipse is at (70, 35), but the center point of the path gradient brush is set to (120, 40).

```
// Create a path that consists of a single ellipse.
GraphicsPath path = new GraphicsPath();
path.AddEllipse(0, 0, 140, 70);

// Use the path to construct a brush.
PathGradientBrush pgBrush2 = new PathGradientBrush(path);

// Set the center point to a location that is not
// the centroid of the path.
pgBrush2.CenterPoint = new PointF(120, 40);

// Set the color at the center of the path to blue.
pgBrush2.CenterColor = Color.FromArgb(255, 0, 0, 255);

// Set the color along the entire boundary
// of the path to aqua.
Color[] colors = {Color.FromArgb(255, 0, 255, 255)};
pgBrush2.SurroundColors = colors;

e.Graphics.FillEllipse(pgBrush2, 0, 0, 140, 70);
```

# How to Create Transparent Shapes



- The alpha value of the color component indicates the transparency of the color
- To create semitransparent objects, set the alpha component of the color to a value less than ‘255’

CodeExample

****************************ILLEGAL FOR NON-TRAINER USE******************************

### Introduction

In GDI+, a color is a 32-bit value with 8 bits each for alpha, red, green, and blue. The alpha value indicates the transparency of the color—the extent to which the color is blended with the background color. Alpha values range from 0 through 255, where 0 represents a fully transparent color, and 255 represents a fully opaque color.

### Procedure

When you fill a shape, you must pass the address of a **Brush** object to one of the fill methods of the **Graphics** class. One parameter of the **SolidBrush** constructor is a **Color** object.

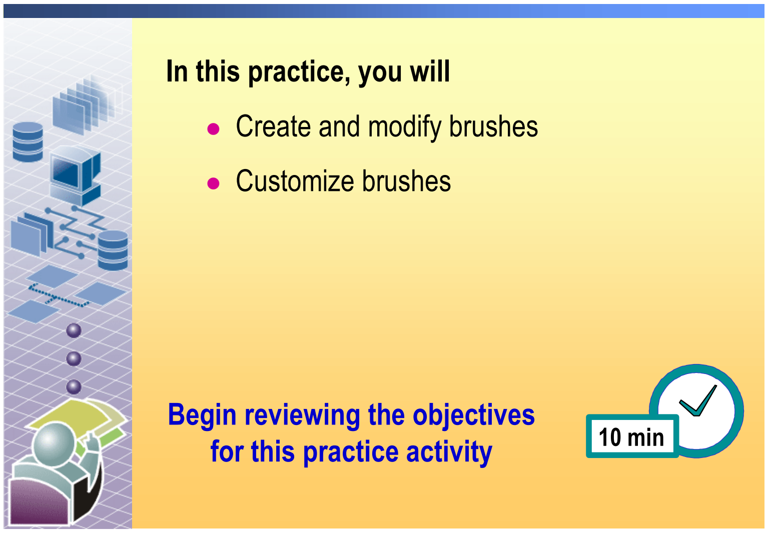
To fill an opaque shape, set the alpha component of the color to 255. To fill a semitransparent shape, set the alpha component to any value from 1 through 254. For a transparent object, set the alpha component value to 0.

When you fill a semitransparent shape, the color of the shape is blended with the colors of the background. The alpha component specifies how the shape and background colors are mixed. Alpha values near 0 place more weight on the background colors, and alpha values near 255 place more weight on the shape color.

The following example draws two ellipses on a textured background. The first ellipse uses an alpha component of 255, so it is opaque. The second ellipse uses an alpha component of 128, so it is semitransparent. Background images can be seen through a semitransparent shape.

```
Bitmap bitmap = new Bitmap(imageFileName);
e.Graphics.DrawImage(bitmap, 50, 50, bitmap.Width,
                        bitmap.Height);

SolidBrush opaqueBrush = new SolidBrush(Color.FromArgb(255,
                                        Color.Blue));
SolidBrush semiTransBrush = new SolidBrush(Color.FromArgb(128,
                                        Color.Blue));

e.Graphics.FillEllipse(opaqueBrush, 0, 200, 45, 30);
e.Graphics.FillEllipse(semiTransBrush, 0, 250, 45, 30);
```

# Practice: Creating Brushes and Filled Shapes



In this practice, you will

- Create and modify brushes
- Customize brushes

Begin reviewing the objectives for this practice activity

## Introduction

You are a software developer at Northwind Traders. You have been asked to create a **Form** class that can be used throughout the company as a splash screen that displays while an application loads.

In this practice, you will work with hatched, textured, gradient, and semitransparent brushes.

## Instructions

### ► Open the practice project

1. Using Windows Explorer, navigate to *install_folder*\Practices\AppendixA\AppendixA_01\Starter.

**Note** If you performed a default installation of the course files, *install_folder* corresponds to C:\Program Files\Msdntrain\2555.

2. Double-click the **Filled Shapes.sln** solution file to open the project.
3. On the **Debug** menu, click **Start**.
4. Click **Show Splash Screen**.

   Identify the brush types used on the splash screen form.

### ► Create and modify brushes

1. Stop the splash screen application, and then open the Code Editor for splashForm.cs.
2. On the **View** menu, point to **Show Tasks**, and then click **Comment**.
3. In the Task List, double-click **TODO: modify TextureBrush**.
4. Enable the code statement that follows the TODO line.
5. In the Task List, double-click **TODO: modify HatchBrush**.
6. Modify the code statement that creates the **HatchBrush** by changing the **HatchStyle** from **LargeCheckerBoard** to **Cross**.

7. In the Task List, double-click **TODO: modify LinearGradientBrush**.
8. Modify the code statement that creates the **LinearGradientBrush** by changing the **LinearGradientMode** from **Vertical** to **Horizontal**.
9. In the Task List, double-click **TODO: create PathGradientBrush**.
10. Add the following code statements below the TODO line.

```
PathGradientBrush brushShape1 = new
                     PathGradientBrush(pathShape1);
brushShape1.CenterColor = Color.White;
brushShape1.SurroundColors = colorsShape1;
```

11. In the Task List, double-click **TODO: draw pathShape1**.
12. Add the following code statement below the TODO line.

```
e.Graphics.FillPath(brushShape1, pathShape1);
```

13. Do any of the code statements that you modified or added require a reference to the **Drawing2D** namespace?

    **Yes. The hatch and gradient brush types require the Drawing2D namespace.**

14. On the **Debug** menu, click **Start**.
15. Click **Show Splash Screen**.

    Examine the change to the splash screen form.

16. Close the splash form, and then close the splash screen application.

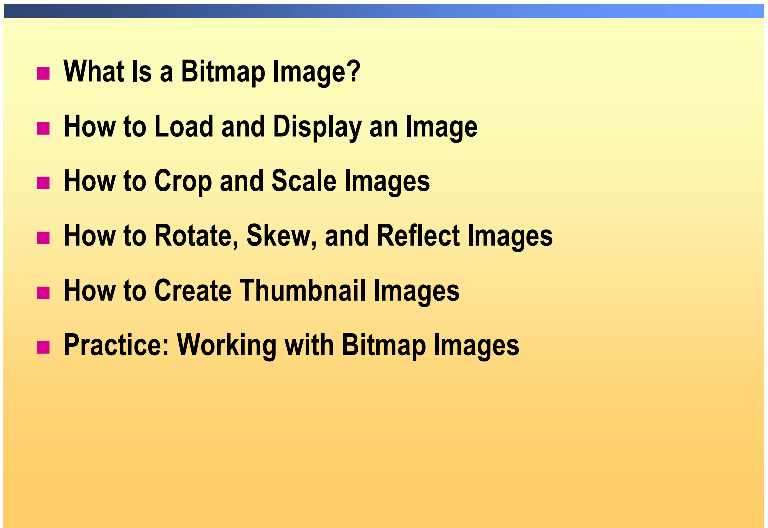
► **Customize brushes**

1. In the Task List, double-click **TODO: customize the linear gradient brush**.
2. Modify the code statement that uses the **SetBlendTriangularShape** method by changing the value passed to the method from **0.0** to **0.5**.
3. In the Task List, double-click **TODO: enable customizations to the path gradient brush**.
4. Enable the code statement that follows the TODO line.
5. On the **Debug** menu, click **Start**.
6. Click **Show Splash Screen**.

   Examine the change to the splash screen form. If necessary, change the code statements back to their original state, run the application again to see the appearance of the “Purchase Order Application” text and “star-shaped path” before the changes were made, and then reintroduce the changes made in steps 2 and 4 above.

7. Close the splash screen form, and then close the splash screen application.
8. If time permits, experiment with the code used to set the alpha value (transparency) for the path gradient brushes. You can find the appropriate code section by double-clicking **TODO: modify transparency settings for path gradient brushes** in the Task list.

# Lesson: Working with Bitmap Images



- What Is a Bitmap Image?
- How to Load and Display an Image
- How to Crop and Scale Images
- How to Rotate, Skew, and Reflect Images
- How to Create Thumbnail Images
- Practice: Working with Bitmap Images

**Introduction**

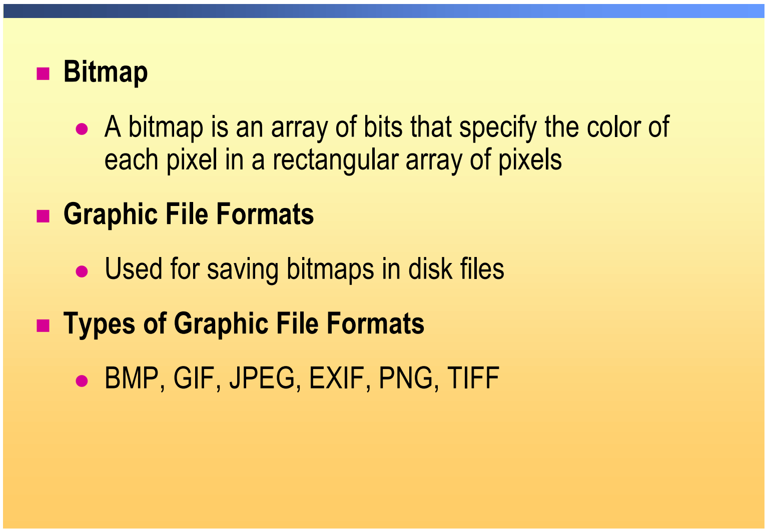
Certain kinds of pictures are difficult or impossible to display with the techniques of vector graphics. For example, the pictures on toolbar buttons and the pictures that appear as icons would be difficult to specify as collections of lines and curves. Images of this type are stored as bitmaps, arrays of numbers that represent the colors of individual dots on the screen.

**Lesson objectives**

After completing this lesson, you will be able to:

- Describe bitmaps and the various file formats for saving bitmaps.
- Load and display bitmaps on screen.
- Crop and scale images.
- Rotate, skew, and reflect images.
- Create thumbnail images.

# What Is a Bitmap Image?



- **Bitmap**
  - A bitmap is an array of bits that specify the color of each pixel in a rectangular array of pixels
- **Graphic File Formats**
  - Used for saving bitmaps in disk files
- **Types of Graphic File Formats**
  - BMP, GIF, JPEG, EXIF, PNG, TIFF

**Introduction**

A *bitmap* is an array of bits that specify the color of each pixel in a rectangular array of pixels. The number of bits devoted to an individual pixel determines the number of colors that can be assigned to that pixel. For example, if each pixel is represented by 4 bits, then a pixel can be assigned one of 16 different colors (2^4 = 16).

Disk files that store bitmaps usually contain one or more information blocks that store information such as the number of bits per pixel, number of pixels in each row, and number of rows in the array. Such a file might also contain a color table (sometimes called a color palette). A color table maps numbers in the bitmap to specific colors.

A bitmap that stores indexes in a color table is called a *palette-indexed* bitmap. Some bitmaps have no need for a color table. For example, if a bitmap uses 24 bits per pixel, that bitmap can store the colors themselves rather than indexes in a color table.

**Graphics file formats**

There are many standard formats for saving bitmaps in disk files. GDI+ supports the following graphics file formats.

- BMP

  BMP is a standard format used by Windows to store device-independent and application-independent images. BMP files are usually not compressed and therefore are not well suited for transfer across the Internet.

- Graphics Interchange Format (GIF)

  GIF is a common format for images that appear on Web pages. GIFs work well for line drawings, pictures with blocks of solid color, and pictures with sharp boundaries between colors. GIFs are compressed, but no information is lost in the compression process; a decompressed image is exactly the same as the original. GIF files can store only colors with 1, 2, 4, or 8 bits per pixel.

- Joint Photographic Experts Group (JPEG)

  JPEG is a compression scheme that is used for natural scenes such as scanned photographs. Some information is lost in the compression process, but often the loss is imperceptible to the human eye. JPEGs store 24 bits per pixel, so they are capable of displaying more than 16 million colors.

- Exchangeable Image File (EXIF)

  EXIF is a file format used for photographs captured by digital cameras. An EXIF file contains an image that is compressed according to the JPEG specification. An EXIF file also contains information about the photograph (date taken, shutter speed, exposure time, and so on) and information about the camera (manufacturer, model, and so on).

- Portable Network Graphics (PNG)

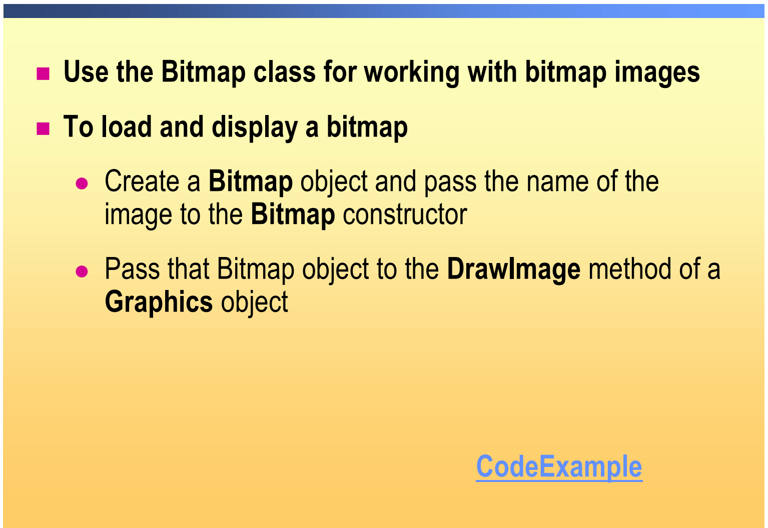
  The PNG format retains many of the advantages of the GIF format but also provides capabilities beyond those of GIF. Like GIF files, PNG files are compressed with no loss of information. PNG files can store colors with 8, 24, or 48 bits per pixel and grayscales with 1, 2, 4, 8, or 16 bits per pixel. In contrast, GIF files can use only 1, 2, 4, or 8 bits per pixel.

  PNG improves on GIF in its ability to progressively display an image; that is, to display better and better approximations of the image as it arrives over a network connection. PNG files can contain gamma correction and color correction information so that the images can be accurately rendered on a variety of display devices.

- Tagged Image File Format (TIFF)

  TIFF is a flexible and extendable format that is supported by a wide variety of platforms and image-processing applications. TIFF files can store images with an arbitrary number of bits per pixel and can use a variety of compression algorithms. Several images can be stored in a single, multiple-page TIFF file. Information related to the image, such as scanner make, host computer, type of compression, orientation, samples per pixel, and so on can be stored in the file and arranged through the use of tags.

# How to Load and Display an Image



- **Use the Bitmap class for working with bitmap images**
- **To load and display a bitmap**
  - Create a **Bitmap** object and pass the name of the image to the **Bitmap** constructor
  - Pass that Bitmap object to the **DrawImage** method of a **Graphics** object

CodeExample

**Introduction**

GDI+ provides the **Bitmap** class for working with bitmap images. The **Bitmap** class derives from the **Image** class but provides additional methods for loading saving, and manipulating bitmap images.

**Procedure**

To load a bitmap from a file and display that bitmap on the screen, you use a **Bitmap** object and a **Graphics** object. The **Bitmap** class supports several file formats, including BMP, GIF, JPEG, PNG, and TIFF.

To load and display a bitmap:

1. Create a **Bitmap** object.

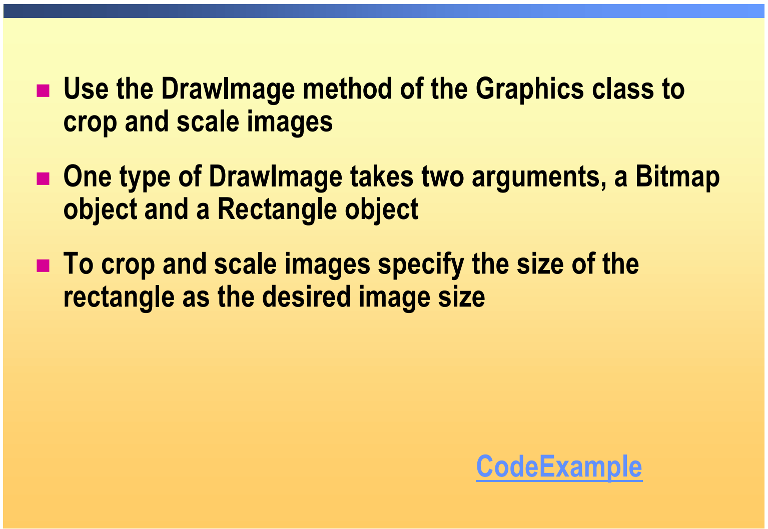
   When you create the **Bitmap** object, pass the name of the image file to the **Bitmap** constructor.

2. After you have created a **Bitmap** object, pass that **Bitmap** object to the **DrawImage** method of a **Graphics** object.

   The following example creates a **Bitmap** object from a JPEG file located in the startup folder of the application and then draws the bitmap with its upper-left corner at (60, 10).

```
string fileName = Application.StartupPath + @"\Bitmap.jpg";
Bitmap fileImage = new Bitmap(fileName);
e.Graphics.DrawImage(fileImage, 60, 10);
```

# How to Crop and Scale Images



- Use the DrawImage method of the Graphics class to crop and scale images
- One type of DrawImage takes two arguments, a Bitmap object and a Rectangle object
- To crop and scale images specify the size of the rectangle as the desired image size

CodeExample

**Introduction**

The **Graphics** class in GDI+ provides several **DrawImage** methods, some of which have source and destination rectangle parameters that you can use to crop and scale images.

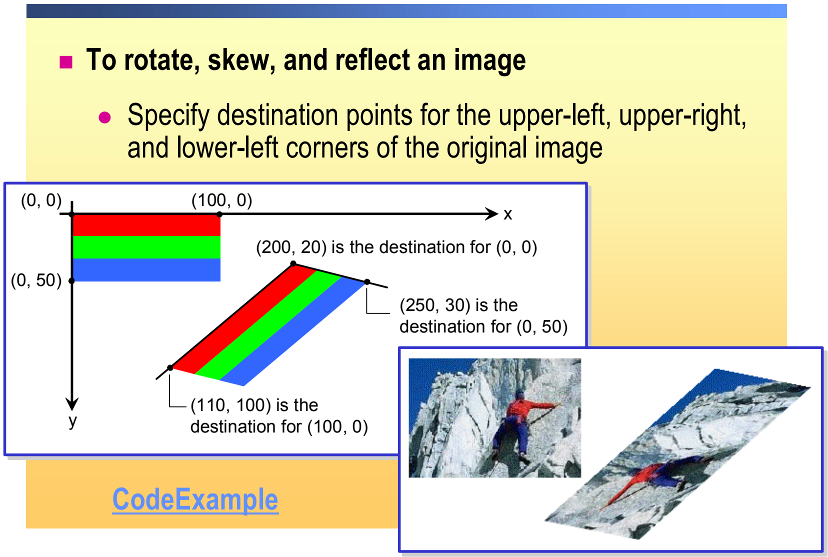
### Procedure

One of the **DrawImage** methods receives a **Bitmap** object and uses a **Rectangle** object to specify the size of the drawn image. The rectangle specifies the destination for the drawing operation; that is, it specifies the rectangle in which to draw the image. If the size of the destination rectangle is different from the size of the original image, the image is scaled to fit the destination rectangle. The following example draws an image full scale and then draws the image with scaled coordinates.

```
// draw the original image
e.Graphics.DrawImage(fileImage, 0, 0);

// create a  rectangle that can be used to create
// a scaled version of the original image
float topImage = e.PageBounds.Top;
float leftImage = e.PageBounds.Left;
float widthImage = fileImage.Size.Width;
float heightImage = fileImage.Size.Height;

Rectangle scaledCoordinates = new Rectangle(
                               leftImage,
                               topImage + heightImage,
                               0.5 * widthImage,
                               0.5 * heightImage);

// draw  a smaller version of the bitmap image by
// specifying the image and a rectangle
e.Graphics.DrawImage(fileImage, scaledCoordinates);
```

Some overloads of the **DrawImage** method include a source-rectangle parameter as well as a destination-rectangle parameter. The source-rectangle parameter specifies the portion of the original image to draw. The destination rectangle parameter specifies the rectangle in which to draw that portion of the image. If the source rectangle specifies a region of the original image that is smaller than the original image, the image drawn in the destination rectangle will be a cropped version of the original image showing only that portion of the image specified by the source rectangle. If the size of the destination rectangle is different from the size of the source rectangle, the image is scaled to fit the destination rectangle.

# How to Rotate, Skew, and Reflect Images



- **To rotate, skew, and reflect an image**
  - Specify destination points for the upper-left, upper-right, and lower-left corners of the original image

CodeExample

(0, 0)
(100, 0)
x
(0, 50)
y
(200, 20) is the destination for (0, 0)
(250, 30) is the destination for (0, 50)
(110, 100) is the destination for (100, 0)

*****************************ILLEGAL FOR NON-TRAINER USE******************************

**Introduction**

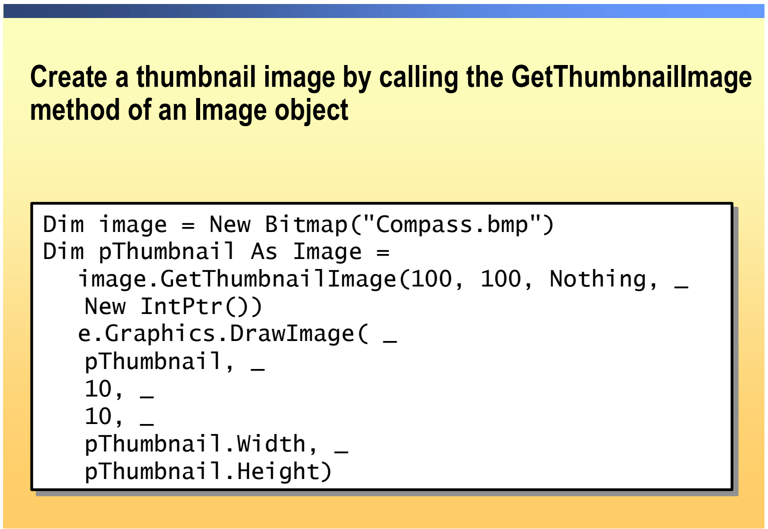
You can rotate, skew, and reflect an image by specifying destination points for the upper-left, upper-right, and lower-left corners of the original image.

### Procedure

The following code demonstrates how to create the point arrays that can be used to rotate, skew, and reflect an image.

```
// create a set of points that can be used to
//   draw a version of the original image
//   that is rotated 90 degrees counter-clockwise
//
//   upper-left becomes lower-left
//   upper-right becomes upper-left
//   lower-left becomes lower-right
Point[] rotatedCoordinates = {
new Point(leftImage, topImage + heightImage + widthImage),
new Point(leftImage, topImage + heightImage),
new Point(leftImage + heightImage,
             topImage + heightImage + widthImage)
};

e.Graphics.DrawImage(fileImage, rotatedCoordinates);

// create a set of points that can be used to
//   draw a skewed version of the original image
//
//   upper-left remains upper-left
//   upper-right remains upper-right
//   lower-left becomes lower-left + .5 width
Point[] skewedCoordinates = {
new Point(leftImage, topImage + heightImage),
new Point(leftImage + widthImage, topImage + heightImage),
new Point(leftImage + widthImage / 2,
                    topImage + heightImage * 2)
};

e.Graphics.DrawImage(fileImage, skewedCoordinates);

// create a set of points that can be used to
//   draw a reflected version of the original image
//
//   upper-left becomes lower-left + .5 width
//   upper-right becomes lower-right +.5 width
//   lower-left becomes upper-left
Point[] reflectedCoordinates = {
new Point(leftImage + widthImage / 2,
                        topImage + heightImage * 2),
new Point(leftImage + widthImage + widthImage / 2,
                        topImage + heightImage * 2),
new Point(leftImage, topImage + heightImage)
};

e.Graphics.DrawImage(fileImage, reflectedCoordinates);
```

# How to Create Thumbnail Images



Create a thumbnail image by calling the GetThumbnailImage method of an Image object

```
Dim image = New Bitmap("Compass.bmp")
Dim pThumbnail As Image =
  image.GetThumbnailImage(100, 100, Nothing, _
   New IntPtr())
  e.Graphics.DrawImage( _
   pThumbnail, _
   10, _
   10, _
   pThumbnail.Width, _
   pThumbnail.Height)
```

**Introduction**

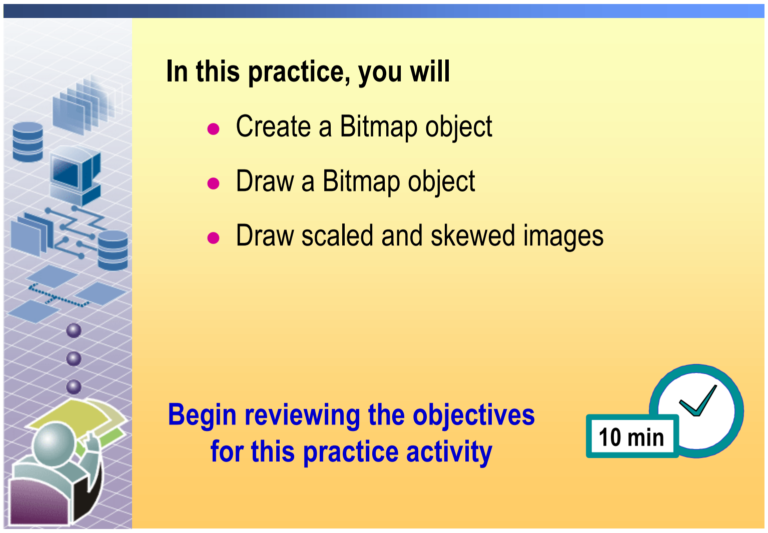
A thumbnail image is a small version of an image. You can create a thumbnail image by calling the **GetThumbnailImage** method of an **Image** object.

**Procedure**

The following code constructs an **Image** object from the file WinForms.jpg. The code creates a thumbnail image that is 1/6 the size of the original. The *callback* and *callbackData* parameters of the **GetThumbnailImage** method are not used in this example.

```
string fileName = Application.StartupPath +
@"\WinForms.jpg";
Bitmap fileImage = new Bitmap(fileName);
Image pThumbnail = fileImage.GetThumbnailImage(
                        fileImage.Width / 6,
                        fileImage.Height / 6,
                        null, new IntPtr());

e.Graphics.DrawImage(pThumbnail, 100, 100,
             pThumbnail.Width, pThumbnail.Height);
```

# Practice: Working With Bitmaps



In this practice, you will

- Create a Bitmap object
- Draw a Bitmap object
- Draw scaled and skewed images

Begin reviewing the objectives for this practice activity

10 min

**Introduction**

You are a software developer at Northwind Traders. One of your current projects requires that you create a routine for opening, modifying, and printing bitmap images.

In this practice, you will draw and preview a bitmap object on a print document, scale the image, skew the image, and examine the code used to create a semitransparent image.

**Instructions**

### ► Open the practice project file

1. Using Windows Explorer, navigate to *install_folder*\Practices\AppendixA\AppendixA_02\Starter.
2. Double-click the Bitmap Images.sln solution file to open the project.
3. Open Form1 in the Code Editor.
4. On the **View** menu, point to **Show Tasks**, and then click **Comment**.

### ► Create a Bitmap object

1. In the Task List, double-click **TODO: create bitmap object**.
2. Add the following code statement below the TODO line.

```
Bitmap fileImage = new Bitmap(fileName);
```

### ► Draw a Bitmap object

1. In the Task List, double-click **TODO: draw the bitmap image**.
2. Add the following code statement below the TODO line.

```
e.Graphics.DrawImage(fileImage, leftImage, topImage);
```

3. On the **Debug** menu, click **Start**, and then click **Print Preview**.
4. Close the **Print Preview** dialog box and the Bitmap Images application.

### ► Draw scaled and skewed images

1. In the Task List, double-click **TODO: draw a scaled image**.
2. Add the following code statement below the TODO line.

```
e.Graphics.DrawImage(fileImage, scaledCoordinates);
```

3. On the **Debug** menu, click **Start**, and then click **Print Preview**.
4. Close the **Print Preview** dialog box and the Bitmap Images application.
5. In the Task List, double-click **TODO: draw a skewed image**.
6. Add the following code statement below the TODO line.

```
e.Graphics.DrawImage(fileImage, skewedCoordinates);
```

7. On the **Debug** menu, click **Start**, and then click **Print Preview**.
8. Close the **Print Preview** dialog box.
9. Click **Draw semitransparent reflection**, and then click **Print Preview**.
10. Close the **Print Preview** dialog box and the Bitmap Images application.
11. Examine the code used to set up the scaled and skewed versions of the image and the transparency settings.