Microsoft

MCTS EXAM
70-505

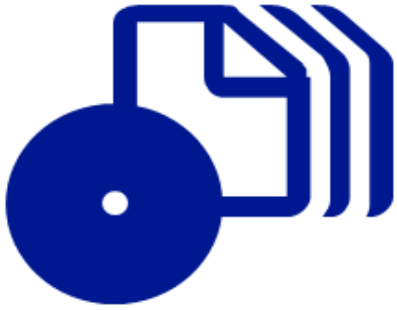
Microsoft® .NET Framework 3.5– Windows® Forms Application Development

Matthew A. Stoecker,
Steve Stein

SELF-PACED

Training Kit

How to access your CD files



The print edition of this book includes a CD. To access the CD files, go to http://aka.ms/626379/files, and look for the Downloads tab.

Note: Use a desktop web browser, as files may not be accessible from all ereader devices.

Questions? Please contact: mspinput@microsoft.com

Microsoft Press

PUBLISHED BY
Microsoft Press
A Division of Microsoft Corporation
One Microsoft Way
Redmond, Washington 98052-6399

Library of Congress Control Number: 2008940503

Printed and bound in the United States of America.

1 2 3 4 5 6 7 8 9 QWT 4 3 2 1 0 9

Distributed in Canada by H.B. Fenn and Company Ltd.

A CIP catalogue record for this book is available from the British Library.

Microsoft Press books are available through booksellers and distributors worldwide. For further information about international editions, contact your local Microsoft Corporation office or contact Microsoft Press International directly at fax (425) 936-7329. Visit our Web site at www.microsoft.com/mspress. Send comments to tkinput@microsoft.com.

Acquisitions Editor: Ken Jones
Developmental Editor: Laura Sackerman
Project Editor: Maureen Zimmerman
Editorial Production: nSight, Inc.
Technical Reviewer: Kurt Meyer; Technical Review services provided by Content Master, a member of CM Group, Ltd.
Cover: Tom Draper Design

Body Part No. X15-32081

Contents at a Glance

Contents

Introduction

This training kit is designed for developers who plan to take Microsoft Certified Technology Specialist (MCTS) exam 70-505, as well as for developers who need to know how to develop Microsoft Windows–based applications using the Microsoft .NET Framework 3.5. We assume that before you begin using this kit you have a working knowledge of Windows, Microsoft Visual Studio, and Microsoft Visual Basic or C#.

By using this training kit, you'll learn how to do the following:

- Create a user interface (UI) for a Windows Forms application by using standard controls.
- Integrate data in a Windows Forms application.
- Implement printing and reporting functionality in a Windows Forms application.
- Enhance usability.
- Implement asynchronous programming techniques to improve the user experience.
- Develop Windows Forms controls.
- Configure and deploy applications.

Hardware Requirements

The following hardware is required to complete the practice exercises:

- Computer with a 1.6 GHz or faster processor
- 384 MB of RAM or more (786 MB of RAM or more for Windows Vista)
- 2.2 GB of available hard disk space
- DVD-ROM drive
- 1024 x 768 or higher resolution display with 256 colors
- Keyboard and Microsoft mouse or compatible pointing device

Software Requirements

The following software is required to complete the practice exercises:

- One of the following operating systems:
  - Windows XP with Service Pack 2
  - Windows XP Professional x64 Edition (WOW)
  - Windows Server 2003 with Service Pack 1
  - Windows Server 2003, x64 Editions (WOW)
  - Windows Server 2003 R2
  - Windows Server 2003 R2, x64 Editions (WOW)
  - Microsoft Windows Vista (all editions except Starter Edition)
- Microsoft Visual Studio 2008. (A 90-day evaluation edition of Visual Studio 2008 Professional Edition is included on DVD with this book.)

Using the CD and DVD

A companion CD and an evaluation software DVD are included with this training kit. The companion CD contains the following:

- **Practice tests** You can reinforce your understanding of how to create .NET Framework 3.5 applications by using electronic practice tests that you customize to meet your needs from the pool of Lesson Review questions in this book. Or you can practice for the 70-505 certification exam by using tests created from a pool of 200 realistic exam questions, which is enough to give you many different practice exams to ensure that you're prepared.
- **Code** Each chapter in this book includes sample files associated with the lab exercises at the end of every lesson. For some exercises, you will be instructed to open a project prior to starting the exercise. For other exercises, you will create a project on your own. In either case, you can reference a completed project on the CD in the event you experience a problem following the exercise.
- **An eBook** An electronic version (eBook) of this book is included for times when you don't want to carry the printed book with you. The eBook is in Portable Document Format (PDF), and you can view it by using Adobe Acrobat or Adobe Reader. You can also use it to cut and paste code as you work through the exercises.

The evaluation software DVD contains a 90-day evaluation edition of Visual Studio 2008 Professional Edition, in case you want to use it with this book.

How to Install the Practice Tests

To install the practice test software from the companion CD to your hard disk, do the following:

1. Insert the companion CD into your CD drive, and accept the license agreement. A CD menu appears.

> ***NOTE*** **IF THE CD MENU DOESN'T APPEAR**
>
> **If the CD menu or the license agreement doesn't appear, AutoRun might be disabled on your computer. Refer to the Readme.txt file on the CD-ROM for alternate installation instructions.**

2. Click the Practice Tests item and follow the instructions on the screen.

How to Use the Practice Tests

To start the practice test software, follow these steps:

1. Click Start/All Programs/Microsoft Press Training Kit Exam Prep. A window appears that shows all the Microsoft Press training kit exam prep suites installed on your computer.
2. Double-click the lesson review or practice test you want to use.

> ***NOTE*** **LESSON REVIEWS VS. PRACTICE TESTS**
>
> **Select the (70-505) Microsoft .NET Framework 3.5–Windows-Based Client Development Foundation *lesson review* to use the questions from the "Lesson Review" sections of this book. Select the (70-505) Microsoft .NET Framework 3.5–Windows-Based Client Development *practice test* to use a pool of 200 questions similar to those in the 70-505 certification exam.**

Lesson Review Options

When you start a lesson review, the Custom Mode dialog box appears so that you can configure your test. You can click OK to accept the defaults, or you can customize the number of questions you want, how the practice test software works, which exam objectives you want the questions to relate to, and whether you want your lesson review to be timed. If you're retaking a test, you can select whether you want to see all the questions again or only those questions you missed or didn't answer.

After you click OK, your lesson review starts.

- To take the test, answer the questions and use the Next, Previous, and Go To buttons to move from question to question.
- After you answer an individual question, if you want to see which answers are correct—along with an explanation of each correct answer—click Explanation.
- If you'd rather wait until the end of the test to see how you did, answer all the questions and then click Score Test. You'll see a summary of the exam objectives you chose and the percentage of questions you got right overall and per objective. You can print a copy of your test, review your answers, or retake the test.

Practice Test Options

When you start a practice test, you choose whether to take the test in Certification Mode, Study Mode, or Custom Mode.

- **Certification Mode** Closely resembles the experience of taking a certification exam. The test has a set number of questions, it's timed, and you can't pause and restart the timer.
- **Study Mode** Creates an untimed test in which you can review the correct answers and the explanations after you answer each question.
- **Custom Mode** Gives you full control over the test options so that you can customize them as you like.

In all modes, the user interface you see when taking the test is basically the same but with different options enabled or disabled, depending on the mode. The main options are discussed in the previous section, "Lesson Review Options."

When you review your answer to an individual practice test question, a "References" section is provided that lists where in the training kit you can find the information that relates to that question and provides links to other sources of information. After you click Test Results to score your entire practice test, you can click the Learning Plan tab to see a list of references for every objective.

How to Uninstall the Practice Tests

To uninstall the practice test software for a training kit, use the Add Or Remove Programs option (for Windows XP) or the Programs and Features option (for Windows Vista) in Windows Control Panel.

Microsoft Certified Professional Program

The Microsoft certifications provide the best method to prove your command of current Microsoft products and technologies. The exams and corresponding certifications are developed to validate your mastery of critical competencies as you design and develop, or implement and support, solutions with Microsoft products and technologies. Computer professionals who become Microsoft-certified are recognized as experts and are sought after industry-wide. Certification brings a variety of benefits to the individual and to employers and organizations.

MORE INFO **ALL THE MICROSOFT CERTIFICATIONS**

For a full list of Microsoft certifications, go to *www.microsoft.com/learning/mcp/default.asp*.

Technical Support

Every effort has been made to ensure the accuracy of this book and the contents of the companion CD. If you have comments, questions, or ideas regarding this book or the companion CD, please send them to Microsoft Press by using either of the following methods:

E-mail:

• tkinput@microsoft.com

Postal Mail:

• *Microsoft Press*
Attn: MCTS Self-Paced Training Kit (Exam 70-505): Microsoft .NET Framework 3.5—Windows Forms Application Development, *Editor*
One Microsoft Way
Redmond, WA 98052–6399

For additional support information regarding this book and the CD-ROM (including answers to commonly asked questions about installation and use), visit the Microsoft Press Technical Support Web site at *www.microsoft.com/learning/support/books/*. To connect directly to the Microsoft Knowledge Base and enter a query, visit *http://www.microsoftpressstore.com/title/9780735626379*. For support information regarding Microsoft software, please connect to *http://support.microsoft.com*.

Evaluation Edition Software Support

The 90-day evaluation edition provided with this training kit is not the full retail product and is provided only for the purposes of training and evaluation. Microsoft and Microsoft Technical Support do not support this evaluation edition.

Information about any issues relating to the use of this evaluation edition with this training kit is posted to the Support section of the Microsoft Press Web site (*www.microsoft.com /learning/support/books/*). For information about ordering the full version of any Microsoft software, please call Microsoft Sales at (800) 426-9400 or visit *www.microsoft.com*.

CHAPTER 3

Advanced Windows Forms Controls

This chapter continues where Chapter 2, "Configuring Controls and Creating the User Interface," left off, with an in-depth examination of Windows Forms controls. In this chapter, you will learn how to create and configure controls for displaying lists, setting values and dates, displaying images, browsing the Web, and notifying the user of background processes. You will also learn how to create access keys for controls without using the *Label* control as shown in Chapter 2.

Exam objectives in this chapter:

- Add and configure a Windows Forms control.
  - Provide a list of options on a Windows Form by using a *ListBox* control, a *ComboBox* control, or a *CheckedListBox* control.
  - Configure the layout and functionality of a Windows Form to display a list of items.
  - Implement value-setting controls on a Windows Form.
  - Configure a *WebBrowser* control.
  - Add and configure date-setting controls on a Windows Form.
  - Display images by using Windows Forms controls.
  - Configure the *NotifyIcon* component.
  - Create access keys for Windows Forms controls.

Lessons in this chapter:

Before You Begin

To complete the lessons in this chapter, you must have:

- A computer that meets or exceeds the minimum hardware requirements listed in the "Introduction" at the beginning of the book.
- Microsoft Visual Studio installed on your computer.
- An understanding of Microsoft Visual Basic or C# syntax and familiarity with the .NET Framework.
- Have a good understanding of Windows Forms, how to add controls to forms, and how to use the Visual Studio Integrated Development Interface (IDE).

REAL WORLD

Matt Stoecker

When I am creating a user interface (UI), the large variety of controls that are available for use dramatically streamlines the UI creation process. Built-in controls for displaying lists and images and setting dates and other values allow me to spend less time on UI coding tasks and more time developing the application's custom functionality.

Lesson 1: Creating and Configuring List-Display Controls

A common scenario in user interface design is to present lists of data to users and to allow them to select items from that list. Visual Studio provides several list-based controls that allow a variety of presentation options. In this lesson, you will learn about the basic list-based controls, such as the *ListBox*, *ComboBox*, and *CheckedListBox*, as well as more specialized list-based controls, such as *ListView*, *TreeView*, *NumericUpDown*, and *DomainUpDown*. You will learn how to display lists and select items from lists.

After this lesson, you will be able to:

- Programmatically determine which item in a list appears in a given position.
- Add or remove items from a list of items in a list-based control.
- Bind a list-based control to a data source.
- Sort list data.
- Display data in a drop-down combo box.
- Select one or more items from a predefined list.
- Use the *ListView* control to display a list of items with icons.
- Use the *TreeView* control to display a list of items in a hierarchical view.
- Configure the *DomainUpDown* control to display a list of strings.
- Configure the *NumericUpDown* control to display a list of numbers.

Estimated lesson time: 60 minutes

Overview of List-Based Controls

The basic list-based controls are the *ListBox*, *ComboBox*, and *CheckedListBox* controls. Although differing somewhat in appearance and functionality, each of these controls organizes and presents lists of data in the same way, and each contains an *Items* collection that organizes the items contained in one of these controls.

The *Items* collection is basically a collection of objects. Although these objects are often strings, they do not have to be. If a collection does contain a string, however, the string representation of the object is displayed in the control.

ListBox Control

The *ListBox* control is the simplest of the list-based controls. It serves primarily to display a simple list of items in an easy-to-navigate user interface. Users can select one or more items. Table 3-1 describes the important properties of the *ListBox* control.

TABLE 3-1 Important Properties of the *ListBox* Control

PROPERTY	DESCRIPTION
DataSource	Sets the source for data binding in this control.
DisplayMember	Represents the data member that is displayed in this control.
FormatString	Specifies a formatting string that will be used to format the entries in the control if *FormattingEnabled* is set to *True*.
FormattingEnabled	Determines whether the entries in the control are formatted using the *FormatString*.
Items	Returns the collection of items contained in this control.
MultiColumn	Indicates whether this item shows multiple columns of items or only a single item.
SelectedIndex	Gets the index of the selected item or, if the *SelectionMode* property is set to *MultiSimple* or *MutilExtended*, returns the index to any selected item.
SelectedIndices	Returns a collection of all selected indexes.
SelectedItem	Returns the selected item or, if the *SelectionMode* property is set to *MultiSimple* or *MultiExtended*, returns the index to any selected item.
SelectedItems	Returns a collection of all selected items.
SelectedValue	In a data-bound control, returns the value associated with the selected item. If the control is not data-bound, or, if the *ValueMember* is not set, this property returns the *ToString* value of the selected item.
SelectionMode	Determines how many items can be selected in a *ListBox*. Can be set to *None*, *One*, *MultiSimple*, or *MultiExtended*. *MultiSimple* allows the selection of multiple objects, and *MultiExtended* allows the use of the Shift and Ctrl keys when making multiple selections.
ValueMember	Indicates the data member that will provide the values for the *ListBox*.

ComboBox Control

The *ComboBox* control is similar to the *ListBox* control, but, in addition to allowing the user to select items from a list, it provides a space for a user to type an entry as well as select items from a list. Additionally, you can configure the *ComboBox* either to display a list of options or to provide a drop-down list of options. Table 3-2 describes the important properties of the *ComboBox* control.

TABLE 3-2 Important Properties of the *ComboBox* Control

PROPERTY	DESCRIPTION
DataSource	Sets the source for data binding in this control.
DisplayMember	Represents the data member that is displayed in this control.
DropDownHeight	Sets the maximum height for the drop-down box.
DropDownStyle	Determines the style of the combo box. Can be set to *Simple*, which is similar to a *ListBox*; *DropDown*, which is the default; or *DropDownList*, which is similar to *DropDown* but does not allow the user to type a new value.
DropDownWidth	Sets the width of the drop-down section of the combo box.
FormatString	Specifies a formatting string that will be used to format the entries in the control if *FormattingEnabled* is set to *True*.
FormattingEnabled	Determines whether the entries in the control are formatted using the *FormatString*.
Items	Returns the collection of items contained in this control.
SelectedIndex	Gets the index of the selected item.
SelectedItem	Returns the selected item.
SelectedValue	In a data-bound control, returns the value associated with the selected item. If the control is not data-bound, or, if the *ValueMember* is not set, this property returns the *ToString* value of the selected item.
ValueMember	Indicates the data member that will provide the values for the *ComboBox*.

CheckedListBox Control

The *CheckedListBox* displays a list of items to users and allows them to select multiple items by checking boxes that are displayed next to the items. Any number of items can be checked, but only one item can be selected at a time. You can retrieve a collection that represents the checked items by accessing the *CheckedItems* collection, and you can get a collection of the checked indexes by accessing the *CheckedIndices* collection. Table 3-3 describes the important properties of the *CheckedListBox* control.

TABLE 3-3 Important Properties of the *CheckedListBox* Control

PROPERTY	DESCRIPTION
CheckedIndices	Returns a collection that represents all of the checked indexes
CheckedItems	Returns a collection that exposes all of the checked items in the control
FormatString	Specifies a formatting string that will be used to format the entries in the control if *FormattingEnabled* is set to *True*
FormattingEnabled	Determines whether the entries in the control are formatted using the *FormatString*
Items	Returns the collection of items contained in this control
MultiColumn	Indicates whether this control shows multiple columns of items or only a single item
SelectedIndex	Gets the index of the selected item, or, if the *SelectionMode* property is set to *MultiSimple* or *MultiExtended*, it can return any selected index
SelectedItem	Returns the selected item, or, if the *SelectionMode* property is set to *MultiSimple* or *MultiExtended*, it can return any selected item

You can set an item to be checked or unchecked by calling the *SetItemChecked* method, as shown below:

```
' VB
CheckedListBox.SetItemChecked(0, True)

// C#
checkedListBox.SetItemChecked(0, true);
```

Likewise, you can use the *SetItemCheckState* method to set the *CheckState* of an item:

```
' VB
CheckedListBox.SetItemCheckState(0, CheckState.Indeterminate)

// C#
checkedListBox.SetItemCheckState(0, CheckState.Indeterminate);
```

Adding Items to and Removing Items from a List-Based Control

You can add items to or remove items from a list-based control through either the Designer at design time or code at run time.

To add items to a list-based control at design time, you select the control in the Designer and then, in the Properties window, select the *Items* property. The String Collection Editor

(shown in Figure 3-1) opens. All of the items currently contained in the control are shown. Items can then be added to or removed from this list.

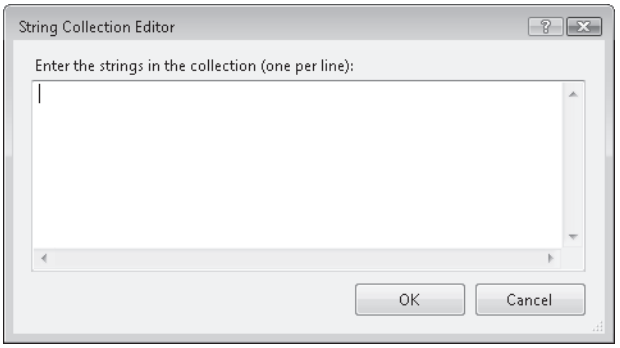


FIGURE 3-1 The String Collection Editor

You can also use code to programmatically add and remove items from the control at run time. To add an item, you use the *Items.Add* method, as shown in the following code example:

```
' VB
ListBox1.Items.Add("This string will be added to the list")

// C#
listBox1.Items.Add("This string will be added to the list");
```

If you have several items to add at once, you can use the *AddRange* method to add an array of items to the control, as shown here:

```
' VB
ListBox1.Items.AddRange(New String() {"Item1", "Item2", "Item3"})

// C#
listBox1.Items.AddRange(new String[] {"Item1", "Item2", "Item3"});
```

You can use the *Items.Insert* method to add an item to a specific index in the list. The index of items is a zero-based index, so the first item in the control is at index 0. When you add an item to an index that is already occupied by an item, that item and any items beneath it are shifted down one index. The following code shows how to insert an item to be third in the displayed list, assuming that the *ListBox1* control is already populated with several items:

```
' VB
ListBox1.Items.Insert(2, "This item will be third")

// C#
listBox1.Items.Insert(2, "This item will be third");
```

You can use the *Items.Remove* method to remove an item from the list. This method requires a reference to the object that you want to remove from the items collection. Note

that if your control contains a collection of objects that are not strings, you will need to pass a reference to the object itself to remove it, not just to the string representation that appears in the control. The following example demonstrates the *Items.Remove* method:

```
' VB
ListBox1.Items.Remove("This string will be removed")

// C#
listbox1.Items.Remove("This string will be removed");
```

If you do not know the actual item that you want to remove at run time but have the index of the item you want to remove, you can use the *Items.RemoveAt* method. This method removes the item at a given index and adjusts the indexes of the other items accordingly. The *Items.RemoveAt* method is demonstrated in the following code example:

```
' VB
' Removes the third item in the list
ListBox1.Items.RemoveAt(2)

// C#
// Removes the third item in the list
listBox1.Items.RemoveAt(2);
```

To remove all items from a list-based control, you can use the *Items.Clear* method, as shown here:

```
' VB
ListBox1.Items.Clear()

// C#
listBox1.Items.Clear();
```

Determining Where an Item Appears in a List

If you want to determine where an item appears in a list programmatically, you can do so by using the *Items.IndexOf* method. This method takes the item you want to find as an argument and returns an integer that represents the index of that item. If the item is not found in the *Items* collection, the *IndexOf* method returns -1. An example of the *IndexOf* method is shown here:

```
' VB
Dim anIndex As Integer
anIndex = ListBox1.Items.IndexOf("A String")

// C#
int anIndex;
anIndex = listBox1.Items.IndexOf("A String");
```

You can also programmatically determine the index of an item that has been selected by the user by using the *SelectedIndex* property. The *SelectedIndex* property returns the item that has been selected in the user interface at run time. If more than one item has been selected, the *SelectedIndex* property can return any of the selected items. The *SelectedIndex* property is demonstrated here:

```
' VB
Dim anIndex As Integer
anIndex = ListBox1.SelectedIndex

// C#
int anIndex;
anIndex = listBox1.SelectedIndex;
```

In controls where the *SelectionMode* property is set to *MultiSimple* or *MultiExtended*, you can return all of the selected indexes by using the *SelectedIndices* property, as shown in the following example:

```
' VB
For Each i As Integer In ListBox1.SelectedIndices
   Console.WriteLine(ListBox1.Items(i).ToString)
Next

// C#
foreach (int i in listBox1.SelectedIndices)
{
   Console.WriteLine(listBox1.Items[i].ToString());
}
```

Binding List-Based Controls to Data Sources

You will frequently want to expose data to the user in list-based controls. You can bind *ListBox* controls and *ComboBox* controls (but not *CheckedListBox* controls) to a data source by using the *DataSource*, *DisplayMember*, and *ValueMember* properties to bind a list-based control to a column of data in a data table.

Add a data source to your project. Adding data sources to your project is covered in detail in Chapter 5, "Configuring Connections and Connecting to Data."

TO BIND A LIST-BASED CONTROL TO A DATA SOURCE

1. In the Designer, select the list-based control that you want to bind to a data source.
2. In the Properties window, click the *DataSource* property to open the data source configuration interface, as shown in Figure 3-2. Set the *DataSource* property to a table contained in one of the data sources in your project.

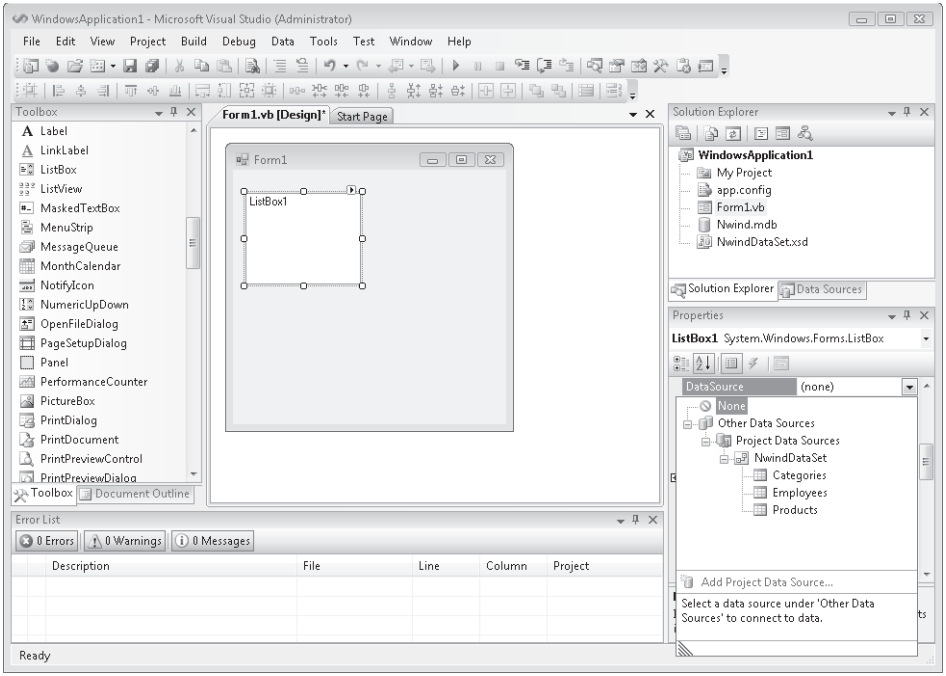


FIGURE 3-2 Setting the *DataSource* property

3. In the Properties window, click the *DisplayMember* property. Visual Studio displays the columns in the selected table. This is the column whose rows will be displayed in the control.
4. In the Properties window, click the *ValueMember* property. Choose a column name in the interface to bind the control to. This is the column whose members will provide the value that is returned from the selected index in the control.

The *DataSource* property indicates the data source (usually a data table) that the data in the control is drawn from. The *DisplayMember* property represents the column of data in the data source that is displayed to the user in the control. The *ValueMember* property allows you to designate an additional column of values to be represented in the control. For example, you might set the *DisplayMember* property to the Products column to display a list of products to the user but set the *ValueMember* to a *ProductsCode* column that returns a numeric code for each product. In this instance, whenever an item is selected, the *SelectedItem* property returns the item displayed in the *ListBox*, and the *SelectedValue* property returns the corresponding item from the *ProductsCode* column.

Sorting in List-Based Controls

You can sort the objects displayed in a list-based control by setting the *Sorted* property to *True*, as shown here:

```
' VB
ListBox1.Sorted = True

// C#
listBox1.Sorted = true;
```

Sorting data at the data source will be covered in Chapter 7, "Create, Add, Delete, and Edit Data in a Disconnected Environment."

Setting a Format for Items Displayed in a List-Based Control

You can format the items that you display in a list-based control. For example, if you are displaying a list of monetary values, you can format them all as currency and they will be displayed in the currency format that is appropriate to the culture the application is running under.

You can set a format for a list-based control by setting the *FormatString* property at design time. Selecting and clicking the *FormatString* property in the Properties window launches the Format String Dialog dialog box, shown in Figure 3-3.

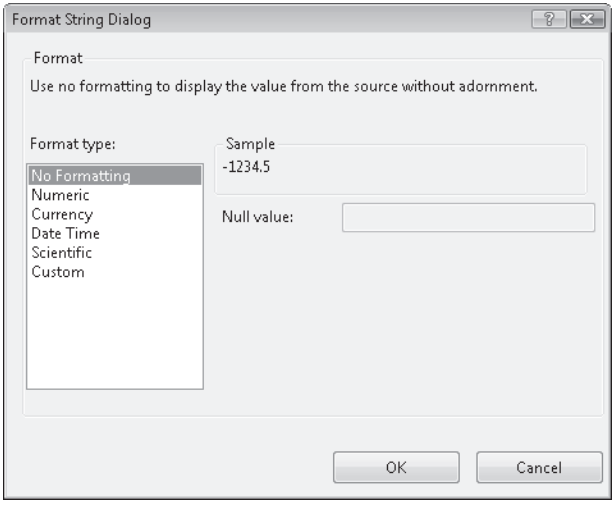


FIGURE 3-3 The Format String Dialog dialog box

The *FormattingEnabled* property determines whether to use the formatting indicated by the *FormatString*. When the *FormattingEnabled* property is set to *True*, the entries in the control are displayed in the format indicated by the *FormatString* property.

CUSTOM FORMAT STRINGS

If the preset format strings do not provide the correct format for an item, you can create a custom format string. Table 3-4 describes the characters that you can use to create a custom format string.

TABLE 3-4 Custom Format String Characters

CHARACTER	DESCRIPTION
0	Zero placeholder. If the value being formatted has a digit in the position where the '0' appears in the format string, then that digit is copied to the result string. The position of the left-most '0' before the decimal point and the right-most '0' after the decimal point determines the range of digits that are always present in the result string. Note that the "00" specifier causes the value to be rounded to the nearest digit preceding the decimal, where rounding away from zero is always used. For example, formatting 34.5 with "00" would result in the value 35.
#	Digit placeholder. If the value being formatted has a digit in the position where the '#' appears in the format string, then that digit is copied to the result string. Otherwise, nothing is stored in that position in the result string. Note that this specifier never displays the '0' character if it is not a significant digit, even if '0' is the only digit in the string. It will display the '0' character if it is a significant digit in the number being displayed. The "##" format string causes the value to be rounded to the nearest digit preceding the decimal, where rounding away from zero is always used. For example, formatting 34.5 with "##" would result in the value 35.
.	Decimal separator. The first '.' character determines the location of the first decimal separator in the formatted value. Additional '.' characters are ignored. Note that the actual character used will be the decimal separator determined by the current locale.
,	Thousands separator and scaling. First, if the format string contains a ',' character between two digit placeholders (0 or #) and to the left of the decimal point (if one is present), then the output will have thousand separators inserted between each group of three digits to the left of the decimal separator. The actual character used as the decimal separator in the result string is determined by the *NumberGroupSeparator* property of the current *NumberFormatInfo* that controls formatting. If the format string contains one or more ',' characters immediately to the left of the decimal point, then the number will be divided by the number of ',' characters multiplied by 1000 before it is formatted. For example, the format string "0," will represent 100 million as simply 100.
%	Percentage placeholder. The presence of the % symbol causes the number represented to be multiplied by 100 before formatting. The % symbol appears in the place that it occurs in the format string.

CHARACTER	DESCRIPTION
E0, E+0, E-0, e0, e+0, e-0	Scientific notation. If any of the strings "E", "E+", "E-", "e", "e+", or "e-" are present in the format string and are followed immediately by at least one '0' character, then the number is formatted using scientific notation with an 'E' or 'e' inserted between the number and the exponent. The number of '0' characters following the scientific notation indicator determines the minimum number of digits to output for the exponent. The "E+" and "e+" formats indicate that a sign character (plus or minus) should always precede the exponent. The "E", "E-", "e", or "e-" formats indicate that a sign character should precede only negative exponents.
\	Escape character. In C# this character is used to indicate that the character immediately following the '\' is to be interpreted as an escape sequence. In Visual Basic this character has no effect.
"ABC", 'ABC'	Literal strings. Characters enclosed in " or '' are displayed as literal strings in the formatted string.
;	Section separator. The ';' character is used to separate sections for positive, negative, and zero numbers in the format string. You can provide up to three sections in a format string, each containing its own format. These sections should be separated by ';' characters and will be applied to positive, negative, and zero numbers, respectively.
Other Characters	Other characters in the format string are represented as literal strings.

Selecting Items in a List-Based Control

You can programmatically select items in a list-based control by using the *SelectedItem* or *SelectedIndex* property. You can select an item in a list-based control as shown in the following example:

```
' VB
ListBox1.SelectedItem = "This item will be selected"

// C#
listBox1.SelectedItem = "This item will be selected";
```

If the *SelectedItem* property is set to an item that is not contained in the control, there is no effect, and no item is selected.

If the control allows multiple selections, you can select multiple items by setting the *SelectedItem* property multiple times if the *SelectionMode* property is set to *MultiSimple* or *MultiExtended* (which is supported only by the *ListBox* control). Once selected, an item remains selected until unselected by the user. An example is shown here:

```
' VB
ListBox1.SelectedItem = "This item will be selected"
```

```
ListBox1.SelectedItem = "This item will be selected too"

// C#
listBox1.SelectedItem = "This item will be selected";
listBox1.SelectedItem = "This item will be selected too";
```

The *SelectedIndex* property functions in a way similar to the *SelectedItem* property, except that it is an Integer type that corresponds to the sequential item in the list. You can select an item in the control by setting the *SelectedIndex* property to the corresponding index, and you can select multiple items by setting the property multiple times in succession. The main difference between the behavior of the *SelectedItem* property and the *SelectedIndex* property is that the *SelectedIndex* property throws an exception if an attempt is made to set it to a nonexistent index.

The *ListView* Control

The *ListView* control allows you to view lists of items with optional associated icons in the manner of Windows Explorer. Using the *ListView* control, you can display items with large associated icons, small associated icons, or additional details about the item. Table 3-5 shows important properties of the *ListView* control.

TABLE 3-5 Important Properties of the *ListView* Control

PROPERTY	DESCRIPTION
Columns	Contains the collection of columns to be displayed when the *View* property is set to *Details*
Groups	Contains an optional collection of groups that can be used to categorize the items contained in the *Items* collection
Items	A collection of *ListViewItems* that is displayed in the *ListView* control
LargeImageList	The *ImageList* component from which images for *ListViewItems* are drawn when the *View* property is set to *LargeIcon*
ShowGroups	Determines whether the groups contained in the *Groups* collection are shown
SmallImageList	The *ImageList* component from which images for *ListViewItems* are drawn when the *View* property is set to *SmallIcon*
View	Indicates the manner in which *ListView* items are displayed

The most important property in the *ListView* control is the *Items* property. This property contains a collection of *ListViewItem* objects. Unlike the list-based controls examined earlier, *ListViewItems* are specific objects that contain additional information about the item being displayed, such as icons that are shown in the control. Table 3-6 shows important properties of the *ListViewItem* class.

TABLE 3-6 Important Properties of the *ListViewItem* Class

PROPERTY	DESCRIPTION
Group	The group, if any, in the *ListView* control's *Groups* collection that this *ListViewItem* belongs to.
ImageIndex	The index, if any, of the image to be used for this item when the *View* property is set to *LargeIcon* or *SmallIcon*. If the *ImageIndex* property is set, the *ImageKey* property is set to "."
ImageKey	The key of the image, if any, to be used for this item when the *View* property is set to *LargeIcon* or *SmallIcon*. If the *ImageKey* property is set, the *ImageIndex* property is set to -1.
SubItems	Contains the subitems that will be shown when the *View* property is set to *Details*. These items should correspond to the columns in the *ListView* control's *Columns* collection.
Text	The text that is shown in the *ListView* property.

You can add *ListViewItems* to the *ListView* and edit the properties of individual *ListViewItems* by clicking the *Items* property of the *ListView* control to open the ListViewItem Collection Editor, shown in Figure 3-4.

The *ListView* control organizes the images associated with the *ListViewItems* in *ImageList* objects that are exposed in the *SmallImageList* and *LargeImageList* properties. The *ImageList* class will be discussed in greater detail in Lesson 2, "Creating and Configuring Value-Setting, Date-Setting, and Image-Display Controls" of this chapter. You can set the images associated with a particular *ListViewItem* by setting either the *ImageIndex* or *ImageKey* property of each *ListViewItem*. The *View* property determines if the *ListView* items are shown with large images, with small images, or in a view that exposes the subitems of the *ListViewItems*.

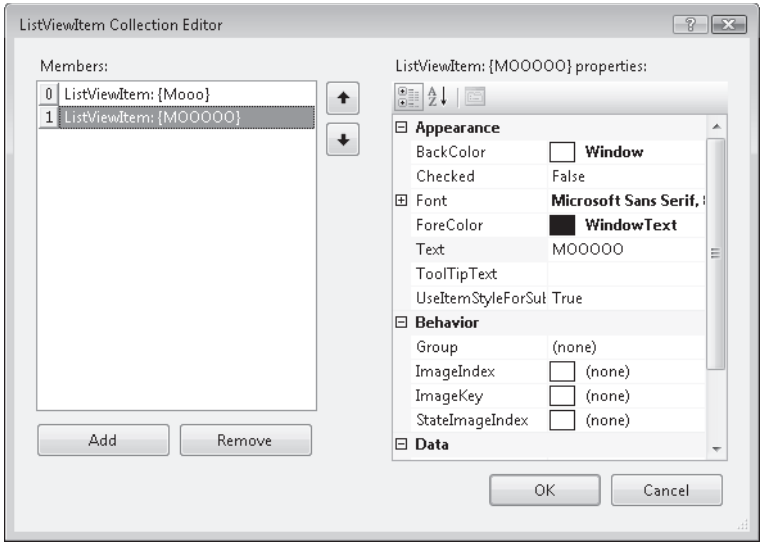


FIGURE 3-4 The ListViewItem Collection Editor

TO DISPLAY A LIST OF ITEMS WITH ICONS IN A *LISTVIEW* CONTROL

1. In the Designer, drag an *ImageList* control from the Toolbox to a design surface that already contains a *ListView* control.
2. In the Properties window, click the *Images* property of the *ImageList* to add images to the *Images* collection.
3. In the Designer, select the *ListView* control. In the Properties window, set the *SmallImageList*, *LargeImageList*, or both to the *ImageList* object.
4. In the Properties window, click Items to add *ListViewItems* to the *ListView*. In the ListViewItem Collection Editor, set either the *ImageIndex* or the *ImageKey* property for each *ListViewItem* to the appropriate image in the *ImageList*. Also, set any other properties, such as *Text*, at this point.
5. In the Designer, select the *ListView* control. In the Properties window, set the *View* property to either *LargeIcon* or *SmallIcon*.

TreeView Control

The *TreeView* control allows you to display a list of objects in a hierarchical manner. Each object in the *TreeView* control represents an instance of the *TreeNode* class, which contains information about the location of the node within the *TreeView* control. Nodes containing child nodes in the *TreeView* control can be collapsed and expanded. Figure 3-5 shows a *TreeView* control in a form.

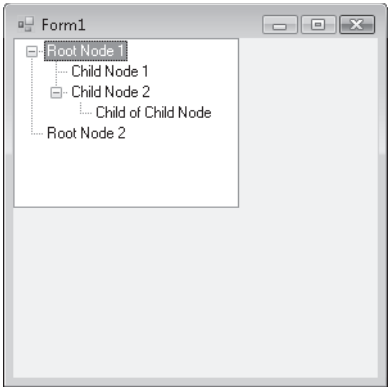


FIGURE 3-5 The *TreeView* control

The primary property of the *TreeView* control is the *Nodes* property. This property contains the collection of *TreeNodes* that comprise the root objects in the *TreeView*. Each individual *TreeNode* object contains its own collection of *TreeNodes* that represent child nodes of that node. Table 3-7 describes some of the important properties of the *TreeNode* class.

TABLE 3-7 Important Properties of the *TreeNode* Class

PROPERTY	DESCRIPTION
FirstNode	Returns the first node in the current group of child nodes.
LastNode	Returns the last node in the current group of child nodes.
NextNode	Returns the next sibling tree node.
NextVisibleNode	Returns the next visible node.
Nodes	Returns the collection of child nodes belonging to this node.
Parent	Returns the parent node of the current node. If the current node is a root node in the *TreeView*, accessing this property will return null.
PrevNode	Returns the previous sibling tree node.
PrevVisibleNode	Returns the previous visible node.
TreeView	Returns a reference to the *TreeView* control that the *TreeNode* is contained in.

Adding and Removing Nodes from the *TreeView* Controls

At design time you can add nodes to a *TreeView* control by clicking the *Nodes* property in the Properties window to display the TreeNode Editor (shown in Figure 3-6). You can add new root nodes or new child nodes and set the properties of each *TreeNode*.

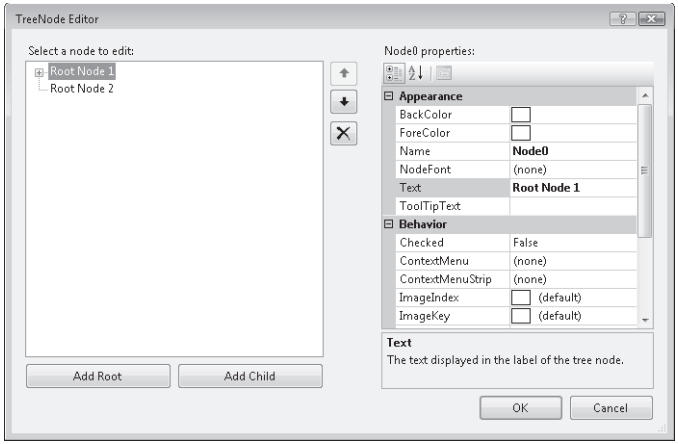


FIGURE 3-6 The TreeNode Editor

At run time, you can create new *TreeNode* objects and add them as root nodes to the *TreeView* control or add them as child nodes to another *TreeNode*. For both of these procedures, you use the *Nodes.Add* method, as shown here:

```
' VB
Dim aNode As New TreeNode("New Node")
' Add a child node to the new node
aNode.Nodes.Add(New TreeNode("New Child"))
' Adds aNode and its child As a new root node in a TreeView control named TreeView1
TreeView1.Nodes.Add(aNode)
' Adds a second child node to the first node in TreeView1
TreeView1.Nodes(0).Nodes.Add(New TreeNode("Second Child"))

// C#
TreeNode aNode = new TreeNode("New Node");
// Add a child node to the new node
aNode.Nodes.Add(new TreeNode("New Child"));
// Adds aNode and its child as a new root node in a TreeView control named TreeView1
treeView1.Nodes.Add(aNode);
// Adds a second child node to the first node in TreeView1
treeView1.Nodes[0].Nodes.Add(new TreeNode("Second Child"));
```

You can remove nodes from the *Nodes* collection by using the *Remove* and *RemoveAt* methods. The *Remove* method takes a reference to a particular node as a parameter and removes it from the collection if it exists in the collection. If the specified node does not exist in the collection, this method call is ignored. The *RemoveAt* method removes the node at a specified index. If the specified index is not present in the nodes collection, an *ArgumentOutOfRange* exception is thrown. The following example demonstrates the *Remove* and *RemoveAt* methods:

```
' VB
' Removes the node named aNode from the collection
TreeView1.Nodes.Remove(aNode)
' Removes the node at index 3 from the collection.
TreeView1.Nodes.RemoveAt(3)

// C#
// Removes the node named aNode from the collection
treeView1.Nodes.Remove(aNode);
// Removes the node at index 3 from the collection.
treeView1.Nodes.RemoveAt(3);
```

Expanding and Collapsing Nodes

The *TreeView* control presents a hierarchical view of nodes that can be expanded or collapsed to reveal or hide the child nodes as appropriate. You can expand or collapse child nodes programmatically at run time by using the *Expand* and *Collapse* methods, as shown in the following example:

```
' VB
' Expands the child nodes of a TreeNode named aNode
aNode.Expand()
' Collapses the child nodes of a TreeNode named aNode
aNode.Collapse()

// C#
// Expands the child nodes of a TreeNode named aNode
aNode.Expand();
// Collapses the child nodes of a TreeNode named aNode
aNode.Collapse();
```

NumericUpDown Control

The *NumericUpDown* control allows you to set a range of numbers that a user can browse and select. A range of numbers is presented in the control, and the user can click the up and down arrows to increase or decrease the number. Table 3-8 shows important properties of the *NumericUpDown* control.

TABLE 3-8 Important Properties of the *NumericUpDown* Control

PROPERTY	DESCRIPTION
Hexadecimal	Indicates whether the numeric value will be shown in hexadecimal
Increment	Gets or sets the amount to increment or decrement with each button click
Maximum	Indicates the maximum value for the control

PROPERTY	DESCRIPTION
Minimum	Indicates the minimum value for the control
ThousandsSeparator	Indicates whether the culture-appropriate thousands separator will be used when displaying values greater than 1000
Value	Gets or sets the current value of the control

TO CONFIGURE THE *NUMERICUPDOWN* CONTROL

1. Set the *Minimum* property to the minimum numeric value for the control.
2. Set the *Maximum* property to the maximum numeric value for the control.
3. Set the *Increment* property to the amount you want to increment and decrement with each arrow button click.
4. If desired, set the *Value* property to a default value.

DomainUpDown Control

The *DomainUpDown* control is similar to the *NumericUpDown* control in that it allows users to browse a specified series of data and set a value for the control. Instead of browsing numeric values, however, the *DomainUpDown* control allows the user to browse a collection of preset strings. Table 3-9 describes the important properties of the *DomainUpDown* control.

TABLE 3-9 Important Properties of the *DomainUpDown* Control

PROPERTY	DESCRIPTION
Items	Contains the collection of strings that are displayed in the *DomainUpDown* control
ReadOnly	Indicates whether the user can alter the *Text* of the control
Text	Gets or sets the text of the control

The *Items* collection contains the strings that are displayed in the *DomainUpDown* control. You can add strings by clicking the *Items* property in the Properties window to display the String Collection Editor. When *ReadOnly* is set to *False*, the user can choose to type a string in the *DomainUpDown* control instead of choosing one of the strings. Note that strings typed by the user are not added to the *Items* collection. Also note that the *Text* property defines the default value for the control, not the *Items* collection.

Quick Check

1. What is the purpose of the *TreeView* control?
2. What is the purpose of a *ListView* control and when would you use one?

Quick Check Answers

1. The *TreeView* control allows you to display a list of data in a hierarchically related manner.
2. The *ListView* control provides a highly configurable control that allows you to display lists of data in a variety of ways. You can use a *ListView* control when you want to provide different options to the user for the display of list data, such as providing icons or details about the data.

LAB The Adventure Works Ski Instructor Reservation Form

Over the next two labs, you will use what you have learned in this lesson and the next to add functionality to a simple application designed to reserve ski instructors. In this lab, you will add a *ComboBox* that allows the user to select the mountain they want to ski on, a *ListView* control to select a ski instructor, and a *NumericUpDown* control to select the length of the lesson.

EXERCISE 1 The Ski Instructor Reservation Form

1. In Visual Studio, open the partial solution for Lesson 1, "Creating and Configuring List-Display Controls." This solution can be installed from the companion CD.
2. In Form1, beneath the Name *TextBox*, add a *Label* control and a *ComboBox* control. Set the *Text* of the *Label* control to **Choose Ski Run**.
3. Set the *DropDownStyle* property of the *ComboBox* to *DropDownList*.
4. Add the following items to the *ComboBox Items* property: **Camelback**, **Powder Point**, and **The Plunge**.
5. Add a *Label* control and a *NumericUpDown* control to the form. Set the *Text* property of the *Label* to **Lesson Length**.
6. Set the *Minimum* property of the *NumericUpDown* control to **1** and the *Maximum* property to **3**.
7. Add a *Label* control and a *ListView* control to the form. Set the *Text* property of the *Label* control to **Choose Instructor**.
8. In the Properties window, set the *View* property of the *ListView* control to *SmallIcon*. In Lesson 2, "Creating and Configuring Value-Setting, Date-Setting, and Image-Display Controls," you will associate the items in this list with images.

9. In the Properties window, click the *Items* property to add four *ListViewItems* to the *ListView*. In the ListViewItem Collection Editor, set their *Text* properties to **Sandy**, **Jack**, **Libby**, and **Christa**.
10. Add a *Button* control to the form and set the *Text* property to **Make Reservation**.
11. In the Designer, double-click the button and add the following code to the *Button_Click* event handler:

```
' VB
If  ListView1.SelectedItems.Count > 0 Then
   MsgBox("Your reservation with " &  listView1.SelectedItems(0).Text & _
      " is confirmed.")
End If

// C#
if (listView1.SelectedItems.Count > 0)
{
   MessageBox.Show("Your reservation with " + listView1.SelectedItems[0].Text +
      " is confirmed.");
}
```

12. Press F5 to test your application.

Lesson Summary

- List-based controls are used to organize and present lists of information to the user. The *ListBox*, *ComboBox*, and *CheckedListBox* controls organize items through the *Items* property and share many common methods and properties.
- The *ListBox* control allows you to display a selection of items to the user and enables the user to select one or more items from that list.
- The *ComboBox* control can appear similar to a *ListBox* control or as a drop-down list. You can require the user to select from a list or choose to allow them to type an entry that is not present in the list.
- The *CheckedListBox* control allows you to display a list of items with a check box beside each one, enabling the user to check as many items as desired. Although multiple items can be checked, only one item can be selected in the *CheckedListBox* at any time.
- The *ListView* control allows specialized displays of lists of items. Items can be displayed in association with icons that are provided by an *ImageList* component or with additional columns of subitems.
- The *TreeView* control allows you to display lists of items in a hierarchical format. Each node contains a collection of child nodes, which can themselves have child nodes. Nodes can be expanded or collapsed.

- The *NumericUpDown* control allows the user to click up or down arrows to select a numeric value. The *DomainUpDown* control allows the user to click up or down arrows to select from a preselected set of options.

Lesson Review

You can use the following questions to test your knowledge of the information in this lesson. The questions are also available on the companion CD if you prefer to review them in electronic form.

> ***NOTE*** **ANSWERS**
>
> **Answers to these questions and explanations of why each answer choice is correct or incorrect are located in the "Answers" section at the end of the book.**

1. Which of the following properties and methods can be used to find the index of a selected item in a *ListBox* control? (Choose all that apply.)
   - **A.** *ListBox.IndexOf*
   - **B.** *ListBox.SelectedIndex*
   - **C.** *ListBox.SelectedIndices*
   - **D.** *ListBox.Select*
2. Which of the following methods cannot be used to add an item to the *Items* collection of a *ComboBox*, *ListBox*, or *CheckedListBox* control?
   - **A.** *Items.Add*
   - **B.** *Items.Insert*
   - **C.** *Items.AddRange*
   - **D.** *Items.Contains*
3. Which of the following is NOT a valid setting for the *View* property of the *ListView* control?
   - **A.** *LargeIcon*
   - **B.** *Details*
   - **C.** *Tree*
   - **D.** *SmallIcon*

Lesson 2: Creating and Configuring Value-Setting, Date-Setting, and Image-Display Controls

Allowing users to select or choose from a set of options, to set dates, and to work with images are common scenarios for user interface design. In this lesson, you will learn to use value-setting controls, such as *CheckBox*, *RadioButton*, and *TrackBar*, and date-setting controls, such as *DateTimePicker* and *MonthCalendar*. You will also learn to work with images using the *ImageList* component and the *PictureBox* control.

After this lesson, you will be able to:

- Set two or more mutually exclusive options in the user interface using a *RadioButton*.
- Use the *CheckBox* control to indicate whether a condition is on or off.
- Allow navigation through a large amount of information or visually adjust a numeric setting using a *TrackBar*.
- Allow the user to select a single item from a list of dates or times using the *DateTimePicker* control.
- Present an intuitive graphical interface for users to view and set date information using the *MonthCalendar*.
- Add images to or remove images from the *ImageList* component.
- Display graphics using the *PictureBox* control.

Estimated lesson time: 45 minutes

Value-Setting Controls

Value-setting controls allow the user to set values or pick options from a preset list in the user interface. The *CheckBox* control allows a user to select or clear particular options in a non-exclusive manner, while the *RadioButton* allows you to present a range of options to the user, only one of which can be selected. The *TrackBar* control allows the user to rapidly set a value in a graphical interface.

The *CheckBox* Control

The *CheckBox* control is a very familiar control to users. It allows the user to mark a check box next to a label to indicate acceptance or rejection of the option presented. *CheckBox* controls function in a nonexclusive manner—you can have multiple *CheckBox* controls on a single form, and any combination of them can be checked or cleared at a single time. Table 3-10 shows important properties of the *CheckBox* control.

TABLE 3-10 Important Properties of the *CheckBox* Control

PROPERTY	DESCRIPTION
AutoCheck	Determines whether the *CheckBox* is automatically checked when the text is clicked.
Checked	Gets or sets whether the *CheckBox* is checked.
CheckState	Returns the *CheckState* of the control. Possible values for this property are *Checked*, *Unchecked*, and *Indeterminate*.
Text	The text displayed next to the check box.
ThreeState	Determines whether the *CheckBox* control allows two check states or three.

The most common use for the *CheckBox* control is to allow the user to make a binary decision about an option by either checking the box or not checking it. Typically, the check box is used for nonexclusive options—that is, checking a particular check box usually does not affect the state of other text boxes. Figure 3-7 shows an example of a hypothetical pizza order form. Radio buttons are used to choose between the exclusive options Pizza or Calzone, and *CheckBox* controls are used to select toppings for the pizza or calzone that is selected.

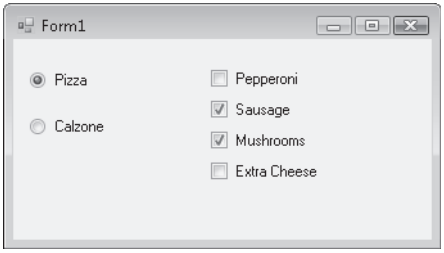


FIGURE 3-7 Example of *CheckBox* and *RadioButton* controls

You can programmatically determine if a *CheckBox* control is checked by accessing the *Checked* property. This property returns *True* if the control is checked and *False* if the control is cleared or indeterminate.

A less common use for the *CheckBox* is to allow the user to choose among three settings: *Checked*, *Unchecked*, or *Indeterminate*. This can be useful to indicate to the user that a conscious decision must be made for each option rather than simply setting a default option. You enable three-state *CheckBox* controls by setting the *ThreeState* property to *True*. This allows the user to cycle through the three states, rather than just the two, for the check box. You can determine the state of the check box by accessing the *CheckState* property.

Note that you can set the *CheckState* property to *Indeterminate* at design time even if you set the *ThreeState* property to *False*. This causes the *CheckBox* controls to start in the

indeterminate state, but once the user makes a selection, the *CheckBox* must be either checked or cleared. In this case the user is not allowed to reset the check box to indeterminate.

The *RadioButton* Control

The *RadioButton* control is used to present exclusive options to the user. The hypothetical pizza order form in Figure 3-7 demonstrates the use of *RadioButton* controls, allowing the user to choose either a pizza or a calzone, but not both. Table 3-11 shows important properties of the *RadioButton* control.

TABLE 3-11 Important Properties of the *RadioButton* Control

PROPERTY	DESCRIPTION
CHECKED	Indicates whether the *RadioButton* is selected
Text	The text displayed next to the radio button

You can determine if a particular *RadioButton* is selected by accessing the *Checked* property, which returns *True* if selected.

All *RadioButton* controls in a given container control are exclusive of one another. That means that if one *RadioButton* control is selected, the others will all be cleared. This has the net effect of allowing the user to choose only one of a group of options.

If you want to have several exclusive groups of *RadioButton* controls, the most common method is to group them in a *GroupBox* control. Each group of *RadioButton* controls in a particular *GroupBox* will be exclusive of one another but unaffected by other *RadioButton* controls in other *GroupBox* containers. An example of *RadioButton* controls in *GroupBox* containers is shown in Figure 3-8.

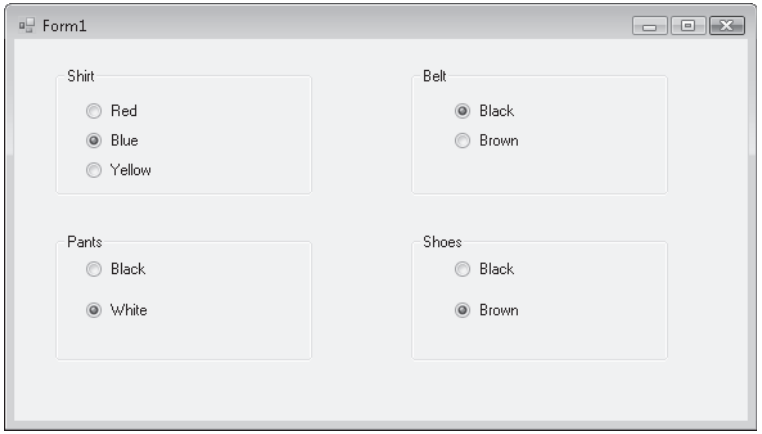


FIGURE 3-8 Example of grouped *RadioButton* controls

The *TrackBar* Control

The *TrackBar* control provides a simple interface that allows the user to set a value from a predetermined range of values by graphically manipulating a slider with the mouse or keyboard commands. This allows the user to rapidly set a value from a potentially very large range. Table 3-12 shows important properties of the *TrackBar* control.

TABLE 3-12 Important Properties of the *TrackBar* Control

PROPERTY	DESCRIPTION
LargeChange	The number of positions the slider moves in response to mouse clicks or the Page Up and Page Down keys
Maximum	The maximum value for the *TrackBar*
Minimum	The minimum value for the *TrackBar*
SmallChange	The number of positions the slider moves in response to arrow key keystrokes
TickFrequency	The number of positions between tick marks on the *TrackBar*
TickStyle	Indicates where ticks appear on the *TrackBar*
Value	The value returned by the *TrackBar*

The *Trackbar* control is shown in Figure 3.9.

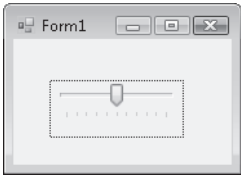


FIGURE 3-9 The *TrackBar* control

The *TrackBar* control can return an integer value in any range between the values of the *Minimum* and *Maximum* properties. The user can set this value by manipulating the graphical slider on the track bar. Clicking the control or using the Page Up and Page Down keys while the control is selected causes the value to change by the increment set in the *LargeChange* property. Using the arrow keys while the control is selected causes the value to change by the increment set in the *SmallChange* property. The user can also grab the slider with the mouse and adjust it to whatever value is needed. The *Value* property indicates the current value of the track bar.

Choosing Dates and Times

User interfaces frequently require that the user be allowed to set a date or time. For example, an application that allowed a user to make a reservation would require that a date for the reservation be entered. Visual Studio provides two controls that enable date and time choosing: the *DateTimePicker* and the *MonthCalendar*.

DateTimePicker Control

The *DateTimePicker* control allows the user to set a date, a time, or both, in an easy-to-understand graphical interface. The interface is similar to a *ComboBox* control. The user can click the drop-down box to display a calendar interface that allows the user to choose a day from a calendar or type a time into the text area in the *DateTimePicker*. The chosen day or time is then displayed in the text area of the *DateTimePicker*, and the *Value* property is set to the chosen *DateTime*. Table 3-13 shows important properties of the *DateTimePicker* control.

TABLE 3-13 Important Properties of the *DateTimePicker* Control

PROPERTY	DESCRIPTION
CustomFormat	The custom DateTime format to be used when the *Format* property is set to *Custom*.
Format	Sets the format for the *DateTime* format that is displayed in the *DateTimePicker*. Can be set to *Long*, which displays the value in long date format; *Short*, which displays the value in short date format; *Time*, which displays the time only; or *Custom*, which uses the custom *DateTime* format indicated by the *CustomFormat* property.
MaxDate	The maximum *DateTime* value the *DateTimePicker* will accept.
MinDate	The minimum *DateTime* value the *DateTimePicker* will accept.
Value	The *DateTime* value that the *DateTimePicker* is currently set to.

When the *Format* property is set to *Long* or *Short*, only the date is displayed and the date can be set only through the graphical interface. When the *Format* property is set to *Time*, the user can type a new time value into the text area of the *DateTimePicker*. The user can still choose a day through the drop-down interface. Although this day is reflected in the *Value* property, it is not displayed when the *Format* property is set to *Time*.

MonthCalendar Control

The *MonthCalendar* control is a highly configurable control that allows the user to select a range of dates in a highly intuitive interface. Table 3-14 shows the important properties of the *MonthCalendar* control.

TABLE 3-14 Important Properties of the *MonthCalendar* Control

PROPERTY	DESCRIPTION
AnnuallyBoldedDates	Contains an array of dates and times that will appear bold every year
BoldedDates	Contains an array of dates and times that will appear bold
FirstDayOfWeek	Determines which day of the week is set as the first day of the week in the *MonthCalendar* control
MaxDate	Sets the maximum date that can be chosen in the *MonthCalendar*
MinDate	Sets the minimum date that can be chosen in the *MonthCalendar*
MaxSelectionCount	Sets the maximum number of days that can be selected in the *MonthCalendar*
MonthlyBoldedDates	Contains an array of dates and times that will appear bold every month in the *MonthCalendar*
SelectionEnd	Indicates the ending date and time of the *SelectionRange* property
SelectionRange	Contains the range of dates selected by the user
SelectionStart	Indicates the starting date and time of the *SelectionRange* property

The user can select a single date by clicking a date in the *MonthCalendar* or a continuous range of dates by holding down the *Shift* key while clicking the starting date and the ending date. The range of dates selected cannot be a greater number of days than the *MaxSelectionCount* property indicates.

At run time you can retrieve the selected dates by accessing the *SelectionStart* and *SelectionEnd* properties, which expose the *Start* and *End* properties of the *SelectionRange* property. The following example demonstrates how to access the *SelectionStart* and *SelectionEnd* properties:

```
' VB
MsgBox("Your vacation starts on " & _
   MonthCalendar1.SelectionStart.ToLongDateString & _
   " and ends on " & MonthCalendar1.SelectionEnd.ToLongDateString)

// C#
MessageBox.Show("Your vacation starts on " +
   monthCalendar1.SelectionStart.ToLongDateString() + " and ends on " +
    monthCalendar1.SelectionEnd.ToLongDateString());
```

Working with Images

Images allow you to liven up your user interface as well as provide important information to the user. Visual Studio contains several components and controls that facilitate the display of images. The *PictureBox* control is an all-around control that displays pictures in different formats. The *ImageList* manages and organizes a collection of images and can be used to display images in *ListView* or to organize images for other controls.

PictureBox Control

The *PictureBox* control is the basic control used for displaying images in the user interface. The *PictureBox* control can display pictures in a variety of formats, including .bmp, .jpg, .gif, metafiles, and icons. You can display images that are present in application resource files or compiled into the application, or you can load images from a Web or disk address. Table 3-15 describes important properties of the *PictureBox* control.

TABLE 3-15 Important Properties of the *PictureBox* Control

PROPERTY	DESCRIPTION
ErrorImage	The image that will be displayed if the selected image fails to load
Image	The image to be loaded in the *PictureBox*
ImageLocation	A Web or disk address to load the image from
InitialImage	The image to be displayed in the *PictureBox* while the image is loading
SizeMode	Determines how the control handles image placement and sizing

You can set the *Image* property at design time by clicking it in the Properties window, which opens the Select Resource dialog box, shown in Figure 3-10.

You can select an image resource that is already present in a project resource file by selecting the Project Resource File radio button and selecting the .resx file that contains the image. Or you can import a new image into a resource file by clicking the Import button and navigating to the image you want to import. The selected image is added to the selected .resx file. You can also import the image as a local resource by selecting the Local Resource radio button and clicking the Import button to browse to the image you want to import. Importing an image as a local resource makes it available only to the *PictureBox* control and unavailable to the rest of the application.

Instead of loading an image from a resource, you can specify a URL from which to load an image by setting the *ImageLocation* property. When the *ImageLocation* property is set, the image is loaded from the specified address and the *Image* property is set to that image.

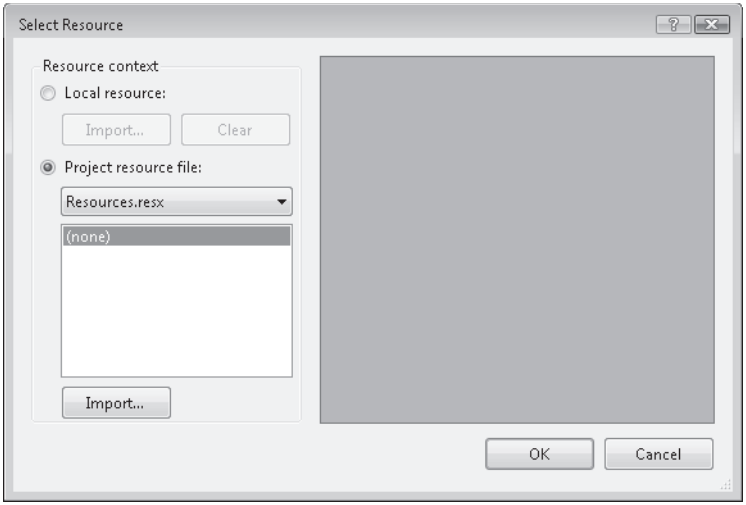


FIGURE 3-10 The Select Resource dialog box

At run time you can set the *Image* property to an instance of an image, as shown in the following example:

```
' VB
Dim anImage As New System.Drawing.Bitmap("C:\anImage.bmp")
PictureBox1.Image = anImage

// C#
System.Drawing.Bitmap anImage = new
   System.Drawing.Bitmap(@"C:\anImage.bmp");
pictureBox1.Image = anImage;
```

ImageList Component

The *ImageList* component is not a control as such, but it is a component that allows you to organize groups of images. Although it has no visual representation itself, it can supply images to other controls, such as a *ListView*, or serve as a repository for images to be loaded into a picture box. You can set the size and color depth for the images and iterate through them as you would a collection. Table 3-16 shows important properties of the *ImageList* component.

TABLE 3-16 Important Properties of the *ImageList* Component

PROPERTY	DESCRIPTION
ColorDepth	Sets the color depth for the images contained in the *ImageList* component
Images	The collection of images organized by the *ImageList* component
ImageSize	Sets the size for the images contained in the *ImageList* control

You can add new items to the *ImageList* control by clicking the *Images* property in the Properties window. This opens the Images Collection Editor, shown in Figure 3-11.

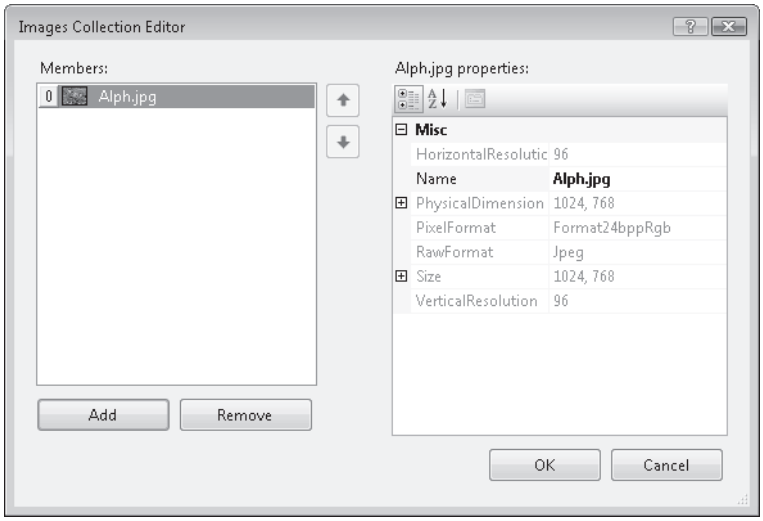


FIGURE 3-11 The Image Collection Editor

You can use the Images Collection Editor to add or remove images. You can also use it to change their order. Once you have added images to the *ImageList* component, you can set the color depth for each image by setting the *ColorDepth* property and you can set all of the images to a specified size by setting the *ImageSize* property.

At run time you can access the images contained in the *ImageList* by means of the *Images* collection, as shown in the following example:

```
' VB
PictureBox1.Image = ImageList1.Images(0)

// C#
pictureBox1.Image = imageList1.Images[0];
```

You can use *ImageList* components to provide images to other controls in your user interface. Several controls, such as *Button*, *CheckBox*, *RadioButton*, and others, host *ImageList*, *ImageKey*, and *ImageIndex* properties. You can provide images from an *ImageList* component to these controls by setting these properties.

TO PROVIDE AN IMAGE TO A CONTROL FROM AN *IMAGELIST* COMPONENT

1. Set the *ImageList* property of the control to the *ImageList* component that hosts the image you want to provide.
2. Set either the *ImageIndex* property or the *ImageKey* property to the appropriate image in the *ImageList*.

Quick Check

1. What is the difference between how a *RadioButton* control and a *CheckBox* control are used?
2. What is the purpose of an *ImageList* control and how is it used?

Quick Check Answers

1. Radio buttons allow the user to choose a single option from a set of mutually exclusive options. *Checkbox* controls allow the user to select multiple options, usually without regard to whether any other options in the group are selected.
2. An *ImageList* control is used to organize a set of related images. An *ImageList* is generally used to provide images to the controls on a form. You can set the *ImageList* property of the controls on a form to an instance of the *ImageList* and then set either the *ImageIndex* or the *ImageKey* property to specify the image.

LAB Adventure Works Ski Instructor Reservation Form

In this lab, you will build on the solution you created in the lab in Lesson 1. You will add a group of *CheckBox* controls to allow the user to indicate required ski rental equipment, a group of *RadioButton* controls to allow the user to indicate his or her ski skill level, and an *ImageList* component to integrate with the *ListView* control so the user will be able to see faces to go with the names.

EXERCISE 1 Adding to the Ski Instructor Reservation Form

1. Open the solution you completed in Lesson 1 or open the Lesson 1 completed solution from the CD.
2. Open Form1 in Design view. Drag a *GroupBox* onto the form. Set the *Text* property of the *GroupBox* to **Rental Equipment**.
3. Drag three *CheckBox* controls into the *GroupBox*. Set the *Text* properties of the *CheckBox* controls to **Skis**, **Poles**, and **Boots**.
4. Drag a *GroupBox* onto the form. Set the *Text* property of the *GroupBox* to **Skill Level**.
5. Drag three *RadioButton* controls into the *GroupBox*. Set the *Text* properties of the *RadioButton* controls to **Beginner**, **Intermediate**, and **Advanced**.
6. Drag a *Label* control and a *DateTimePicker* control onto the form. Set the *Text* property of the *Label* to **Select Lesson Time**.
7. Set the *Format* property of the *DateTimePicker* to **Time**.
8. Drag an *ImageList* component from the *Toolbox* onto the form.
9. In the Properties window, set the *ImageSize* property of the *ImageList* component to **32,32** and set the *ColorDepth* property to **Depth16Bit**.

10. In the Properties window, click Images to add four images to the *ImageList* component. You will find sample images on the Companion CD in the Images subfolder of the Code folder.
11. In the Designer, select the *ListView* control. In the Properties window, set the *SmallImageList* property to **ImageList1**.
12. In the Properties window, click Items to open the ListViewItem Collection Editor. In the ListViewItem Collection Editor, set the *ImageIndex* property for *ListViewItems* 0,1,2, and 3 to **0**,**1**,**2**, and **3**, respectively. Images should now display next to the icons in the *ListView* control.
13. Press F5 to build and test your application.

Lesson Summary

- The *CheckBox* control allows users to select options nonexclusively. You can use groups of *CheckBox* controls to allow the user to select multiple options.
- The *RadioButton* control allows you to present a group of exclusive options to the user. You can use groups of *RadioButton* controls to present a list of options, only one of which can be chosen.
- The *TrackBar* control allows the user to rapidly and graphically set a numeric value by adjusting a slider with mouse or keyboard commands.
- The *DateTimePicker* control allows the user to set a date or time. When set to Time format, times can be typed into the *DateTimePicker*. Days can be chosen from the drop-down calendar interface.
- The *MonthCalendar* control is a highly configurable control that allows the user to select a range of dates from an intuitive user interface. You can configure bold dates and set the maximum length of the date range to be selected by the user.
- The *PictureBox* control is an all-purpose control for displaying images in the user interface. It can display images in a variety of formats. The *ImageList* component organizes a collection of images and can set images to a common size and color depth.

Lesson Review

The following questions are intended to reinforce key information presented in this lesson. The questions are also available on the companion CD if you prefer to review them in electronic form.

> ***NOTE*** **ANSWERS**
>
> **Answers to these questions and explanations of why each answer choice is correct or incorrect are located in the "Answers" section at the end of the book.**

1. Which of the following are possible values for the *Checked* property of a *CheckBox* control? (Choose all that apply.)
   - **A.** *Checked*
   - **B.** *False*
   - **C.** *Indeterminate*
   - **D.** *Unchecked*
   - **E.** *True*
   - **F.** *NotChecked*
2. You are designing an application that asks the user to select a period ranging from one day to seven days in a given month. Which of the following configurations for a *MonthCalendar* control are best choices to facilitate this functionality? (Choose all that apply.)
   - **A.** Set the *MaxSelectionCount* property to 7.
   - **B.** Set the *SelectionRange* property to the first and last days of the month in question.
   - **C.** Set the *MaxDate* property to the last day of the month in question.
   - **D.** Set the *MinDate* property to the first day of the month in question.
3. Which of the following code examples correctly associates an image from an *ImageList* component with a *Button* control? Assume an *ImageList* component named *ImageList1* and a *Button* control named *Button1*. (Choose all that apply.)
   - **A.**

```
' VB
Button1.Image = ImageList1

// C#
button1.Image = imageList1;
```

B.

```
' VB
Button1.ImageList = ImageList1
Button1.ImageKey = ImageList1.Images(0)

// C#
button1.ImageList1 = imageList1;
button1.ImageKey = imageList1.Images(0);
```

C.

```
' VB
Button1.ImageList = ImageList1
Button1.ImageIndex = 0

// C#
button1.ImageList = imageList1;
button1.ImageIndex = 0;
```

D.

```
' VB
Button1.ImageList = ImageList1
Button1.ImageKey = "myImage"

// C#
button1.ImageList = imageList1;
button1.ImageKey = "myImage";
```

Lesson 3: Configuring the *WebBrowser* Control and the *NotifyIcon* Component and Creating Access Keys

Visual Studio provides several ways to extend the user interface. The *WebBrowser* control provides an all-purpose control for viewing HTML files and loading content from the World Wide Web. The *NotifyIcon* component allows you to notify users when processes are running in the background, and access keys provide additional options to the user for navigating between controls.

After this lesson, you will be able to:

- Configure properties and use methods of the *WebBrowser* control.
- Add application icons to the task bar with *NotifyIcon*.
- Associate a context menu with a *NotifyIcon* component.
- Create an access key for a control.

Estimated lesson time: 30 minutes

The *WebBrowser* Control

The *WebBrowser* control provides all of the functionality required to load and display HTML pages and other file types, as well as the functionality needed to navigate to locations on the World Wide Web. You can configure the *WebBrowser* to expose online help for your application, to load and print documents, or to display files in a variety of formats. Table 3-17 shows important properties of the *WebBrowser* control.

TABLE 3-17 Important Properties of the *WebBrowser* Control

PROPERTY	DESCRIPTION
AllowWebBrowserDrop	Determines if documents dropped into the *WebBrowser* control are automatically opened
CanGoBack	Returns whether the *WebBrowser* control is able to navigate backward
CanGoForward	Returns whether the *WebBrowser* control is able to navigate forward
Document	Returns the current HTML document in the *WebBrowser* control
DocumentStream	Returns the stream associated with the current document
DocumentText	Returns a string representation of the current document
DocumentTitle	Returns the title of the current document

PROPERTY	DESCRIPTION
DocumentType	Returns the type of the current document
IsOffline	Returns whether the system is offline
IsWebBrowserContextMenuEnabled	Determines if the standard Microsoft Internet Explorer context menu is enabled for the *WebBrowser*
ScriptErrorsSuppressed	Determines whether script errors that occur in the document are suppressed or shown in a dialog box
ScrollBarsEnabled	Determines whether scrollbars are enabled for the control
URL	Gets or sets the URL for the current document
WebBrowserShortcutsEnabled	Gets or sets whether standard Internet Explorer keyboard shortcuts are enabled for the *WebBrowser*

The *WebBrowser* control also contains a variety of methods that enable navigation within the *WebBrowser*. Table 3-18 describes important methods of the *WebBrowser* control.

TABLE 3-18 Important Methods of the *WebBrowser* Control

METHOD	DESCRIPTION
GoBack	Navigates to the previous page in the navigation history if one is available
GoForward	Navigates to the next page in the navigation history if one is available
GoHome	Navigates to the browser's home page
GoSearch	Navigates to the browser's search page
Navigate	Navigates to the specified URL
Print	Prints the current document
ShowPageSetupDialog	Displays the Internet Explorer page setup dialog box
ShowPrintDialog	Displays the Internet Explorer print dialog box
ShowPrintPreviewDialog	Displays the Internet Explorer print preview dialog box
ShowPropertiesDialog	Displays the Internet Explorer properties dialog box
ShowSaveAsDialog	Displays the Internet Explorer Save As dialog box if the document is of a type other than an HTML page
Stop	Cancels any pending navigation and stops any dynamic page elements

Navigating the Web with the *WebBrowser* Control

The *WebBrowser* control provides methods that enable navigation of the Web in your application. The primary method for navigation is the *Navigate* method. This method takes a string argument that indicates the URL for the document to be loaded into the *WebBrowser* control. The following example demonstrates the Navigate method:

```
' VB
WebBrowser1.Navigate("www.Microsoft.com")

// C#
webBrowser1.Navigate("www.Microsoft.com");
```

Once navigation is complete, the *WebBrowser* control raises the *DocumentCompleted* event. By handling this event, you can execute code after the document has loaded.

You can use other methods of the *WebBrowser* control to access your document history. The *GoBack* method navigates to the previous page in the document history, and the *GoForward* method navigates to the next page in the document history. If no page is available in the document history, there is no effect.

Working with Documents in the *WebBrowser* Control

You can also use the *Navigate* method to load other documents into the *WebBrowser* control. The following example demonstrates how to load a Microsoft Office Word document into the *WebBrowser* control:

```
' VB
WebBrowser1.Navigate("C:\Test.doc")

// C#
webBrowser1.Navigate(@"C:\Test.doc");
```

When working with documents in the *WebBrowser* control, you can allow the user to save the document by using the *ShowSaveAsDialog* method. This method displays the Save As dialog box and allows the user to choose a format to save the document.

You can also use the *WebBrowser* control for printing documents. You can call the *ShowPrintDialog* and *ShowPrintPreview* methods to enable printing of the document. These methods show the Print dialog box and the Print Preview dialog box, respectively, and allow the user to continue on to printing the document.

The *NotifyIcon* Component

The *NotifyIcon* component is not a control but a component that represents an icon that appears in the system tray. The *NotifyIcon* component is usually used with applications that run in the background. They can provide information about the program execution by displaying balloon tips, and you can associate a *ContextMenuStrip* with the *NotifyIcon* to allow

the user to execute commands from a context menu. Table 3-19 shows important properties of the *NotifyIcon* component.

TABLE 3-19 Important Properties of the *NotifyIcon* Component

PROPERTY	DESCRIPTION
BallonTipIcon	The icon that will be shown in the balloon tip. This property can be set to *None*, which displays no icon, or to *Info*, *Warning*, or *Error*.
BalloonTipText	Sets the text that is displayed in the balloon tip.
BalloonTipTitle	Sets the title of the balloon tip.
ContextMenuStrip	Gets or sets the *ContextMenuStrip* associated with the *NotifyIcon*.
Icon	The icon that is shown in the system tray.
Text	The text that is shown when the user's mouse rests on the icon in the system tray.
Visible	Indicates whether the icon is visible in the system tray.

To display a *NotifyIcon* in the system tray, you must set the *Icon* property to the icon you want to display and set the *Visible* property to *True*. You can add icons to your project by creating a new instance of the *System.Drawing.Icon* class or by adding existing icon files to your project by choosing Add Existing Item from the Project menu.

The *NotifyIcon* component contains properties that govern the display of the balloon tip. You can use the balloon tip to display information to the user. You can set the *Icon*, *Text*, and *Title* of the balloon tip by setting the *BalloonTipIcon*, *BalloonTipText*, and *BalloonTipTitle* properties, respectively. After the appropriate properties are set, you can display the balloon tip by calling the *ShowBalloonTip* method. The *ShowBalloonTip* method takes a parameter that indicates the number of seconds that the balloon tip is shown. Following is an example of the *ShowBalloonTip* method:

```
' VB
NotifyIcon1.ShowBalloonTip(12)

// C#
notifyIcon1.ShowBalloonTip(12);
```

You can associate a *ContextMenuStrip* with the *NotifyIcon* component to allow users to execute commands from the menu by right-clicking the icon. You can associate a *ContextMenuStrip* with the *NotifyIcon* component by clicking the *ContextMenuStrip* property in the Properties window and setting the property to a *ContextMenuStrip* in your solution. Creating *ContextMenuStrips* will be discussed in detail in Chapter 4, "ToolStrips, Menus, and Events."

Creating Access Keys

Access keys enable the user to move the focus to a particular control by pressing the Alt key and the access key you have defined for a particular control. In Chapter 2 you learned how to use a *Label* control to create an access key for another control. The following procedure describes how to create access keys for individual controls.

> ***NOTE*** **CREATING AN ACCESS KEY FOR A CONTROL**
>
> **To create an access key for a control with this procedure, the control must be capable of receiving the focus, it must have a *Text* property, and it must have a *UseMnemonic* property. If the control you want to create an access key for can receive the focus but does not have a *UseMnemonic* property, use the procedure described in Chapter 2. If the control cannot receive the focus, you cannot create an access key for it by any procedure.**

TO CREATE AN ACCESS KEY FOR A CONTROL

1. Set the *Text* property to the text you want the control to display.
2. In the *Text* property, prepend the letter that you want to make the access key with the ampersand (&) symbol.
3. In the Properties window, set the *UseMnemonic* property to *True*. The letter preceded by the ampersand symbol appears underlined, and at run time the user is able to shift the focus to the control by pressing the Alt key along with the underlined key.

Quick Check

- **What is the purpose of access keys?**

Quick Check Answer

- **Access keys allow you to provide keyboard shortcuts that move the focus to the control that the access key is associated with.**

LAB Creating a Web Browser

In this lab, you will create a limited but functional Web browser. You will add controls to facilitate backward and forward navigation, as well as allowing a user to type a URL and navigate to the specified location.

EXERCISE 1 Creating a Web Browser

1. In Visual Studio, start a new Windows Forms project.
2. In the Properties window for Form1, set the *Size* property to **600;400**.
3. From the *Toolbox*, drag a *SplitContainer* onto the form.

4. From the *Toolbox*, drag a *WebBrowser* control onto *Panel2*.
5. From the *Toolbox*, drag three *Button* controls and a *TextBox* control onto *Panel1*.
6. Set the *Text* property of the *Button* controls to **&Back**, **&Forward**, and **&Navigate**.
7. Set the *UseMnemonic* property of each *Button* control to *True*.
8. Select the *SplitContainer*. In the Properties window, set the *Orientation* property to *Horizontal* and adjust the size of *Panel1* so that just the buttons are showing. Set the *FixedPanel* property to *Panel1*.
9. In the Designer, double-click the Back button to open the *Button_Click* event handler for this button. Add the following line of code:

```
' VB
WebBrowser1.GoBack()

// C#
webBrowser1.GoBack();
```

10. In the Designer, double-click the Forward button to open the *Button_Click* event handler for this button. Add the following line of code:

```
' VB
WebBrowser1.GoForward()

// C#
webBrowser1.GoForward();
```

11. In the Designer, double-click the Navigate button to open the *Button_Click* event handler for this button. Add the following line of code:

```
' VB
WebBrowser1.Navigate(TextBox1.Text)

// C#
webBrowser1.Navigate(textBox1.Text);
```

12. Press F5 to build and test your application.

Lesson Summary

- The *WebBrowser* control encapsulates all of the functionality necessary to access the Internet and load a variety of document types. It contains methods that facilitate navigation of the World Wide Web and the file system.
- The *NotifyIcon* component allows you to set an icon in the system tray and provide notifications to users regarding processes running in the background. You can display messages to the user through balloon tips and enable commands by associating a *ContextMenuStrip* with the *NotifyIcon*.

- You can use the *Text* and *UseMnemonic* properties to define access keys for controls that can receive the focus. Only controls that are capable of receiving the focus can have access keys defined for them. If a control can receive the focus but does not have *Text* or *UseMnemonic* properties, you can define an access key with a *Label* control, as described in Chapter 2.

Lesson Review

The following questions are intended to reinforce key information presented in this lesson. The questions are also available on the companion CD if you prefer to review them in electronic form.

> ***NOTE*** **ANSWERS**
>
> **Answers to these questions and explanations of why each answer choice is correct or incorrect are located in the "Answers" section at the end of the book.**

1. Which of the following methods can be used to print the current document in a *WebBrowser* control? (Choose all that apply.)
   - **A.** *WebBrowser.Print*
   - **B.** *WebBrowser.ShowPrintDialog*
   - **C.** *WebBrowser.ShowPrintPreviewDialog*
   - **D.** *WebBrowser.ShowPropertiesDialog*
2. You are designing an application that runs in the background and want to enable the application to notify the user when a severe error occurs. Which of the following properties of the *NotifyIcon* component can facilitate this functionality? (Choose all that apply.)
   - **A.** *BalloonTipIcon*
   - **B.** *BalloonTipText*
   - **C.** *BalloonTipTitle*
   - **D.** *Text*
3. Which of the following are required to create an access key for a control without using an associated label? (Choose all that apply.)
   - **A.** The *Enabled* property must be set to *True*.
   - **B.** The control must have a *Text* property.
   - **C.** The *UseMnemonic* property must be set to *True*.
   - **D.** The control must be of a type that is able to receive the focus.

Chapter Review

To further practice and reinforce the skills you learned in this chapter, you can perform the following tasks:

- Review the chapter summary.
- Review the list of key terms introduced in this chapter.
- Complete the case scenarios. These scenarios set up real-world situations involving the topics of this chapter and ask you to create a solution.
- Complete the suggested practices.
- Take a practice test.

Chapter Summary

- List-based controls are used to organize and present lists of information to the user. Basic list-based controls, such as *ListBox*, *ComboBox*, and *CheckedListBox*, organize their contents in the *Items* property, which exposes common methods for adding, removing, and otherwise manipulating contained items.
- Specialized list-based controls, such as *ListView* and *TreeView*, are designed to fill specific roles. The *ListView* control allows you to display icons and other information about its contained members. The *TreeView* control displays contained members in a hierarchical tree display that the user can expand or collapse as needed.
- Value-setting controls allow the user to set a value through the user interface. *CheckBox* and *RadioButton* controls set Boolean values for their *Checked* property, allowing the user to choose yes or no to a set of presented options.
- The *ImageList* component organizes images and makes them available to controls in the application. Controls that expose an *ImageList* property can reference a given image list and display contained images.
- The *WebBrowser* control is an all-purpose control for browsing the Web and file system. It allows you to work with a variety of document types and contains methods that facilitate navigation, printing, and saving documents.
- The *NotifyIcon* component can display information about a process that is running in the background. You can display information by setting the *BalloonTip* properties and showing the balloon tip. You can expose commands to the user by associating a *ContextMenuStrip* with the *NotifyIcon* component.
- You can use the *Text* and *UseMnemonic* properties to designate access keys for a control. Any control that can receive the focus and has *Text* and *UseMnemonic* properties can define its own access key. If a control can receive the focus but does not have *Text* or *UseMnemonic* properties, you can define an access key using a *Label* control as shown in Chapter 2.

Key Terms

Do you know what these key terms mean? You can check your answers by looking up the terms in the glossary at the end of the book.

- access keys
- list
- list-based control
- value-setting control

Case Scenarios

In the following case scenarios, you will apply what you've learned about how to use controls to design user interfaces. You can find answers to these questions in the "Answers" section at the end of this book.

Case Scenario 1: Incorporating List-Based Controls into the User Interface

Humongous Insurance has grown so large that it needs some help keeping track of its employees. You have been put on the team that will design the new human resources application. Other developers will supply a programmatic representation of the organization chart and a database of information about the employees. Your job is to create a user interface that allows the user to browse the organization chart and allows additional information about each employee to be displayed in the user interface.

QUESTIONS

Answer the following questions for your manager:

1. What is your suggested control layout for the user interface? How will you be able to display the organization chart in a compact, easy-to-browse format?
2. How can we display photos of our employees as part of this application?

Case Scenario 2: Working with Files and Background Processes

As part of its document backup plan, Humongous Insurance has created an automated program that reads its electronic documents in a variety of different formats (such as .doc, .txt, and .htm), saves them to a backup location, and prints a hard copy on a high-throughput printer. For the most part, this application works fine without user interaction and displays no user interface. Occasionally, however, a problem occurs with a document that requires user intervention. You have been put in charge of designing the user interface for the rare occasions that do arise.

TECHNICAL REQUIREMENTS

- The user interface must display only when there is a problem and cannot be launched without action by a user.
- The user must be able to examine the document and manually save and print it.

QUESTIONS

Answer the following questions for your manager:

1. How can we warn the user of a problem without displaying the user interface at all times? How will we allow the user to launch a user interface when there is a problem?
2. When there is a problem, how can we design the user interface so that the user is able to examine, print, and save individual files?

Suggested Practices

To master the Add and Configure a Windows Forms Control exam objective, complete the following practices, as well as the practices in Chapter 2.

- **Practice 1** Build an application that duplicates the functionality of Windows Explorer. You should be able to display a directory tree in one pane and files in a particular directory in another pane.
- **Practice 2** Build an application that acts like an appointment book. It should allow the user to choose a date and time, add information about the appointment, track and display details about the appointment, and visually display to the user on a *MonthCalendar* control what days have appointments set.
- **Practice 3** Expand the Web browser you created in Lesson 3, "Configuring the *WebBrowser* Control and the *NotifyIcon* Component and Creating Access Keys," to disable the Back and Forward buttons if *webBrowser1.CanGoBack* or *webBrowser1.CanGoForward* are *False*. You can do this by handling the *WebBrowser.CanGoBackChanged* and *WebBrowser.CanGoForwardChanged* events. Also, allow the user to navigate to a page by typing an address in the *TextBox* control and pressing Enter.

Take a Practice Test

The practice tests on this book's companion CD offer many options. For example, you can test yourself on just the content covered in this chapter, or you can test yourself on all the 70-505 certification exam content. You can set up the test so that it closely simulates the experience of taking a certification exam, or you can set it up in study mode so that you can look at the correct answers and explanations after you answer each question.

MORE INFO **PRACTICE TESTS**

For details about all the practice test options available, see the "How to Use the Practice Tests" section in this book's Introduction.

CHAPTER 5

Configuring Connections and Connecting to Data

Typically, most real-world applications use databases as a store for the data in that application. For example, inventory systems, contact management systems, and airline reservation systems store data in a database and then retrieve the necessary records into the application as needed. In other words, the data that an application uses is stored in a database external to the actual application, and it is retrieved into the application as required by the program.

When creating applications that work with data, the Microsoft .NET Framework provides many classes that aid in the process. The classes that you use for common data tasks, such as communicating, storing, fetching, and updating data, are all located in the System.Data namespace. The classes in the *System.Data* namespace make up the core data access objects in the .NET Framework. These data access classes are collectively known as ADO.NET.

Before you can begin working with data in an application, you must first establish and open a connection to communicate with the desired data source. This chapter describes how to create the various connection objects that are used to connect applications to different data sources and sets the basis for working with data in the following chapters. After learning to establish connections to databases in this chapter, we will move on to Chapter 6, "Working with Data in a Connected Environment," which provides instructions for running queries, saving data, and creating database objects directly between your application and a database. Chapter 7, "Create, Add, Delete, and Edit Data in a Disconnected Environment," describes how to create *DataSet* and *DataTable* objects that allow you to temporarily store data while the data is being used in a running application. Finally, Chapter 8, "Implementing Data-Bound Controls," provides information on binding data to be displayed and worked with in Windows Forms controls.

Typically, data sources are relational databases like Microsoft SQL Server and Oracle, but you can also connect to data in files such as Microsoft Office Access (.mdb) and SQL Server (.mdf) database files. The connection object you use is based on the type of data source your application needs to communicate with.

Exam objectives in this chapter:

- Manage connections and transactions.
  - Configure a connection to a database using the Data Source Configuration Wizard.
  - Configure a connection to a database using Server Explorer.
  - Configure a connection to a database using the *Connection* class.
  - Connect to a database using specific database connection objects.
  - Enumerate instances of SQL Server.
  - Open an ADO.NET connection to a database.
  - Close an ADO.NET connection to a database by using the *Close* method of the connection object.
  - Protect access to the connection details of a data source.
  - Create a connection designed for reuse in a connection pool.
  - Control a connection pool by configuring *ConnectionString* values based on database type.
  - Use the *Connection* events to detect database information.
  - Handle exceptions when connecting to a database.

Lessons in this chapter:

Before You Begin

To complete the lessons in this chapter, you must have:

- A computer that meets or exceeds the minimum hardware requirements listed in the "Introduction" at the beginning of the book.
- Microsoft Visual Studio 2008 Professional Edition installed on your computer.
- An understanding of Microsoft Visual Basic or C# syntax and familiarity with the .NET Framework.
- A basic understanding of relational databases.
- Available data sources, including SQL Server 2005 or later (SQL Server 2005 Express Edition or later is acceptable), the *Northwind* sample database for SQL Server, and the Nwind.mdb Access database file. Directions for setting up the sample databases are located in the Setting up Sample Databases Read Me file on the companion CD.

REAL WORLD

Steve Stein

At a previous employer, I was responsible for extracting data from an arcane proprietary database that was virtually impossible to connect to directly. As a result, time-consuming reports were periodically generated that were then imported into a workable database management system for further processing. Thinking back, I realize how much easier life would have been if I had been able to spin up a connection object and communicate directly with the data source without the need for the intermediary process of creating reports and exporting and importing data.

Lesson 1: Creating and Configuring Connection Objects

This lesson describes the two ways to create and configure connection objects:

- Through a user interface (UI), using the Add Connection dialog box
- Programmatically, by handcrafting the objects in code

Whether you choose to create connections through the UI or programmatically, the result is the same—a configured connection object ready to open a connection and communicate with your data source. For this lesson, we will focus only on creating connection objects as opposed to actually connecting and communicating with a data source. In Lesson 2, "Connecting to Data Using Connection Objects," we will move on to the next level to open the connection and retrieve information from the data source.

After this lesson, you will be able to:

- Configure a connection to a database using the Server Explorer.
- Configure a connection to a database using the Data Source Configuration Wizard.
- Configure a connection to a database using the *Connection* class.
- Connect to a database using specific database connection objects.

Estimated lesson time: 30 minutes

What Is a Connection Object?

A connection object is simply a representation of an open connection to a data source. The easiest way to describe a connection object is, first, to explain what a connection object is not! A connection object does not fetch or update data, it does not execute queries, and it does not contain the results of queries. It is merely the pipeline through which commands and queries send their SQL statements and receive results. Although connection objects typically can be thought of as the place where you set your connection string, they have additional methods for working with the connection, such as methods that open and close connections, as well as methods for working with connection pools and transactions. Essentially, connection objects provide a conduit for sending commands to a database and retrieving data and information into your application, as shown in Figure 5-1.

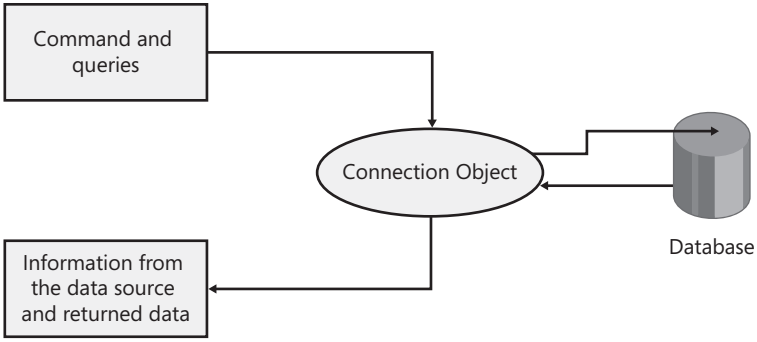


FIGURE 5-1 Connection objects are your application's communication pipeline to a database

Creating Connections in Server Explorer

To simplify the process of creating applications that access data, Visual Studio provides the Server Explorer window as a central location to manage data connections independent of any actual projects. In other words, you can create data connections in Server Explorer and access them in any project. Data connections created in Server Explorer are user-specific settings in Visual Studio that display the connections each time you open Visual Studio (instead of creating connections as part of developing a specific application that stores them in that application). Of course, you can create data connections as part of the development process from within an open project, but that is covered in the next section.

Creating Connections Using Data Wizards

Visual Studio provides a few wizards that simplify the process of creating applications that access data and that create data connections as a result of completing the wizards. The main wizard for bringing data into an application is the Data Source Configuration Wizard. When you run the Data Source Configuration Wizard and select the database path, you end up with a configured connection object ready to use in your application. In addition to creating a configured connection object, the Data Source Configuration Wizard allows you to select the database objects you want to use in your application.

Creating Connection Objects Programmatically

When you do not want to use the visual tools previously described and need to create your connections manually, it is easy to create connection objects in code programmatically. The first step is to decide which type of connection object to create. The choice is fairly simple because it depends on the back-end data source your application needs to communicate with.

Table 5-1 lists the primary connection objects available in ADO.NET and the data sources they are designed to access.

TABLE 5-1 Connection Objects

NAME	TARGET DATA SOURCE
SqlConnection	SQL Server 7.0 and later databases
OleDbConnection	OLE DB data sources (such as Office Access databases through Jet 4.0)
OdbcConnection	Open database connectivity (ODBC) data sources such as a Data Source Name (DSN) as defined in the ODBC Data Source Administrator dialog box
OracleConnection	Oracle 7.3, 8i, or 9i databases

The properties, methods, and events associated with the connection objects in this table vary because each connection object is designed to efficiently connect and interact with its respective data sources. However, each connection object contains the same base properties, methods, and events that are inherited from the *System.Data.Common.DbConnection* class.

Table 5-2 lists the properties common to all connection objects.

TABLE 5-2 Connection Properties

NAME	DESCRIPTION
ConnectionString	Gets or sets the string used to open the connection.
ConnectionTimeout	Read only. Gets the time to wait while establishing a connection before terminating the attempt and generating an error.
Database	Read only. Gets the name of the current database after a connection is opened or the database name specified in the connection string before the connection is opened.
DataSource	Read only. Gets the name of the database server to which it is connected.
ServerVersion	Read only. Gets a string that represents the version of the server to which the object is connected.
State	Read only. Gets a combination of *System.Data.ConnectionState* values that describes the state of the connection.

Table 5-3 lists the methods common to all connection objects.

TABLE 5-3 Connection Methods

NAME	DESCRIPTION
BeginDbTransaction	Starts a database transaction.
BeginTransaction	Starts a database transaction.
ChangeDatabase	Changes the current database for an open connection.
Close	Closes the connection to the database. This is the preferred method of closing any open connection.
CreateCommand	Creates and returns a *System.Data.Common.DbCommand* object associated with the current connection.
CreateDbCommand	Creates and returns a *System.Data.Common.DbCommand* object associated with the current connection.
EnlistTransaction	Enlists in the specified transaction as a distributed transaction.
GetSchema	Returns schema information for the data source of this *System.Data.Common.DbConnection* class.
New	Initializes a new instance of the *System.Data.Common.DbConnection* class.
OnStateChange	Raises the *System.Data.Common.DbConnection.StateChange* event.
Open	Opens a database connection with the settings specified by the *System.Data.Common.DbConnection.ConnectionString*.

Table 5-4 lists the events common to all connection objects.

TABLE 5-4 Connection Events

NAME	DESCRIPTION
StateChange	Occurs when the state of the connection changes
InfoMessage	Occurs when the server returns a warning or informational message

To create connections programmatically using the four primary data providers, you start by instantiating a new connection object and setting its *ConnectionString* property that you will use to open the connection.

NOTE ***SYSTEM.DATA.ORACLECLIENT*** **REFERENCE**

By default, Microsoft Windows applications in Visual Studio are created with references to the *System.Data.SqlClient, System.Data.OleDb*, and *System.Data.Odbc* namespaces, so these are immediately available to be coded against and appear in IntelliSense with no further action. By default, a reference to the *System.Data.OracleClient* namespace is not included and must be added to your application to create *OracleConnection* objects.

Creating SQL Server Connection Objects in Code

You create *SqlConnection* objects with the *New* keyword. You can instantiate the connection and set the connection string in the same call, or you can assign the connection string to the *SqlConnection.ConnectionString* property after instantiating the connection. Be sure to replace ServerName and DatabaseName with valid values for your environment. To eliminate the need to qualify the objects fully in code, add an *Imports System.Data.SqlClient* statement (Visual Basic) or *using System.Data.SqlClient;* statement (C#) to the top of your code file. Use the *WithEvents* keyword (in Visual Basic) or create event handlers in C# if your application needs to respond to the connection objects events.

```
' VB
 Private WithEvents ConnectionToSql As New SqlConnection _
    ("Data Source=ServerName;Initial Catalog=DatabaseName;Integrated Security=True")

// C#
SqlConnection ConnectionToSql = new SqlConnection
    ("Data Source=ServerName;Initial Catalog=DatabaseName;Integrated Security=True");
```

Creating OLE DB Connection Objects in Code

You create *OleDbConnection* objects with the *New* keyword. You can instantiate the connection and set the connection string in the same call, or you can assign the connection string to the *OleDbConnection.ConnectionString* property after instantiating the connection. Be sure to replace the data source with a valid path if you are connecting to an Office Access database, or replace the connection string with a valid connection string for the OLE DB data source you want to connect to. To eliminate the need to fully qualify the objects in code, add an *Imports System.Data.OleDb* statement (Visual Basic) or *using System.Data.OleDb;* statement (C#) to the top of your code file.

```
' VB
Private WithEvents ConnectionToOleDb As New System.Data.OleDb.OleDbConnection _
   ("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=""Nwind.mdb"";Persist Security
Info=False")
```

```
// C#
System.Data.OleDb.OleDbConnection ConnectionToOleDb = new System.Data.OleDb.
OleDbConnection
   ("Provider=Microsoft.Jet.OLEDB.4.0;Data Source=\"Nwind.mdb";Persist Security
Info=False");
```

Creating ODBC Connection Objects in Code

You create *OdbcConnection* objects with the New keyword. You can instantiate the connection and set the connection string in the same call, or you can assign the connection string to the *OdbcConnection.ConnectionString* property after instantiating the connection. Be sure to replace the connection string with a valid connection string for the ODBC data source you want to connect to. To eliminate the need to qualify the objects fully in code, add an Imports *System.Data.Odbc* statement (Visual Basic) or using *System.Data.Odbc*; statement (C#) to the top of your code file.

```
' VB
 Private WithEvents ConnectionToOdbc As New OdbcConnection _
    ("Dsn=MS Access Database;dbq=C:\Nwind.mdb;defaultdir=C:\DataSources;" & _
    "driverid=281;fil=MS Access;maxbuffersize=2048;pagetimeout=5;uid=admin")

// C#
OdbcConnection ConnectionToOdbc = new OdbcConnection
    ("Dsn=MS Access Database;dbq=C:\\DataSources;" +
    "driverid=281;fil=MS Access;maxbuffersize=2048;pagetimeout=5;uid=admin");
```

Creating Oracle Connection Objects in Code

You create *OracleConnection* objects with the *New* keyword. You can instantiate the connection and set the connection string in the same call, or you can assign the connection string to the *OracleConnection.ConnectionString* property after instantiating the connection. Be sure to replace the connection string with a valid one for the Oracle database you want to connect to. To eliminate the need to qualify the objects fully in code, add an *Imports System.Data.OracleClient* statement (Visual Basic) or *using System.Data.OracleClient;* statement (C#) to the top of your code file.

```
' VB
 Private WithEvents ConnectionToOracle As New OracleConnection _
    ("Data Source=Oracle8i;Integrated Security=yes")

// C#
private OracleConnection ConnectionToOracle = new OracleConnection
    ("Data Source=Oracle8i;Integrated Security=yes");
```

LAB Creating New Data Connections

In this lab you will practice creating new Data Connections in Server Explorer and using the Data Source Configuration Wizard.

EXERCISE 1 Creating Connections in Server Explorer

The following steps describe how to create a Data Connection (a connection to a database) in Server Explorer:

1. If the Server Explorer window is not visible, select Server Explorer from the View menu.
2. Right-click the *Data Connections* node and select Add Connection.

   The first time you add a connection in Visual Studio, the Choose Data Source dialog box opens.

> ***NOTE*** **ADD CONNECTION DIALOG BOX**
>
> **If the Add Connection dialog box opens instead of the Choose Data Source dialog box, click the Change button located at the top of the Add Connection dialog box.**

   The Choose Data Source dialog box (or the similar Change Data Source dialog box) is where you select the data source you want to connect to, as well as the data provider to use for the connection. Notice how the proper data provider is automatically populated when you select different data sources. You can choose any valid provider you want for any selected data source, but Visual Studio automatically selects the most appropriate data provider based on the selected data source.

   For the first connection, we'll create a connection to the *Northwind Traders* sample database in SQL Server.

3. Select Microsoft SQL Server for the data source and click OK.

   The Add Connection dialog box now appears with Microsoft SQL Server as the selected data source.

> ***NOTE*** **.NET FRAMEWORK DATA PROVIDERS**
>
> **The .NET Framework Data Provider for SQL Server is designed to connect to SQL Server 7 and later versions. When connecting to SQL Server 6 or earlier, select the <other> data source and select the .NET Framework Data Provider for OLE DB. Then, in the Add Connection dialog box, select the Microsoft OLE DB Provider for SQL Server.**

4. Type the name of your SQL Server in the server name area.
5. Select the appropriate method of authentication to access your SQL Server.
6. Choose the Select Or Enter A Database Name option and select the *Northwind* database from the drop-down list.

7. You can verify the connection is valid by clicking Test Connection and then clicking OK to close the dialog box and create the connection in Server Explorer.

   After creating the connection, the Properties window provides information related to the connection, as well as information related to the actual database you are connected to.

8. Select the connection you just created in the Server Explorer window to view the available information in the properties window.

> ***NOTE*** **CONNECTION PROPERTIES**
>
> **The available properties are based on the type of data source you are connected to as well as the state of the connection. If the connection is closed, you might see only a small list of properties made up of the connection string used to connect to the database, the specific .NET Framework data provider used by the connection, and the state of the connection. To view additional properties, it is necessary to open the connection by expanding the *Connection* node in Server Explorer. Once open, the connection provides additional properties, such as the database owner, whether the database is case sensitive, and the type of database and version number.**

EXERCISE 2 Creating Connections Using the Data Source Configuration Wizard

To create data connections using the Data Source Configuration Wizard, perform the following steps:

1. Create a Windows Forms application.
2. Select Add New Data Source from the Data menu.
3. The default data source type is Database, so just click Next.
4. The Choose Your Data Connection page of the wizard is where you create your connection object.

> ***NOTE*** **AVAILABLE CONNECTIONS**
>
> **The drop-down list is populated with the connections already available in Server Explorer. If you completed the previous section and created a data connection to the *Northwind* database, it will be available in this drop-down list.**

5. For this exercise we will create a new connection, so click the New Connection button to open the Add Connection dialog box.

> ***NOTE*** **ADD CONNECTION DIALOG BOX**
>
> **Just like adding a new connection to Server Explorer, you use the Add Connection dialog box to create the connection. Basically, when creating new connections through the UI, whether using one of the data wizards or Server Explorer, the Add Connection dialog box is always used.**

6. Type the name of your SQL server in the server name area.
7. Select the appropriate method of authentication to access your SQL server.
8. Choose the Select Or Enter A Database Name option and select the *Northwind* database from the drop-down list.
9. You can verify that the connection is valid by clicking Test Connection and then clicking OK to close the dialog box.

> ***NOTE*** **INCLUDING SENSITIVE DATA**
>
> **If your connection uses SQL authentication and requires a user name and password to connect to your database, the option to include or exclude sensitive data in the connection string is enabled. By default, the connection string does not include sensitive data, and you need to provide this information in your application when you attempt to open the connection and connect to the database. You can select the option to include sensitive data in the connection string, but this is not considered best practice because users who have access to the connection string might be able to view the password. Using Integrated Security is the recommended option.**

At this point in the wizard, you have successfully created your data connection and can view the connection string by expanding the Connection string node. To add the connection to your project, finish the wizard by completing the following steps.

10. Click Next. You are presented with the option of saving the Connection string in the application configuration file as well as providing a name for the connection. By default, the selection is set to save the connection; this is probably a good idea for most applications. Saving your connection in the application configuration file would be advantageous if, after deployment, you wanted to point to a different data source. Then you (or a systems administrator) could easily modify the configuration setting rather than having to change the connection string in code and recompile and redeploy the application. Once a connection string is saved to the application configuration file, you can access and modify it using the Project Designer. Open the Project Designer by clicking the My Project toolbar button (VB) or the Properties toolbar button (C#) in Solution Explorer. After the Project Designer opens, click the Settings tab to access the connection strings stored in your application.
11. The Choose Your Database Objects page of the wizard allows you to select the Tables, Views, Stored Procedures, and so on to be used in your application. For this lesson, expand the *Tables* node and select the Customers and Orders tables.
12. Click Finish. A typed dataset with the connection object defined in the wizard is added to your project.

Now that you've completed the wizard, let's take a look at where the connection is and what it contains. The connection created as a result of running the wizard is located within

the designer-generated dataset code file. To view the actual connection object, open the dataset in the Dataset Designer by double-clicking the dataset object in Solution Explorer. (The *Dataset* object is the *NorthwindDataSet.xsd* node.) Select the title bar of a *TableAdapter* on the design surface (for example, select *CustomersTableAdapter*). The connection information is available in the Properties window, where you can expand the node and see the name, modifier, and connection string.

> ***NOTE*** ***CONNECTIONSTRING* PROPERTY**
>
> **The *ConnectionString* property displays the connection string saved in the application configuration file. Modifying the connection string here in the Properties window is the same as editing the connection string in the Settings file and affects all connections that reference that setting.**

Quick Check

1. How do I decide which connection object I need to create?
2. What is the minimum information required to create a connection object?

Quick Check Answers

1. Choose a connection object by selecting the .NET Framework Data Provider that is designed to work best with your particular data source.
2. At the least, you need to know the valid connection string that you can use to connect to your data source.

Lesson Summary

- Connection objects provide two-way communication between your application and a data source.
- Connection objects can be added to Server Explorer, where they can then be easily incorporated into future projects.
- To create connection objects, you must have a valid connection string and the proper credentials to access the data source.
- Connection objects can be created either through the UI or programmatically, depending on user preference and development style.
- There are four primary connection objects, one for each of the .NET Framework Data Providers.

Lesson Review

The following questions are intended to reinforce key information presented in this lesson. The questions are also available on the companion CD if you prefer to review them in electronic form.

> ***NOTE*** **ANSWERS**
>
> **Answers to these questions and explanations of why each answer choice is correct or incorrect are located in the "Answers" section at the end of the book.**

1. Where is the connection object located that was created as a result of running the Data Source Configuration Wizard?
   - **A.** In the application configuration file
   - **B.** In the Data Sources window
   - **C.** In the designer-generated dataset code file
   - **D.** In the generated form code
2. When should you use the *OleDbConnection* object? (Choose all that apply.)
   - **A.** When connecting to an Oracle database
   - **B.** When connecting to an Office Access database
   - **C.** When connecting to SQL Server 6.x or later
   - **D.** When connecting to SQL Server 2000
   - **E.** When connecting to SQL Server 2000
3. What user interface component is used to create connections?
   - **A.** The Data Source Configuration Wizard
   - **B.** The Server Explorer window
   - **C.** The Add Connection dialog box
   - **D.** The Properties window

Lesson 2: Connecting to Data Using Connection Objects

Now that you have learned how to create connection objects using the primary .NET data providers, let's start using them and actually connect to some data sources. This lesson will explain how to use a connection object and open a connection to a data source. After opening the connection, you will verify that the connection is opened by examining the *ConnectionState property*. Once you verify that the connection state is opened, you will also cause the *InfoMessage* event to fire and display the message returned by the data source.

After this lesson, you will be able to:

- Open an ADO.NET connection to a database.
- Close an ADO.NET connection to a database by using the *Close* method of the connection object.
- Use the connection events to detect database information.

Estimated lesson time: 45 minutes

Opening and Closing Data Connections

Open and close connections using the appropriately named *Open* and *Close* methods. To open a connection to a database, the connection object must contain a connection string that points to a valid data source, as well as enough information to pass the appropriate credentials to the data source. When connections are opened and closed, you can keep an eye on the state of the connection by responding to the *StateChange* event. The following example shows how to open and close connections and how to update the text in a label in reaction to the *StateChange* event. We will also demonstrate how you can use the *InfoMessage* event to provide informational messages from a data source to the application. And, finally, we will demonstrate how the connection object can provide information about the data source by retrieving metadata (for example, the server version number) from an open connection.

Connection Events

Connection objects provide the *StateChanged* and *InfoMessage* events to provide information to your application regarding the status of the database and information pertaining to commands executed using a specific connection object.

- ***StateChanged* event** This event is raised when the current state of the database changes from *Open* to *Closed*.

- ***InfoMessage* event** In addition to monitoring the state of a connection, each connection object provides an *InfoMessage* event that is raised when warnings or messages are returned from the server. Informational messages are typically provided when low-severity errors are returned by the data source that the connection object is connected to. For example, SQL Server errors with a severity of 10 or less are provided to the *InfoMessage* event.

> ***NOTE*** **SEVERITY LEVELS**
>
> **Each error message in SQL Server has an associated severity level. The severity level, as its name implies, provides a clue to the type of error being returned. Severity levels range from 0 through 25. Errors with severity levels between 0 and 19 can typically be handled without user intervention, but errors with severity levels between 20 and 25 typically cause your connection to close. For more information on SQL Server errors and severity levels, see the Error Message Severity Levels topic in the SQL Books Online.**

LAB Practice Opening and Closing Data Connections

In this lab you will practice working with connection objects by opening and closing connections and displaying connection information to the user.

EXERCISE 1 Opening and Closing Data Connections

To demonstrate working with connection objects, perform the following steps:

1. Create a new Windows application and name it **DataConnections**.
2. Because Windows applications are not created with a reference to the *System.Data.OracleClient* namespace, from the Project menu, select the Add Reference command, locate the *System.Data.OracleClient* component, and click OK.
3. Add 12 buttons to the form, setting the *Name* and *Text* properties as shown in Table 5-5.

> ***NOTE*** **SIMILAR CONNECTIONS**
>
> **No matter which connection objects you use, the methods for opening and closing, handling events, and so on, are the same. Feel free to only set up the example using the connection object for the provider you are interested in working with.**

TABLE 5-5 Button Settings for Data Connections Form

NAME PROPERTY	TEXT PROPERTY
OpenSqlButton	Open SQL
OpenOleDbButton	Open OLE DB
OpenOdbcButton	Open ODBC
OpenOracleButton	Open Oracle
CloseSqlButton	Close SQL
CloseOleDbButton	Close OLE DB
CloseOdbcButton	Close ODBC
CloseOracleButton	Close Oracle
GetSqlInfoButton	Get SQL Info
GetOleDbInfoButton	Get OLE DB Info
GetOdbcInfoButton	Get ODBC Info
GetOracleInfoButton	Get Oracle Info

4. Add four labels to the form, setting the *Name* and *Text* properties as shown in Table 5-6.

TABLE 5-6 Label Settings for Data Connections Form

NAME PROPERTY	TEXT PROPERTY
SqlConnectionStateLabel	Closed
OleDbConnectionStateLabel	Closed
OdbcConnectionStateLabel	Closed
OracleConnectionStateLabel	Closed

Arrange the controls so the form layout looks similar to Figure 5-2.

To create the connection objects for this lesson, you will take the code examples from Lesson 1, "Creating and Configuring Connection Objects," and add them to your form as follows.

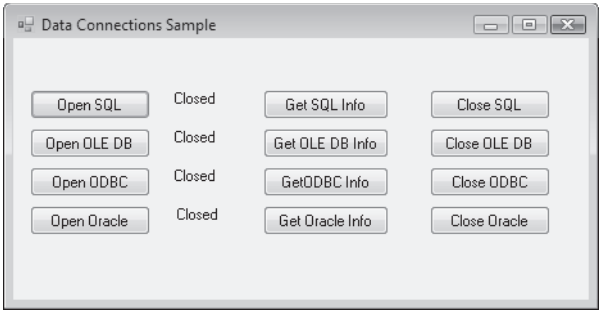


FIGURE 5-2 Form with controls arranged in preparation for creating connection objects

5. Open the form you just created in code view.
6. Add the code to create all four connection objects so that you end up with code that looks like the following:

> ***IMPORTANT*** **CONNECTION STRINGS**
>
> **Be sure to modify the connection strings to point to your specific server and database for each provider.**

```
' VB
Imports System.Data.SqlClient
Imports System.Data.OleDb
Imports System.Data.Odbc
Imports System.Data.OracleClient

Public Class Form1
    ' Declare the connection objects for the four data providers
    Private WithEvents ConnectionToSql As New SqlConnection( _
        "Data Source=.\sqlexpress;Initial Catalog=Northwind;Integrated
Security=True")
    Private WithEvents ConnectionToOleDb As New _
        System.Data.OleDb.OleDbConnection( _
            "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=""C:\DataSources\Nwind.
mdb"";" & _
        "Persist Security Info=False")
    Private WithEvents ConnectionToOdbc As New OdbcConnection( _
        "Dsn=MS Access Database;dbq=C:DataSources\Nwind.mdb;defaultdir=C:\
DataSources;" & _
        "driverid=281;fil=MS Access;maxbuffersize=2048;pagetimeout=5;uid=admin")
    Private WithEvents ConnectionToOracle As New OracleConnection("Data
Source=MyOracleDB;Integrated Security=yes;")
End Class
```

```
// C#
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Data.SqlClient;
using System.Data.OleDb;
using System.Data.Odbc;
using System.Data.OracleClient;

namespace DataConnections
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        // Declare the connection objects for the four data providers
        private SqlConnection ConnectionToSql = new SqlConnection(
            "Data Source=.\\sqlexpress;Initial Catalog=Northwind;Integrated
Security=True");
        private OleDbConnection ConnectionToOleDb = new
            System.Data.OleDb.OleDbConnection(
                "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\\DataSources\\
Nwind.mdb");
        private OdbcConnection ConnectionToOdbc = new OdbcConnection(
            "Dsn=MS Access Database;dbq=C:\\DataSources\\Nwind.mdb;" +
            "defaultdir=C:\\DataSources;driverid=281;fil=MS
Access;maxbuffersize=2048;" +
            "pagetimeout=5;uid=admin");
        private OracleConnection ConnectionToOracle = new OracleConnection(

            "Data Source=MyOracleDB;Integrated Security=yes;");

    }
}
```

To open connections to a database, use the connection object's *Open* method. To demonstrate this, you will call the *Open* method for each connection when the open buttons are clicked.

7. Create event handlers for the open buttons for each provider and add the following code, which opens the connection to the database when the open buttons are clicked:

```
' VB
Private Sub OpenSqlServerButton_Click(ByVal sender As System.Object, _
       ByVal e As System.EventArgs) Handles OpenSqlServerButton.Click
    ConnectionToSql.Open()
End Sub

Private Sub OpenOleDbButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles OpenOleDbButton.Click
   ConnectionToOleDb.Open()
End Sub

Private Sub OpenOdbcButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles OpenOdbcButton.Click
   ConnectionToOdbc.Open()
End Sub

Private Sub OpenOracleButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles OpenOracleButton.Click
   ConnectionToOracle.Open()
End Sub

// C#
private void OpenSqlServerButton_Click(object sender, EventArgs e)
{
    ConnectionToSql.Open();
}
private void OpenOleDbButton_Click(object sender, EventArgs e)
{
    ConnectionToOleDb.Open();
}
private void OpenOdbcButton_Click(object sender, EventArgs e)
{
    ConnectionToOdbc.Open();
}
private void OpenOracleButton_Click(object sender, EventArgs e)
{
    ConnectionToOracle.Open();
}
```

To close database connections, use the connection object's *Close* method. Technically, you can also call the *Dispose* method of the connection object to close the connection, but the preferred technique is to call the *Close* method. It is worth noting that calling the *Close* method also rolls back all pending transactions and releases the connection back

to the connection pool. To implement this, create event handlers for the close buttons for each provider and add code to call the *Close* method to the body of the handler.

8. Add the *Close* methods into the event handlers to close the connection to the database when the close buttons are clicked.

```
' VB
Private Sub CloseSqlButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles CloseSqlButton.Click
   ConnectionToSql.Close()
End Sub

Private Sub CloseOleDbButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles CloseOleDbButton.Click
   ConnectionToOleDb.Close()
End Sub

Private Sub CloseOdbcButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles CloseOdbcButton.Click
   ConnectionToOdbc.Close()
End Sub

Private Sub CloseOracleButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles CloseOracleButton.Click
   ConnectionToOracle.Close()
End Sub

// C#
private void CloseSqlButton_Click(object sender, EventArgs e)
{
    ConnectionToSql.Close();
}

private void CloseOleDbButton_Click(object sender, EventArgs e)
{
    ConnectionToOleDb.Close();
}

private void CloseOdbcButton_Click(object sender, EventArgs e)
{
    ConnectionToOdbc.Close();
}

private void CloseOracleButton_Click(object sender, EventArgs e)
{
    ConnectionToOracle.Close();
}
```

When the state of a connection changes, the value in the *CurrentState* property of the connection object is updated to reflect the connection's current state. When you are opening and closing a connection, you can inspect the value in this property to verify that the connection is actually opening and closing. Each connection object raises a *StateChange* event that you can respond to in order to monitor the state of the connection. To populate the connection-state labels, we need to create event handlers for the *StateChange* events for each provider. Inside the *StateChange* event handlers, add code that updates the connection-state labels with the value of the connection's *CurrentState* property, which is provided as an event argument.

9. Add the following code to the form, which updates the connection-state label values whenever the current state of a connection changes. Create the form load handler for C# so you can add the *StateChange* event handlers.

```
' VB
Private Sub ConnectionToSql_StateChange(ByVal sender As Object, _
    ByVal e As System.Data.StateChangeEventArgs) _
    Handles ConnectionToSql.StateChange
   SqlConnectionStateLabel.Text = e.CurrentState.ToString
End Sub

Private Sub ConnectionToOleDb_StateChange(ByVal sender As Object, _
    ByVal e As System.Data.StateChangeEventArgs) _
    Handles ConnectionToOleDb.StateChange
   OleDbConnectionStateLabel.Text = e.CurrentState.ToString
End Sub

Private Sub ConnectionToOdbc_StateChange(ByVal sender As Object, _
    ByVal e As System.Data.StateChangeEventArgs) _
    Handles ConnectionToOdbc.StateChange
   OdbcConnectionStateLabel.Text = e.CurrentState.ToString
End Sub

Private Sub ConnectionToOracle_StateChange(ByVal sender As Object, _
    ByVal e As System.Data.StateChangeEventArgs) _
    Handles ConnectionToOracle.StateChange
   OracleConnectionStateLabel.Text = e.CurrentState.ToString
End Sub

// C#
private void Form1_Load(object sender, EventArgs e)
{
    ConnectionToSql.StateChange += new
        System.Data.StateChangeEventHandler(this.ConnectionToSql_StateChange);
    ConnectionToOleDb.StateChange += new
        System.Data.StateChangeEventHandler(this.ConnectionToOleDb_StateChange);
```

```
    ConnectionToOdbc.StateChange += new
        System.Data.StateChangeEventHandler(this.ConnectionToOdbc_StateChange);
    ConnectionToOracle.StateChange += new
        System.Data.StateChangeEventHandler(this.ConnectionToOracle_StateChange);
}

private void ConnectionToSql_StateChange(object sender,
    StateChangeEventArgs e)
{
    SqlConnectionStateLabel.Text = e.CurrentState.ToString();
}

private void ConnectionToOleDb_StateChange(object sender,
    StateChangeEventArgs e)
{
    OleDbConnectionStateLabel.Text = e.CurrentState.ToString();
}

private void ConnectionToOdbc_StateChange(object sender,
    StateChangeEventArgs e)
{
    OdbcConnectionStateLabel.Text = e.CurrentState.ToString();
}

private void ConnectionToOracle_StateChange(object sender,
    StateChangeEventArgs e)
{
    OracleConnectionStateLabel.Text = e.CurrentState.ToString();
}
```

10. Press F5 to run the application and test the form to see the functionality you have so far.
11. When the form opens, click the Open SQL button and verify that the connection-state label changes to show that the connection is now open.
12. Click the Close SQL button and verify that the connection-state label changes to reflect the current state of the connection, which is now closed.

    To demonstrate use of the *InfoMessage* event, you need to create an event handler to process the message. To eliminate the need to create a database object that throws an error with a low severity, you can take advantage of a feature built into the *SqlConnection* object that allows you to capture errors with severities up to severity level 16 by setting the connection object's *FireInfoMessageEventOnUserErrors* property to *True* before executing a method that will force an error to be thrown.
13. Add the following code, which will handle the click event for *GetSqlInfoButton* and the *SqlConnection* object's *InfoMessage* event.

Upon examination of the code in the button-click event, you can see that you are going to change the database on the connection to an invalid name, which will raise an error with severity level 11 and cause the *InfoMessage* event to fire. When the event fires, the code in the *InfoMessage* event handler opens a message box displaying the error.

```
' VB
Private Sub GetSqlInfoButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles GetSqlInfoButton.Click
   ConnectionToSql.FireInfoMessageEventOnUserErrors = True
   ConnectionToSql.ChangeDatabase("Northwind1")
End Sub

Private Sub ConnectionToSql_InfoMessage(ByVal sender As Object, _
    ByVal e As System.Data.SqlClient.SqlInfoMessageEventArgs) _
    Handles ConnectionToSql.InfoMessage
   MsgBox(e.Message)
End Sub


// C#
// Add this line of code into the form load handler to hook up the InfoMessage
handler.
ConnectionToSql.InfoMessage += new
    System.Data.SqlClient.SqlInfoMessageEventHandler(this.ConnectionToSql_
InfoMessage);

private void GetSqlInfoButton_Click(object sender, EventArgs e)
{
    ConnectionToSql.FireInfoMessageEventOnUserErrors = true;
    ConnectionToSql.ChangeDatabase("Northwind1");
}

private void ConnectionToSql_InfoMessage(object sender,
    SqlInfoMessageEventArgs e)
{
    MessageBox.Show(e.Message);
}
```

In addition to the previous types of information available from connection objects, you can also return some metadata from the data source you are connected to. In Lesson 1, "Creating and Configuring Connection Objects," we examined the connection properties in the Properties window for the connections available in Server Explorer. This information is available at run time from the connection object as well. As an example, add a few more lines of code to your application and implement the Get Info buttons

of the remaining connections to return the server versions of the data sources they are connected to.

14. Add the following code to the bottom of the form:

```
' VB
Private Sub GetOleDbInfoButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles GetOleDbInfoButton.Click
   MsgBox(ConnectionToOleDb.ServerVersion.ToString, "Server Version")
End Sub

Private Sub GetOdbcInfoButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles GetOdbcInfoButton.Click
   MsgBox(ConnectionToOdbc.ServerVersion.ToString, "Server Version")
End Sub

Private Sub GetOracleInfoButton_Click(ByVal sender As System.Object, _
    ByVal e As System.EventArgs) Handles GetOracleInfoButton.Click
   MsgBox(ConnectionToOracle.ServerVersion.ToString, "Server Version")
End Sub

// C#
private void GetOleDbInfoButton_Click(object sender, EventArgs e)
{
    MessageBox.Show(ConnectionToOleDb.ServerVersion.ToString(), "Server Version");
}

private void GetOdbcInfoButton_Click(object sender, EventArgs e)
{
    MessageBox.Show(ConnectionToOdbc.ServerVersion.ToString(), "Server Version");
}

private void GetOracleInfoButton_Click(object sender, EventArgs e)
{
    MessageBox.Show(ConnectionToOracle.ServerVersion.ToString(), "Server
Version");
}
```

Now let's run the application one more time to check out the additional functionality and verify that the info message and metadata is available from the connection objects.

15. Press F5 to run the application.
16. Click the Open SQL button to open the connection to the SQL server and update the connection-state label.
17. Click the Get SQL Info button to change the database to the invalid *Northwind1* database and raise the *InfoMessage* event that will display in the message box.

IMPORTANT **POSSIBLE INVALID OPERATION EXCEPTION**

The connection must be open or an Invalid Operation exception is thrown and the *InfoMessage* event does not fire.

18. Click the Close SQL button to close the connection to SQL Server and update the connection-state label.
19. Click the Open OLE DB button to open the connection to the OLE DB data source and update the connection-state label.
20. Click the Get OLE DB Info button to retrieve the server version of the OLE DB data source.
21. Click the Close OLE DB button to close the connection and update the connection-state label.
22. Save the application.

IMPORTANT **SAVE THE APPLICATION**

Save this application because you will use it in Lesson 4, "Handling Connection Errors."

Lesson Summary

- Open connections by calling the *Open* method of a connection object.
- Close connections by calling the *Close* method of a connection object.
- Determine whether a connection is opened or closed by monitoring the *StateChanged* event.
- Use the *InfoMessage* event to process any warnings or informational messages that are returned from the server.

Lesson Review

The following questions are intended to reinforce key information presented in this lesson. The questions are also available on the companion CD if you prefer to review them in electronic form.

NOTE **ANSWERS**

Answers to these questions and explanations of why each answer choice is correct or incorrect are located in the "Answers" section at the end of the book.

1. What is the minimal information needed by a connection string to open a connection to a SQL Server 2000 or SQL Server 2005 database? (Choose all that apply.)
   A. A valid data source
   B. A valid provider name
   C. A valid filepath
   D. Appropriate credentials or Integrated Security settings
2. What happens when you call the *Close* method of a connection object? (Choose all that apply.)
   A. The connection is destroyed.
   B. The connection is returned to the connection pool.
   C. The *StateChange* event is fired.
   D. All noncommitted pending transactions are rolled back.
3. What types of information does the *InfoMessage* event typically expose?
   A. Information regarding the current state of a connection
   B. High-severity SQL Server errors (severity 17 and above)
   C. Low-severity SQL Server errors (severity 10 and below)
   D. Network errors that are encountered when attempting to open a connection

Lesson 3: Working with Connection Pools

This lesson explains what connection pooling is and how to control connection pooling options when creating and configuring connection objects.

After this lesson, you will be able to:

- Configure a connection for connection pooling by configuring connection string values.

Estimated lesson time: 30 minutes

What Is Connection Pooling?

Connection pooling allows you to reuse existing connections so you don't have to continuously create and dispose of connections that have the same configuration. In other words, opening and closing connections that use the same connection string and credentials can reuse a connection that is available in the pool. Typical applications use the same connection objects to continuously fetch and update data from a database. Connection pooling provides a much higher level of performance by eliminating the need for the database to constantly create and dispose of connections.

Connection pools are separated by process, application domain, and connection string. For connection strings that use Integrated Security, a separate pool is created for each unique identity.

Controlling Connection Pooling Options

When you create ADO.NET connection objects, connection pooling is enabled by default. You can control connection pooling behavior (or disable pooling altogether) by setting connection string keywords specific to connection pooling. For example, to specifically disable connection pooling, you set Pooling=False in your connection string. Table 5-7 provides a list of connection string keywords that you can use to control how a specific connection interacts with the connection pool. Not all keywords are available for every provider. For example, the OLE DB provider controls connection pooling (also known as resource or session pooling) based on the value set for the OLE DB Services keyword in the connection string.

TABLE 5-7 Connection Pooling Connection String Keywords

NAME	DEFAULT	DESCRIPTION
Connection Lifetime	*0*	When a connection is returned to the pool, if its creation time was longer than *x* seconds ago, with *x* being the value of this property, then the connection is destroyed. Values are in seconds, and a value of *0* indicates the maximum connection timeout.
Connection Reset	*True*	Determines whether the database connection is reset when being drawn from the pool. For SQL Server 7.0, setting to *False* avoids making an additional server round trip when obtaining a connection, but the connection state, such as database context, is not being reset.
Enlist	*True*	If you want to use a connection as part of a transaction, you can set this to *True* and the pooler will automatically enlist the connection in the creation thread's current transaction context.
Load Balance Timeout	*0*	The minimum number of seconds for the connection to live in the connection pool before being destroyed.
Max Pool Size	*100*	The maximum number of connections allowed in the pool for this specific connection string. In other words, if your application continuously connects to the database, you might need to increase the Max Pool Size. For example, if your application has many users who all use the same connection string and you might need more than 100 connections, you would want to increase the *Max Pool Size*. This might happen when many users are accessing the database server using a common client or Web page.
Min Pool Size	*0*	The minimum number of connections allowed in the pool.
Pooling	*True*	When true, the *SqlConnection* object is drawn from the appropriate pool or, if it is required, is created and added to the appropriate pool. Recognized values are *True, False, Yes,* and *No*.

In addition to connection string properties that control connection pooling behavior, there are also methods available on connection objects that can affect the pool as well. You typically use the available methods when you are closing connections in your application and you know they will not be used again. This clears the connection pool by disposing of the connections instead of returning them to the pool when they are closed. Any connections that are already in the pool and open will be disposed of the next time they are closed. Table 5-8 lists the available methods for interacting with connection pools.

TABLE 5-8 Connection Pooling Specific Methods

NAME	OBJECT	DESCRIPTION
ClearAllPools	*SqlConnection and OracleConnection*	Empties all connection pools for a specific provider
ClearPool	*SqlConnection and OracleConnection*	Empties the connection pool associated with the specified connection
ReleaseObjectPool	*OleDbConnection and OdbcConnection*	Indicates that the object pool can be released when the last underlying connection is released

Configuring Connections to Use Connection Pooling

By default, all .NET Framework Data Providers available in ADO.NET have connection pooling turned on, but the level of control available for working with connection pooling varies based on the provider being used.

Configuring Connection Pooling with SQL Server Connections

By default, the *SqlConnection* object automatically uses connection pooling. Each time you call *SqlConnection.Open* with a unique connection string, a new pool is created. You control connection pooling behavior by setting the connection pool keywords in the connection string, as described earlier in Table 5-7. For example, consider a connection where you want to set the minimum pool size. By assigning a value greater than zero to the *Min Pool Size* keyword, you ensure that the pool is not destroyed until after the application ends. To set the minimum pool size to 5, use a connection string similar to the following:

```
Data Source=SqlServerName;Initial Catalog=DatabaseName;
    Integrated Security=True;Min Pool Size=5
```

The minimum pool size is 0 by default, which means that each connection needs to be created and initialized as it is requested. By increasing the minimum pool size in the connection string, the indicated number of connections are created immediately and are then ready to use, which can reduce the time it takes to establish the connection on those initial connections.

Configuring Connection Pooling with OLE DB Connections

The OLE DB connection object (*OleDbConnection*) automatically pools connections through the use of OLE DB session pooling. You control how OLE DB connections use pooling by adding an OLE DB Services keyword to the connection string and setting its value based on the combination of services you want to enable or disable for the connection.

The following connection strings explicitly enable connection pooling by setting the *OLE DB Services* keyword to -1.

OLE DB connection string for an Office Access database (assumes the Nwind.mdb file exists in the following path: C:\DataSources\Nwind.mdb):

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\DataSources\Nwind.mdb;
    OLE DB Services=-1
```

OLE DB Connection for a SQL Server database (replace ServerName and DatabaseName with valid values for your data source):

```
Provider=SQLOLEDB;Data Source=ServerName;OLE DB Services=-1;
    Integrated Security=SSPI;Initial Catalog=DatabaseName
```

The following connection strings disable connection pooling and automatic transaction enlistment by setting the *OLE DB Services* keyword to -4:

```
Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\DataSources\Nwind.mdb;OLE DB Services=-4
```

Table 5-9 lists the OLE DB Services values to set in an OLE DB connection string.

TABLE 5-9 OLE DB Connection String Settings for OLE DB Services

OLE DB SERVICE	CONNECTION STRING KEYWORD/VALUE
All services on	"OLE DB Services = -1;"
All services except *Pooling* and *AutoEnlistment* of transactions	"OLE DB Services = -4;"
All services except *Client Cursor*	"OLE DB Services = -5;"
All services except *Pooling, AutoEnlistment,* and *Client Cursor*	"OLE DB Services = -8;"
No services (all services disabled)	"OLE DB Services = 0;"

Configuring Connection Pooling with ODBC Connections

To enable or disable connection pooling for connections that use the ODBC connection object (*OdbcConnection*), you must use the ODBC Data Source Administrator dialog box in Windows.

ACCESSING THE ODBC DATA SOURCE ADMINISTRATOR DIALOG BOX

Access the ODBC Data Source Administrator dialog box by performing the following steps:

1. In the Administrative Tools folder on your Start menu, open Data Sources (ODBC).
2. Click the Connection Pooling tab.
3. Double-click the driver from the list of available ODBC drivers that you want to set connection pooling options for.

4. In the Set Connection Pooling Attributes dialog box, select the option to either pool connections or not pool connections. If you select the option to pool connections, you can also set the number of seconds for unused connections to remain in the pool (the connection lifetime).
5. Click OK to save the settings and repeat for other drivers if desired.

> ***IMPORTANT*** **ODBC SETTINGS**
>
> **The settings for a particular ODBC driver are in effect for all applications/connections that use that particular driver.**

Configuring Connection Pooling with Oracle Connections

Connections that use the .NET Framework Data Provider for Oracle automatically use connection pooling by default. You can control how the connection uses pooling by setting connection string keywords.

Table 5-10 describes the connection string keywords available for altering connection pooling activities.

TABLE 5-10 Oracle Connection String Settings for Connection Pooling

NAME	DEFAULT	DESCRIPTION
Connection Lifetime	*0*	When a connection is returned to the pool, its creation time is compared with the current time and the connection is destroyed if that time span exceeds the value specified. Values are in seconds and a value of *0* indicates the maximum connection timeout.
Enlist	*True*	When true, the pooler automatically enlists the connection in the creation thread's current transaction context. Recognized values are *True, False, Yes,* and *No*.
Max Pool Size	*100*	The maximum number of connections allowed in the pool.
Min Pool Size	*0*	The minimum number of connections allowed in the pool.
Pooling	*True*	When true, the *OracleConnection* object is drawn from the appropriate pool or, if it is required, is created and added to the appropriate pool.

Lesson Summary

- Connection pooling is enabled by default.
- Connection pooling options are set in the connection string except for the ODBC provider, which uses the ODBC Data Source Administrator dialog box in Windows.

Lesson Review

The following questions are intended to reinforce key information presented in this lesson. The questions are also available on the companion CD if you prefer to review them in electronic form.

> ***NOTE*** **ANSWERS**
>
> **Answers to these questions and explanations of why each answer choice is correct or incorrect are located in the "Answers" section at the end of this book.**

1. What determines the connection pool that a connection should use? (Choose all that apply.)
   - **A.** A connection string
   - **B.** The identity or credentials of the user opening the connection
   - **C.** The database being connected to
   - **D.** The connection object used to connect to the database
2. What are the recommended techniques for enabling connection pooling on for a SQL Server 2000 or SQL Server 2005 database? (Choose all that apply.)
   - **A.** Setting the *OLE DB Services* connection string keyword to -4
   - **B.** Opening a connection and not explicitly disabling pooling
   - **C.** Setting the connection string keyword *Pooling = True* in the connection string
   - **D.** Using the Connection Pooling tab of the ODBC Data Source Administrator dialog box
3. How do I explicitly turn on connection pooling for an OLE DB data source?
   - **A.** By setting the *OLE DB Services* connection string keyword to *0*
   - **B.** By setting the *OLE DB Services* connection string keyword to *-4*
   - **C.** By setting the *OLE DB Services* connection string keyword to *-1*
   - **D.** By setting the *OLE DB Services* connection string keyword to *-7*

Lesson 4: Handling Connection Errors

This lesson explains how to handle errors that are thrown while you are working with SQL Server. ADO.NET provides two classes specifically for processing errors: the *SqlException* class and the *SqlError* class. Let's see how to work with these classes and how to catch and handle errors that might be returned from the data source.

After this lesson, you will be able to:

- Handle exceptions when connecting to a database.
- Use the *SqlException* class to detect connection errors.
- Use the *SqlError* class to detect connection errors.

Estimated lesson time: 20 minutes

When SQL Server returns a warning or an error, the .NET Framework Data Provider for SQL Server creates and throws a *SqlException* that you can catch in your application to deal with the problem. When *SqlException* is thrown, inspect the *SqlException.Errors* property to access the collection of errors that is returned from the SQL server. The *SqlException.Errors* property is a *SqlErrorCollection* class (a collection of *SqlError* classes) that always contains at least one *SqlError* object.

MORE INFO **SQL SERVER ERRORS**

SqlConnection remains open for messages with a severity level of 19 and below, but it typically closes automatically when the severity is 20 or greater.

LAB Handling Database Connection Errors

In this lab you will practice catching a *SqlException* in your application.

EXERCISE 1 Handling Database Connection Errors

In this lab you will practice working with database connection errors (specifically, the *SqlException* and *SqlError* objects) in your application. To do this, let's create a Windows application.

1. Create a new Windows application and name it **HandlingConnectionErrors**.
2. Add three buttons to the form and set the following properties:

   Button1:

   - Name = *GoodConnectButton*
   - Text = Connect (valid connection string)

Button2:

- Name = *ConnectToInvalidUserButton*
- Text = Connect to invalid user

Button3:

- Name = *ConnectToInvalidDatabaseButton*
- Text = Connect to invalid database

3. Double-click each button to create the *button click* event handlers and switch to code view.
4. Add an *Imports* statement (*using* in C#) for the *System.Data.SqlClient* namespace.
5. The following code creates a new connection based on the connection string passed into it, attempts to open the connection, and then displays any errors it encounters. Add this code below the *button click* event handlers:

```
' VB
Private Sub ConnectToDatabase(ByVal connectionString As String)

    Dim connection As New SqlConnection(connectionString)

    Try
        connection.Open()
    Catch ex As SqlException
        Dim errorMessage As String = ""

        ' Iterate through all errors returned
        ' You can check the error numbers to handle specific errors
        For Each ConnectionError As SqlError In ex.Errors
            errorMessage += ConnectionError.Message & " (error: " & _
                ConnectionError.Number.ToString & ")" & Environment.NewLine
            If ConnectionError.Number = 18452 Then
                MessageBox.Show( _
                    "Invalid Login Detected, please provide valid credentials!")
            End If
        Next
        MessageBox.Show(errorMessage)
    Finally
        connection.Close()
    End Try
End Sub

// C#
private void ConnectToDatabase(string connectionString)
{
    SqlConnection connection = new SqlConnection(connectionString);
```

```
    try
    {
        connection.Open();
    }
    catch (SqlException ex)
    {
        string errorMessage = "";
        // Iterate through all errors returned
        // You can check the error numbers to handle specific errors
        foreach (SqlError ConnectionError in ex.Errors)
        {
            errorMessage += ConnectionError.Message + " (error: " +
                ConnectionError.Number.ToString() + ")" + Environment.NewLine;
            if (ConnectionError.Number == 18452)
            {
                MessageBox.Show(
                    "Invalid Login Detected, please provide valid credentials!");
            }
        }
        MessageBox.Show(errorMessage);
    }
    finally
    {
        connection.Close();
    }
}
```

6. Add the following code so the three *button click* event handlers look like the following:

```
' VB
Private Sub GoodConnectButton_Click _
    (ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles GoodConnectButton.Click
    ' This is a valid connection string
    Dim GoodConnection As String = _
        "Data Source=.\sqlexpress;Initial Catalog=Northwind;Integrated
Security=True;"
    ConnectToDatabase(GoodConnection)
End Sub

Private Sub ConnectToInvalidUserButton_Click _
    (ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles ConnectToInvalidUserButton.Click
    ' This connection string has invalid credentials
    Dim InvalidUserConnection As String = _
```

```
        "Data Source=.\sqlexpress;Initial Catalog=Northwind;User ID = InvalidUser"
    ConnectToDatabase(InvalidUserConnection)
End Sub

Private Sub ConnectToInvalidDatabaseButton_Click _
    (ByVal sender As System.Object, ByVal e As System.EventArgs) _
    Handles ConnectToInvalidDatabaseButton.Click
    ' This connection string has an invalid/unavailable database
    Dim InvalidDatabaseConnection As String = _
        "Data Source=.\sqlexpress;Initial Catalog=InvalidDatabase;" & _
        "Integrated Security=True"
    ConnectToDatabase(InvalidDatabaseConnection)
End Sub

// C#
private void GoodConnectButton_Click(object sender, EventArgs e)
{
    // This is a valid connection string
    String GoodConnection =
        "Data Source=.\\sqlexpress;Initial Catalog=Northwind;Integrated
Security=True";
    ConnectToDatabase(GoodConnection);
}

private void ConnectToInvalidUserButton_Click(object sender, EventArgs e)
{
    // This connection string has invalid credentials
    String InvalidUserConnection =
        "Data Source=.\\sqlexpress;Initial Catalog=Northwind;User ID =
InvalidUser";
    ConnectToDatabase(InvalidUserConnection);
}

private void ConnectToInvalidDatabaseButton_Click(object sender, EventArgs e)
{
    // This connection string has an invalid/unavailable database
    String InvalidDatabaseConnection =
        "Data Source=.\\sqlexpress;Initial Catalog=InvalidDatabase;" +
        "Integrated Security=True";
    ConnectToDatabase(InvalidDatabaseConnection);
}
```

7. Run the application.
8. Click the Connect button. No errors should be raised.

9. Click the Connect To Invalid User button. The code to catch the specific login error (error 18452) is executed.
10. Click the Connect To Invalid Database button. You can see that an error was raised and is displayed in the message box.

Lesson Summary

- A *SqlException* object is created when an error is detected on the SQL server.
- Every instance of a *SqlException* exception contains at least one SqlError warning that contains the actual error information from the server.

Lesson Review

The following questions are intended to reinforce key information presented in this lesson. The questions are also available on the companion CD if you prefer to review them in electronic form.

> ***NOTE*** **ANSWERS**
>
> **Answers to these questions and explanations of why each answer choice is correct or incorrect are located in the "Answers" section at the end of the book.**

1. What types of errors will cause a *SqlConnection* object to close? (Choose all that apply.)
   A. Errors wth a severity level of 1 through 9
   B. Errors wth a severity level of 10 through 19
   C. Errors wth a severity level of 20 through 29
   D. Errors wth a severity level of 30 or greater
2. What property contains the actual error message returned by SQL Server? (Choose all that apply.)
   A. *SqlException.Source*
   B. *SqlException.Message*
   C. *SqlError.Class*
   D. *SqlError.Message*

Lesson 5: Enumerating the Available SQL Servers on a Network

This lesson describes how to return a list of visible SQL Server instances on a network comparable to the Server Name drop-down list in the Add Connection dialog box.

After this lesson, you will be able to:

- Enumerate through instances of SQL Server.

Estimated lesson time: 20 minutes

The .NET Framework offers applications a way to discover SQL Server instances on a network so your programs can process this information when necessary. To retrieve the list of available SQL Servers, use the *Instance* property of the *SqlDataSourceEnumerator* class and call the *GetDataSources* method. The *GetDataSources* method returns a DataTable that contains information for each SQL server that is visible on the network. The returned data table contains the columns listed in Table 5-11.

TABLE 5-11 *DataTable* Schema Returned by the *GetDataSources* Method

COLUMN NAME	DESCRIPTION
ServerName	Name of the SQL server containing the visible instance
InstanceName	Name of the server instance, or empty for servers running default instances
IsClustered	Indicates whether the server is part of a cluster
Version	The version number of the SQL server

Why Do Only Some or No SQL Servers Appear in My Grid?

Depending on how your network or even your individual machine is set up, the list of available servers might or might not be complete. In addition to things such as network traffic and timeout issues, the way your network implements security can cause servers to be hidden from the returned list as well. If you are running SQL Server 2005, a service named SQL Browser needs to be running for you to see SQL Server instances. And even if your SQL Browser service is running, your firewall might be blocking the request for SQL information. The firewall is likely to be blocking communication requests through port 1433, which is the

default port that SQL Server default instances are set up to use. There are obvious security implications concerning turning on the SQL Browser service as well as enabling communications through specific ports through your firewall, but these are beyond the scope of this book. A good resource is the "SQL Browser Service" section of SQL Server 2005 Books Online, and I encourage you to read that before changing any settings on your firewall or SQL Server configuration.

LAB Returning the List of Visible SQL Servers

In this lab you will practice enumerating the SQL Servers on your network.

EXERCISE 1 Enumerating the SQL Servers on a Network

To demonstrate how to retrieve the list of visible SQL servers, let's create a small application to display the information returned from the *GetDataSources* method in a *DataGridView*.

1. Create a new Windows application named **SqlServerEnumerator**.
2. Add a *DataGridView* to the form and name it **VisibleSqlServers**.

> ***NOTE DATAGRIDVIEW***
>
> **The *DataGridView* is the control typically used for displaying data. The *DataGridView* is discussed in more detail in Chapter 8.**

3. Add a Button control below the grid and set its *Name* property to **GetDataSources-Button**.
4. Set the Button's *Text* property to **Get Visible Servers**.
5. Double-click the Get Visible Servers button to create the *Click* handler and switch to code view.
6. Add code so that the handler looks like the following:

```
' VB
Dim instance As System.Data.Sql.SqlDataSourceEnumerator = _
    System.Data.Sql.SqlDataSourceEnumerator.Instance
VisibleSqlServers.DataSource = instance.GetDataSources

// C#
System.Data.Sql.SqlDataSourceEnumerator instance =
    System.Data.Sql.SqlDataSourceEnumerator.Instance;
VisibleSqlServers.DataSource = instance.GetDataSources();
```

Now run the application and click the Get Visible Servers button. All visible SQL servers on your network appear in the grid, looking similar to Figure 5-3.

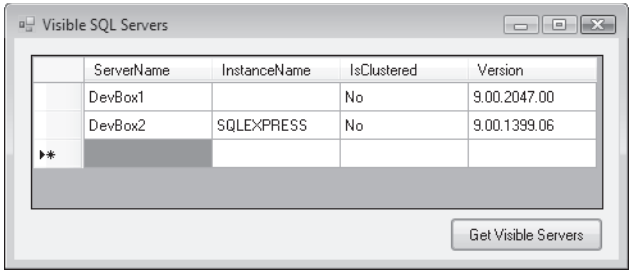


FIGURE 5-3 Grid showing all visible SQL servers on your network

Lesson Summary

- You can use the *SqlDataSourceEnumerator* object to return a list of visible SQL servers on a network.
- The list of servers returned might not be complete, due to factors such as firewall settings and protocol configurations on the SQL Server services.

Lesson Review

The following questions are intended to reinforce key information presented in this lesson. The questions are also available on the companion CD if you prefer to review them in electronic form.

> ***NOTE*** **ANSWERS**
>
> **Answers to these questions and explanations of why each answer choice is correct or incorrect are located in the "Answers" section at the end of this book.**

1. What object is used to return the list of visible SQL Servers?
   - **A.** *VisibleSqlServers*
   - **B.** *GetDataSources*
   - **C.** *SqlDataSourceEnumerator*
   - **D.** *ServerName*
2. What factors can cause SQL servers to be invisible on the network? (Choose all that apply.)
   - **A.** The computer's firewall settings
   - **B.** The amount of network traffic
   - **C.** The availability of the SQL Browser service
   - **D.** The *Visibility* property of the SQL Server

3. Which of the following pieces of information is available through the *SqlServerEnumerator* object? (Choose all that apply.)

   A. The name of the SQL server

   B. The number of databases currently on the server

   C. The version number of the server

   D. The instance name for servers that are not running default instances

Lesson 6: Securing Sensitive Connection String Data

Because of the sensitive nature of most data in real-world scenarios, it is extremely important to protect your servers and databases from unauthorized access. To ensure limited access to your data source, it is a best practice to secure information like user IDs, data source names, and, of course, passwords. Storing this type of information as plain text is not recommended because of the obvious security risk. It is also worth noting that plain text saved in compiled applications is easily decompiled, rendering your data accessible by persons with questionable intent.

After this lesson, you will able to:

- Protect access to a data source's connection details.

Estimated lesson time: 45 minutes

REAL WORLD

Steve Stein

In another of my previous jobs (okay, I've had a few!), I took a position as a system administrator for a local mortgage company. My first task was to get familiar with the infrastructure of their company network. I immediately realized that basically every employee was set up with an administrator account and had access to the entire network. Although this story isn't specific to securing connection strings, it does provide insight into how important it is to lock down your sensitive data!

The suggested method of implementing security in applications that access data is to use Windows Authentication (also known as Integrated Security). To further protect sensitive connection information when using Integrated Security, it is also recommended that you set the *Persist Security Information* keyword to *False* in the connection string. This ensures that the credentials used to open the connection are discarded and not stored where someone might be able to retrieve them.

Table 5-12 provides the key/value pairs to set in the connection string for implementing Integrated Security in the four .NET Framework Data Providers.

TABLE 5-12 Connection String Keywords for Turning On Integrated Security

DATA PROVIDER	KEY/VALUE PAIR
SqlClient	Integrated Security=True
SqlClient and OleDb	Integrated Security=SSPI
Odbc	Trusted_Connection=Yes
OracleClient	Integrated Security=Yes

As stated earlier, if you absolutely must use a connection string that contains sensitive information, do not store the connection string in the compiled application. As an alternative, you can use the application configuration file (app.config). The app.config file stores connection strings as Extensible Markup Language (XML), and your application gets its connection information by querying this file at run time (as opposed to compiling the connection string into the application itself). By default, the application configuration file stores its information unencrypted, as shown in Figure 5-4.

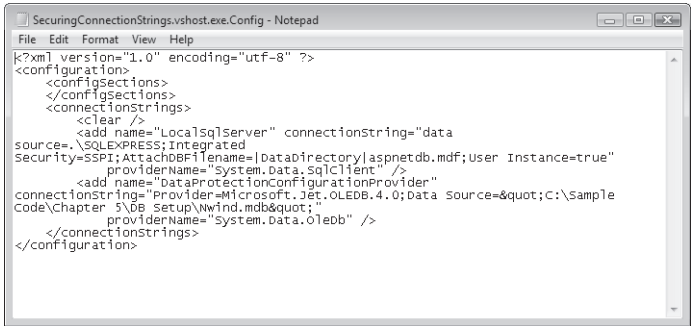


FIGURE 5-4 An unencrypted configuration file

Securing Data in Configuration Files

Now that you've moved your sensitive connection string data out of the compiled application and into the application's configuration file, the connection string is still unencrypted and can be read by anyone with permission to open the configuration file. Therefore, you still need a way to prevent unauthorized personnel from viewing the connection information if they somehow gain access to your configuration file. The suggested method of securing configuration files is to encrypt the sections that contain sensitive information, as shown in Figure 5-5.

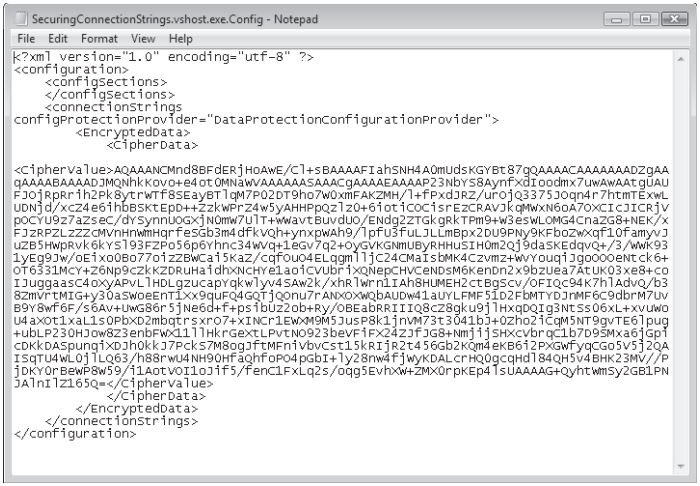


FIGURE 5-5 An encrypted configuration file

The suggested approach to encrypting configuration data is to use a protected-configuration provider. Two protected-configuration providers are available in the .NET Framework, as well as a base class that you can use to implement your own if the two available providers are not sufficient for your application.

LAB Securing a Configuration File

In this lab you will practice encrypting and decrypting a configuration file.

EXERCISE 1 Encrypting and Decrypting a Configuration File

In this lesson you will see how to use the DpapiProtectedConfigurationProvider to encrypt and decrypt the ConnectionStrings section of the app.config file.

1. Create a new Windows Application and name it **SecuringConnectionStrings.**
2. Add a reference to the *System.Configuration* namespace.
3. Add two buttons to the form, setting the *Name* and *Text* properties to the following:

NAME PROPERTY	TEXT PROPERTY
EncryptButton	Encrypt
DecryptButton	Decrypt

4. Create a data source and add a connection string to the application configuration file by running the Data Source Configuration Wizard.
5. Create event handlers for the button-click events.

6. Switch to code view and paste the following code into the editor:

   The following code locates the connection string setting in the application's configuration file. The connection string setting is marked for encryption by calling the *ProtectSection* method. Setting the *ForceSave* property to *True* ensures the configuration file is saved whether changes are made or not; and the *Configuration.Save* call saves the file once it has been encrypted.

```
' VB
Imports System
Imports System.Configuration

Public Class Form1
    Private Sub EncryptConnectionString()
        ' Get the configuration file
        Dim config As System.Configuration.Configuration = _
            ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)

        ' Create the provider name
        Dim provider As String = _
            "DataProtectionConfigurationProvider"

        ' Encrypt the ConnectionStrings
        Dim connStrings As ConfigurationSection = _
            config.ConnectionStrings
        connStrings.SectionInformation.ProtectSection(provider)
        connStrings.SectionInformation.ForceSave = True
        config.Save(ConfigurationSaveMode.Full)
    End Sub

    Private Sub DecryptConnectionString()
        ' Get the configuration file
        Dim config As System.Configuration.Configuration = _
            ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.None)

        ' Decrypt the ConnectionStrings
        Dim connStrings As ConfigurationSection = _
            config.ConnectionStrings
        connStrings.SectionInformation.UnprotectSection()
        connStrings.SectionInformation.ForceSave = True
        config.Save(ConfigurationSaveMode.Full)
    End Sub

    Private Sub EncryptButton_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles EncryptButton.Click
        EncryptConnectionString()
    End Sub
```

```
    Private Sub DecryptButton_Click(ByVal sender As System.Object, _
        ByVal e As System.EventArgs) Handles DecryptButton.Click
        DecryptConnectionString()
    End Sub
End Class

// C#
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Configuration;

namespace SecuringConnectionStrings
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void EncryptConnectionString()
        {
            // Get the configuration file
            System.Configuration.Configuration config =
                ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.
None);

            // Create the provider name
            string provider = "DataProtectionConfigurationProvider";

            //Encrypt the connectionStrings
            ConfigurationSection connstrings = config.ConnectionStrings;
            connstrings.SectionInformation.ProtectSection(provider);
            connstrings.SectionInformation.ForceSave = true;
            config.Save(ConfigurationSaveMode.Full);
        }

        private void DecryptConnectionString()
        {
            //Get the configuration file
            System.Configuration.Configuration config =
```

```
                ConfigurationManager.OpenExeConfiguration(ConfigurationUserLevel.
None);

            // Decrypt the connectionStrings
            ConfigurationSection connstrings = config.ConnectionStrings;
            connstrings.SectionInformation.UnprotectSection();
            connstrings.SectionInformation.ForceSave = true;
            config.Save(ConfigurationSaveMode.Full);
        }

        private void EncryptButton_Click(object sender, EventArgs e)
        {
            EncryptConnectionString();
        }

        private void DecryptButton_Click(object sender, EventArgs e)
        {
            DecryptConnectionString();
        }
    }
}
```

7. Run the application and click the Encrypt button.
8. While the application is running, navigate to the project's folder and locate the configuration file (SecuringConnectionStrings.vshost.exe.config).
9. Open the file and verify that the ConnectionStrings section is encrypted.
10. Go back to the form and click the Decrypt button.
11. Reopen the .config file and notice that the connection string has reverted back to plain text.

Lesson Summary

- Windows Authentication (also called Integrated Security) is the suggested method for connecting to data securely.
- Store connection strings that contain sensitive information in the application configuration file and encrypt all settings that contain confidential information.

Lesson Review

The following questions are intended to reinforce key information presented in this lesson. The questions are also available on the companion CD if you prefer to review them in electronic form.

> ***NOTE*** **ANSWERS**
>
> **Answers to these questions and explanations of why each answer choice is correct or incorrect are located in the "Answers" section at the end of this book.**

1. What is the connection string's key/value pair for using Windows Authentication in SQL Server 2000 and SQL Server 2005? (Choose all that apply.)
   - **A.** Integrated Security = yes
   - **B.** Integrated Security =SSPI
   - **C.** Integrated Security = True
   - **D.** Trusted_Connection = Yes
2. If you must use a user name and password to connect to a database, where should you store the sensitive information?
   - **A.** Compiled in the application
   - **B.** In an encrypted application configuration file
   - **C.** In a resource file deployed with the application
   - **D.** In the registry
3. What is the recommended method for securing sensitive connection string information?
   - **A.** Encrypting the data in the application configuration file
   - **B.** Using a code obfuscator
   - **C.** Using Integrated Security (Windows Authentication)
   - **D.** Querying the user for his or her credentials at run time

Chapter Review

To further practice and reinforce the skills you learned in this chapter, you can perform the following tasks:

- Review the chapter summary.
- Complete the case scenarios. These scenarios set up real-world situations involving the topics of this chapter and ask you to create a solution.
- Complete the additional practices.
- Take a practice test.

Chapter Summary

- You create connection objects by setting a valid connection string and enabling communication between your application and a data source. ADO.NET provides four primary connection objects that you can use to connect to almost any standard database.
- Connection objects contain several properties, methods, and events that are used for opening and closing connections to a data source, providing information on the current state of the connection and surfacing warnings and informational messages from a data source.
- Connection objects enable connection pooling by default. By setting connection–pooling specific connection string keywords, you can control how connections interact with the connection pool.
- By wrapping connection calls in a *try-catch* block, you can process errors returned from SQL Server by using the *SqlException* and *SqlError* classes.
- By using Windows Authentication and application configuration files, you can protect sensitive information such as passwords in your programs.

Key Terms

Do you know what these key terms mean? You can check your answers by looking up the terms in the glossary at the end of the book.

- connection object
- connection pool
- connection string
- encryption
- Integrated Security

Case Scenarios

In the following case scenarios, you will apply what you've learned about configuring connections and connecting to data. You can find answers to these questions in the "Answers" section at the end of this book.

Case Scenario 1: Troubleshooting a SQL Connection

You just landed a sweet job at the Alpine Ski House and have been assigned to maintain the application that keeps track of inventory in the ski rental hut. The client application connects to a SQL Server database where the inventory data is stored. You decide to test the application before the season begins, and the first time you run the application and try to check inventory you get an unhandled exception originating from the SQL server.

How can you modify the application so that users can better identify and troubleshoot connection problems?

Case Scenario 2: Securing Sensitive Data

You are working as an application developer at Contoso Pharmaceuticals and have been asked to rewrite their in-house research and development application. The first thing you notice is that they store user name and password information in plain text within the application code base.

Create a list of suggested remedies to present to upper management.

Suggested Practices

To gain further knowledge on the subject of working with connections, complete the following practices.

- **Practice 1** Create an application that targets different databases, which can be selected when the application starts.
- **Practice 2** Design a reusable block of code that can be used to handle SQL Server errors of any severity.
- **Practice 3** Create a component that writes to a log every time a connection to a database is opened.

Take a Practice Test

The practice tests on this book's companion CD offer many options. For example, you can test yourself on just the content covered in this chapter, or you can test yourself on all the 70-505 certification exam content. You can set up the test so that it closely simulates the experience of taking a certification exam, or you can set it up in study mode so that you can look at the correct answers and explanations after you answer each question.

MORE INFO **PRACTICE TESTS**

For details about all the practice test options available, see the "How to Use the Practice Tests" section in this book's Introduction.