Files

Visual Basic 2010 How to Program

OBJECTIVES

In this chapter you'll learn:

- To use file processing to implement a business application.
- To create, write to and read from files.
- To become familiar with sequential-access file processing.
- To use classes `StreamWriter` and `StreamReader` to write text to and read text from files.
- To organize GUI commands in menus.
- To manage resources with `Using` statements and the Finally block of a `Try` statement.

8.1 Introduction

- Variables and arrays offer only temporary storage of data in memory—the data is lost, for example, when a local variable "goes out of scope" or when the program terminates.
- By contrast, files (and databases, which we cover in Chapter 12) are used for long-term retention of large (and often vast) amounts of data, even after the program that created the data terminates, so data maintained in files is often called persistent data.
- Computers store files on secondary storage devices, such as magnetic disks, optical disks (like CDs, DVDs and Blu-ray Discs™), USB flash drives and magnetic tapes.

8.1 Introduction

- In this chapter, we explain how to create, write to and read from data files.
- We continue our treatment of GUIs, explaining how to organize commands in menus, and showing how to use the Windows Forms Designer to rapidly create menus.
- We also discuss resource management.
- As programs execute, they often acquire resources, such as memory and files, that need to be returned to the system so they can be reused at a later point.
- We show how to ensure that resources are properly returned to the system when they're no longer needed.

8.2 Data Hierarchy

- Ultimately, all data items that computers process are reduced to combinations of 0s and 1s.
- This occurs because it's simple and economical to build electronic devices that can assume two stable states—one represents 0 and the other represents 1.
- It's remarkable that the impressive functions performed by computers involve only the most fundamental manipulations of 0s and 1s!

8.2 Data Hierarchy

- Bits
  - The smallest data item that computers support is called a bit, short for "binary digit"—a digit that can assume either the value 0 or the value 1.
  - Computer circuitry performs various simple bit manipulations, such as examining the value of a bit, setting the value of a bit and reversing a bit (from 1 to 0 or from 0 to 1).
  - For more information on the binary number system, see Appendix C, Number Systems.

8.2 Data Hierarchy

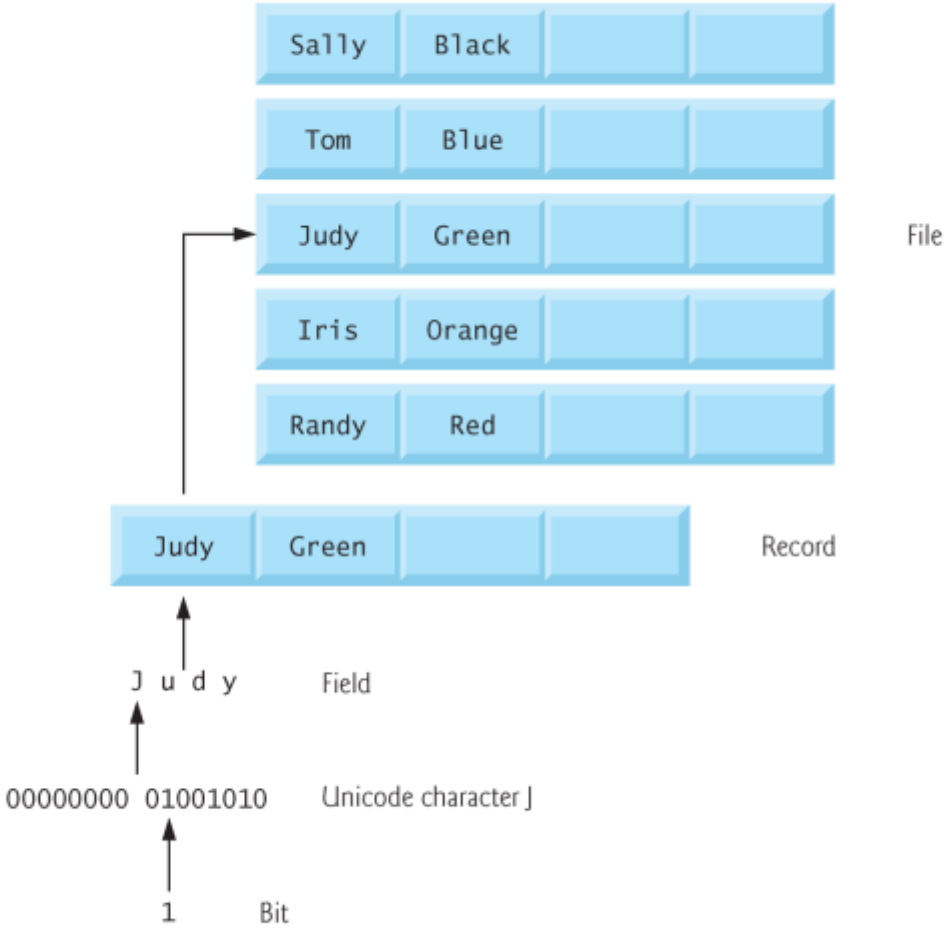
- Characters
  - Programming with data in the low-level form of bits is cumbersome.
  - It's preferable to program with data in forms such as decimal digits (that is, 0, 1, 2, 3, 4, 5, 6, 7, 8 and 9), letters (that is, the uppercase letters A–Z and the lowercase letters a–z) and special symbols (that is, $, @, %, &, *, (,), -, +, ", :, ?, / and many others).
  - Digits, letters and special symbols are referred to as characters.

8.2 Data Hierarchy

- The set of all characters used to write programs and represent data items on a particular computer is called that computer's character set.
- Every character in a computer's character set is represented as a pattern of 0s and 1s.
- Bytes are composed of eight bits.
- Visual Basic uses the Unicode character set, in which each character is composed of two bytes (and hence 16 bits).
- You create programs and data items with characters; computers manipulate and process these characters as patterns of bits.

8.2 Data Hierarchy

- Fields
  - Just as characters are composed of bits, fields are composed of characters.
  - A field is a group of characters that conveys meaning.
  - For example, a field consisting of uppercase and lowercase letters can represent a person's name.
- Data Hierarchy
  - Data items processed by computers form a data hierarchy (Fig. 8.1), in which data items become larger and more complex in structure as we progress up the hierarchy from bits to characters to fields to larger data aggregates.



8.2 Data Hierarchy

- ***Records***
  - Typically, a record is composed of several related fields.
  - In a payroll system, for example, a record for a particular employee might include the following fields:
    - Employee identification number
    - Name
    - Address
    - Hourly pay rate
    - Number of exemptions claimed
    - Year-to-date earnings
    - Amount of taxes withheld

8.2 Data Hierarchy

- In the preceding example, each field is associated with the same employee.
- A data file can be implemented as a group of related records.
- A company's payroll file normally contains one record for each employee.
- Companies typically have many files, some containing millions, billions or even trillions of characters of information.
- To facilitate the retrieval of specific records from a file, at least one field in each record can be chosen as a record key, which identifies a record as belonging to a particular person or entity and distinguishes that record from all others.
- For example, in a payroll record, the employee identification number normally would be the record key.

8.2 Data Hierarchy

- Sequential Files
  - There are many ways to organize records in a file.
  - A common organization is called a sequential file in which records typically are stored in order by a record-key field.
  - In a payroll file, records usually are placed in order by employee identification number.

8.2 Data Hierarchy

- Databases
  - Most businesses use many different files to store data.
  - For example, a company might have payroll files, accounts receivable files (listing money due from clients), accounts payable files (listing money due to suppliers), inventory files (listing facts about all the items handled by the business) and many other files.
  - Related files often are stored in a database.
  - A collection of programs designed to create and manage databases is called a database management system (DBMS).
  - You'll learn about databases in Chapter 12 and you'll do additional work with databases in Chapter 13, Web App Development with ASP.NET, and the online Web Services chapter.

8.3 Files and Streams

- Visual Basic views a file simply as a sequential stream of bytes (Fig. 8.2).
- Depending on the operating system, each file ends either with an end-of-file marker or at a specific byte number that's recorded in a system-maintained administrative data structure for the file.
- For example, the Windows operating system keeps track of the number of bytes in a file.
- You open a file from a Visual Basic program by creating an object that enables communication between a program and a particular file, such as an object of class `StreamWriter` to write text to a file or an object of class `StreamReader` to read text from a file.

Fig. 8.2 | Visual Basic's view of an *n*-byte file.

8.4 Test-Driving the Credit Inquiry Application

- A credit manager would like you to implement a Credit Inquiry application that enables the credit manager to separately search for and display account information for customers with
  - debit balances—customers who owe the company money for previously received goods and services
  - zero balances—customers who do not owe the company money
  - credit balances—customers to whom the company owes money
- The application reads records from a text file then displays the contents of each record that matches the type selected by the credit manager, whom we shall refer to from this point forward simply as "the user."

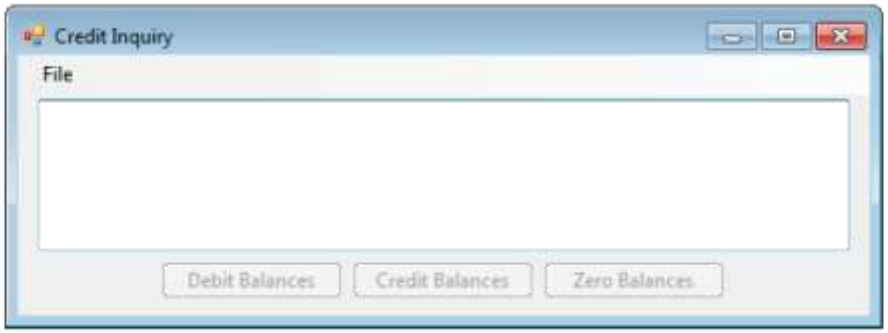
8.4 Test-Driving the Credit Inquiry Application

- Opening the File
  - When the user initially executes the Credit Inquiry application, the `Button`s at the bottom of the window are disabled (Fig. 8.3(a))—the user cannot interact with them until a file has been selected.
  - The company could have several files containing account data, so to begin processing a file of accounts, the user selects Open… from the application's custom File menu (Fig. 8.3(b)), which you'll create in Section 8.6.
  - This displays an Open dialog (Fig. 8.3(c)) that allows the user to specify the name and location of the file from which the records will be read.

8.4 Test-Driving the Credit Inquiry Application

- In our case, we stored the file in the folder `C:\DataFiles` and named the file `Accounts.txt`.
- The left side of the dialog allows the user to locate the file on disk.
- The user can then select the file in the right side of the dialog and click the `Open Button` to submit the file name to the application.
- The `File` menu also provides an `Exit` menu item that allows the user to terminate the application.

a) Initial GUI with `Buttons` disabled until the user selects a file from which to read records



b) Selecting the **Open...** menu item from the **File** menu displays the **Open** dialog in part (c)

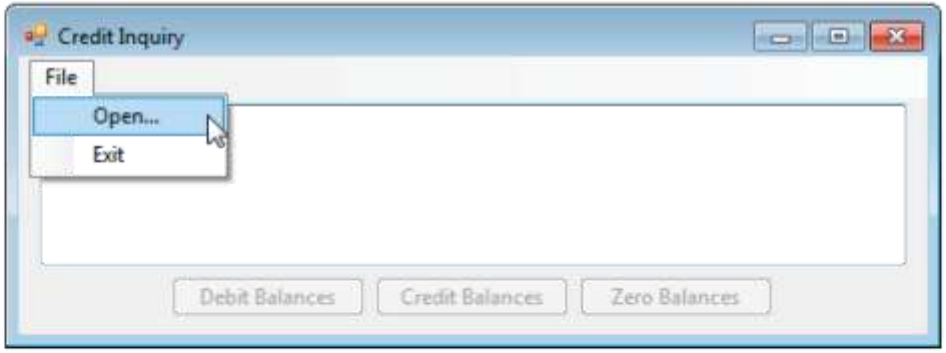


Fig. 8.3 | GUI for the Credit Inquiry application. (Part 1 of 2.)

c) The **Open** dialog allows the user to specify the location and name of the file

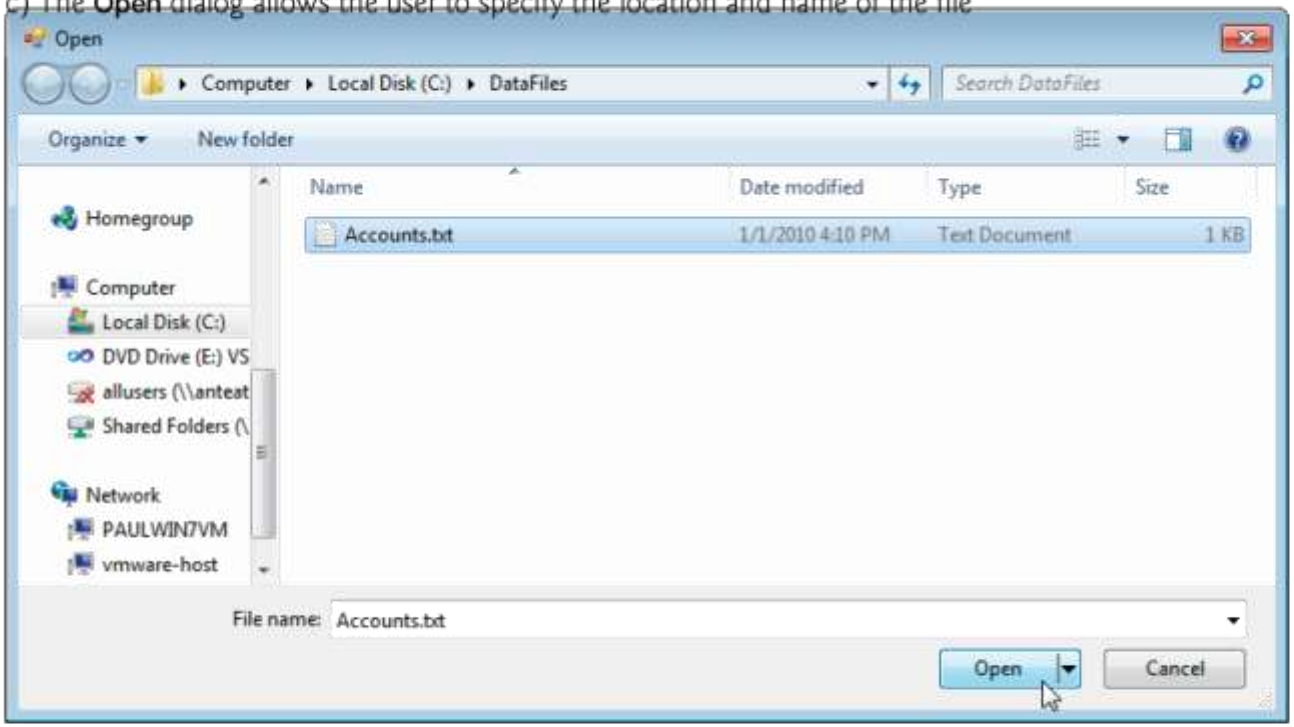


Fig. 8.3 | GUI for the Credit Inquiry application. (Part 2 of 2.)

8.4 Test-Driving the Credit Inquiry Application

- Displaying Accounts with Credit, Debit and Zero Balances
  - After selecting a file name, the user can click one of the `Button`s at the bottom of the window to display the records that match the specified account type.
  - Figure 8.4(a) shows the accounts with debit balances.
  - Figure 8.4(b) shows the accounts with credit balances.
  - Figure 8.4(c) shows the accounts with zero balances.

a) Clicking the **Debit Balances** `Button` displays the accounts with *positive* balances (that is, the people who owe the company money)

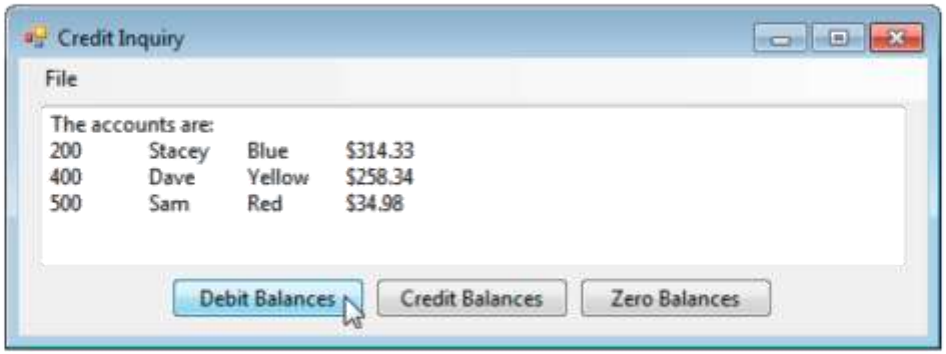


Fig. 8.4 | GUI for Credit Inquiry application. (Part 1 of 3.)

b) Clicking the **Credit Balances** `Button` displays the accounts with *negative* balances (that is, the people to whom the company owes money)

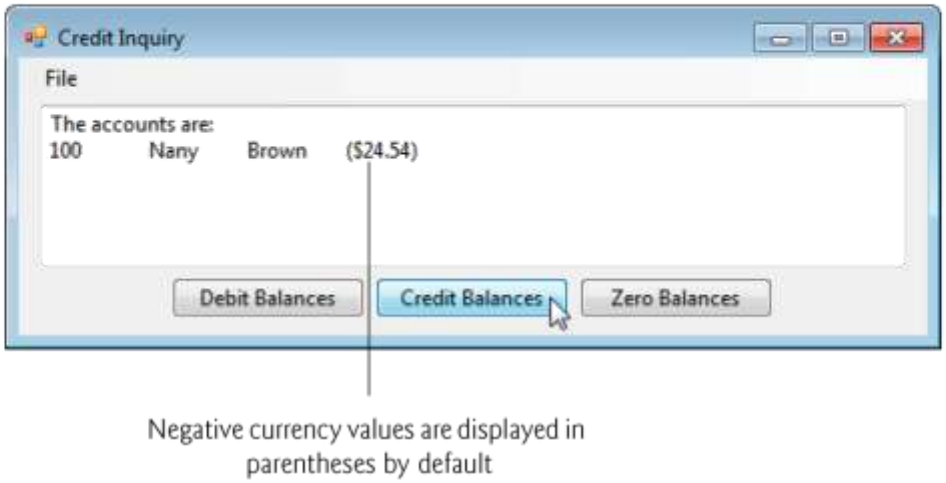


Fig. 8.4 | GUI for Credit Inquiry application. (Part 2 of 3.)

c) Clicking the **Zero Balances** `Button` displays the accounts with zero balances (that is, the people who do not have a balance because they've already paid or have not had any recent transactions)

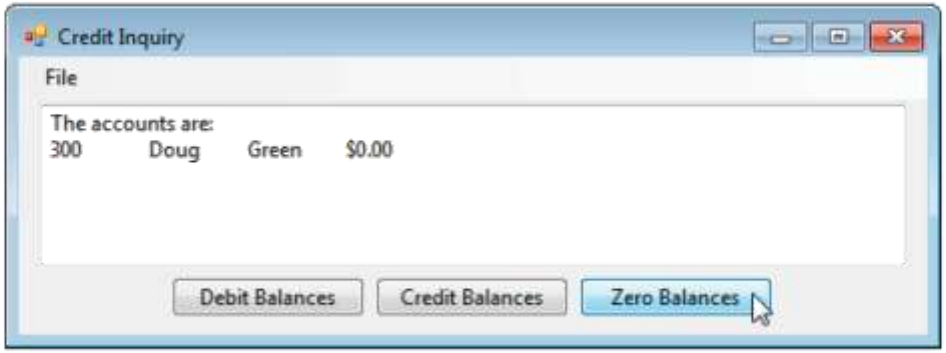


Fig. 8.4 | GUI for Credit Inquiry application. (Part 3 of 3.)

8.5 Writing Data Sequentially to a Text File

- Before we can implement the Credit Inquiry application, we must create the file from which that application will read records.
- Our first program builds the sequential file containing the account information for the company's clients.
- For each client, the program obtains through its GUI the client's account number, first name, last name and balance—the amount of money that the client owes to the company for previously purchased goods and services.

8.5 Writing Data Sequentially to a Text File

- The data obtained for each client constitutes a “record” for that client.
- In this application, the account number is used as the record key—files are often maintained in order by their record keys.
- For simplicity, this program assumes that the user enters records in account number order.

8.5 Writing Data Sequentially to a Text File

- GUI for the *Create Accounts Application*
  - The GUI for the Create Accounts application is shown in Fig. 8.5.
  - This application introduces the MenuStrip control which enables you to place a menu bar in your window.
  - It also introduces ToolStripMenuItem controls which are used to create menus and menu items.
  - We show how use the IDE to build the menu and menu items in Section 8.6.
  - There you'll see that the menu and menu item variable names are generated by the IDE and begin with capital letters.
  - Like other controls, you can change the variable names in the Properties window by modifying the (Name) property.

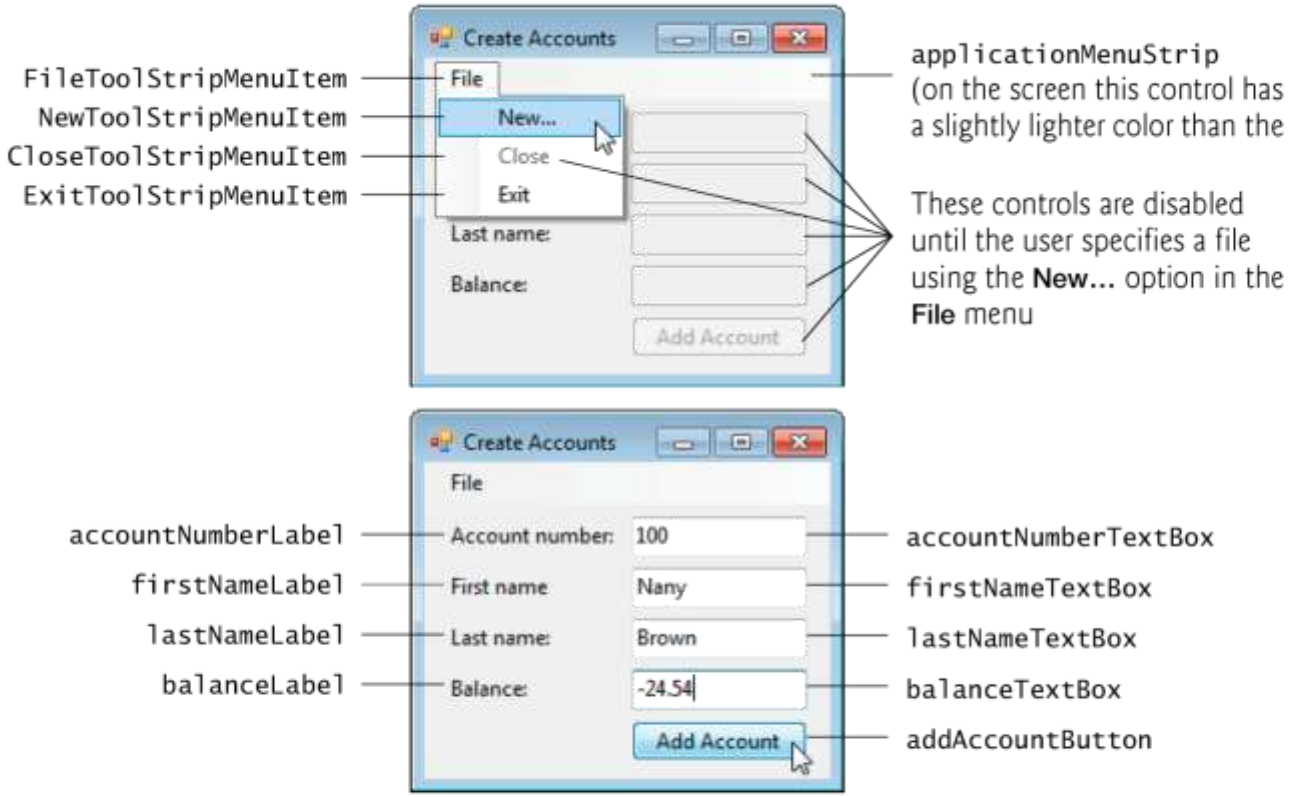


Fig. 8.5 | GUI for the **Create Accounts** application.

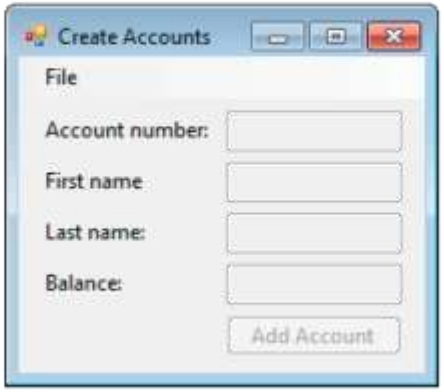
8.5 Writing Data Sequentially to a Text File

- Interacting with the *Create Accounts Application*
  - When the user initially executes this application, the Close menu item, the `TextBox`es and the Add Account `Button` are disabled (Fig. 8.6(a))—the user can interact with these controls only after specifying the file into which the records will be saved.
  - To begin creating a file of accounts, the user selects File > New… (Fig. 8.6(b)), which displays a Save As dialog (Fig. 8.6(c)) that allows the user to specify the name and location of the file into which the records will be placed.
  - The File menu provides two other menu items—Close to close the file so the user can create another file and Exit to terminate the application.

8.5 Writing Data Sequentially to a Text File

- After the user specifies a file name, the application opens the file and enables the controls, so the user can begin entering account information.
- Figure 8.6(d)–(h) shows the sample data being entered for five accounts.
- The program does not depict how the records are stored in the file.
- This is a text file, so after you close the program, you can open the file in any text editor to see its contents.
- Figure 8.6(j) shows the file's contents in Notepad.

a) Initial GUI before user selects a file



b) Selecting **New...** to create a file

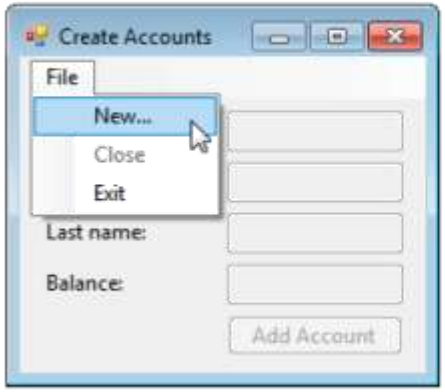


Fig. 8.6 | User creating a text file of account information. (Part 1 of 4.)

c) **Save As** dialog displayed when user selects **New...** from the **File** menu.

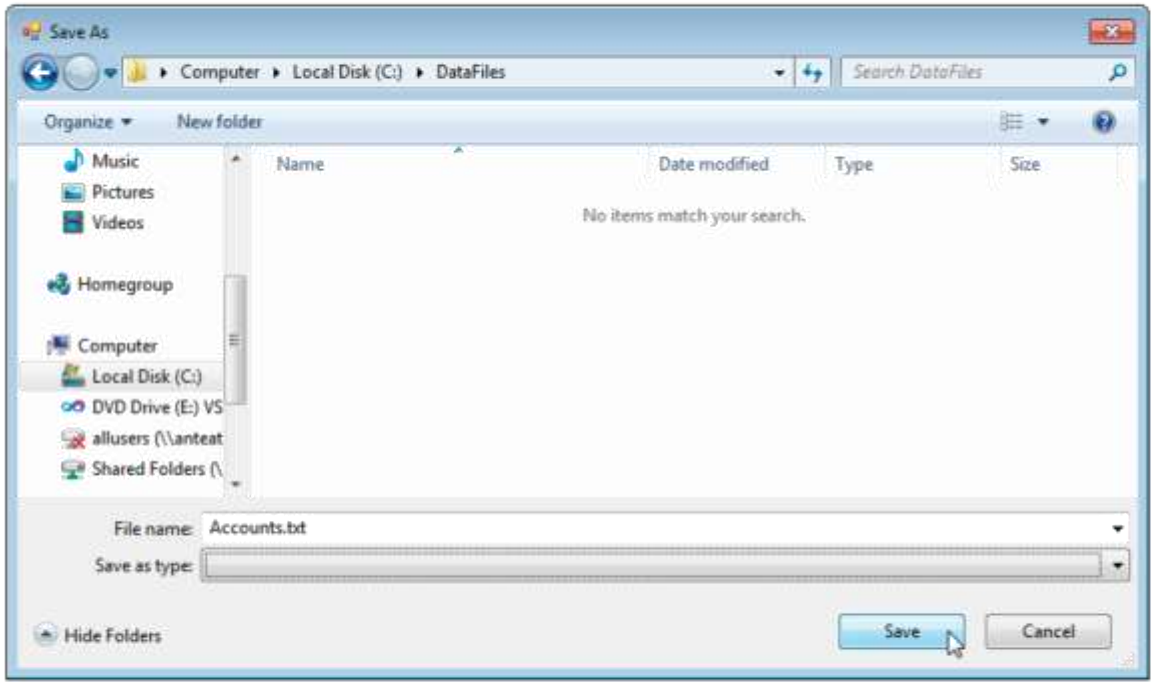


Fig. 8.6 | User creating a text file of account information. (Part 2 of 4.)

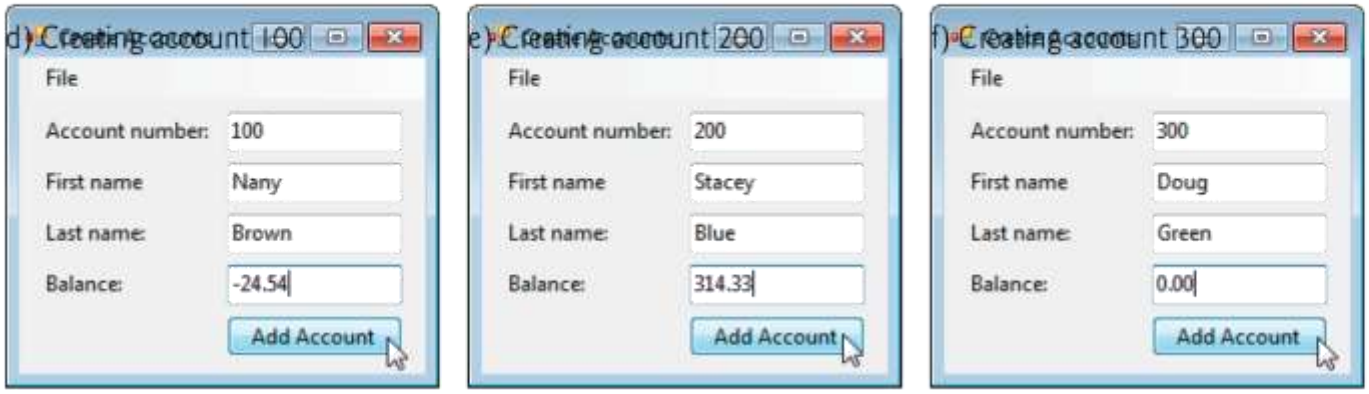


Fig. 8.6 | User creating a text file of account information. (Part 3 of 4.)

j) The `Accounts.txt` file open in **Notepad** to show how the records were written to the file. Note the comma separators between the data items

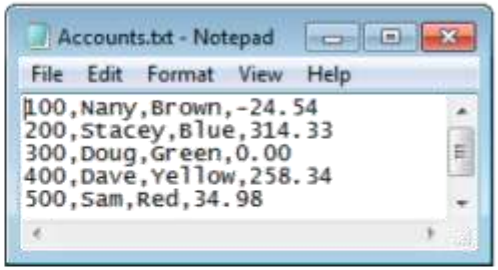


Fig. 8.6 | User creating a text file of account information. (Part 4 of 4.)

8.5.1 Class `CreateAccounts`

- Let's now study the declaration of class `CreateAccounts`, which begins in Fig. 8.7.
- We've split this class into several figures.
- Framework Class Library classes are grouped by functionality into namespaces, which make it easier for you to find the classes needed to perform particular tasks.
- Line 3 is an Imports statement, which indicates that we're using classes from the System.IO namespace.
- This namespace contains stream classes such as StreamWriter (for text output) and StreamReader (for text input).
- Line 6 declares `fileWriter` as an instance variable of type `StreamWriter`.
- We'll use this variable to interact with the user's file.

```
1  ' Fig. 8.7: CreateAccounts.vb
2  ' Program that creates a text file of account information.
3  Imports System.IO ' using classes from this namespace
4
5  Public Class CreateAccounts
6     Dim fileWriter As StreamWriter ' writes data to text file
7
```

Fig. 8.7 | Program that creates a text file of account information.

8.5.2 Class CreateAccounts

- You must import `StreamWriter` before you can use it.
- In fact, all namespaces except `System` must be imported into a program to use the classes in those namespaces.
- Namespace `System` is imported by default into every program.
- Classes like `String`, `Convert` and `Math` used in earlier examples are declared in the `System` namespace.
- So far, we have not used `Imports` statements in any of our programs, but we have used many classes from namespaces that must be imported.
- For example, all of the GUI controls you've used so far are classes in the `System.Windows.Forms` namespace.
- So why were we able to compile those programs? When you create a project, each Visual Basic project type automatically imports several namespaces that are commonly used with that project type.

8.5.2 Class CreateAccounts

- You can see the namespaces (Fig. 8.8) that were automatically imported into your project by right clicking the project's name in the Properties window, selecting Properties from the menu and clicking the References tab.
- The list appears under Imported namespaces:—each namespace with a checkmark is automatically imported into the project.
- This application is a Windows Forms application.The `System.IO` namespace is not imported by default.
- To import a namespace, you can either use an `Imports` statement (as in line 3 of Fig. 8.7) or you can scroll through the list in Fig. 8.8 and check the checkbox for the namespace you wish to import.

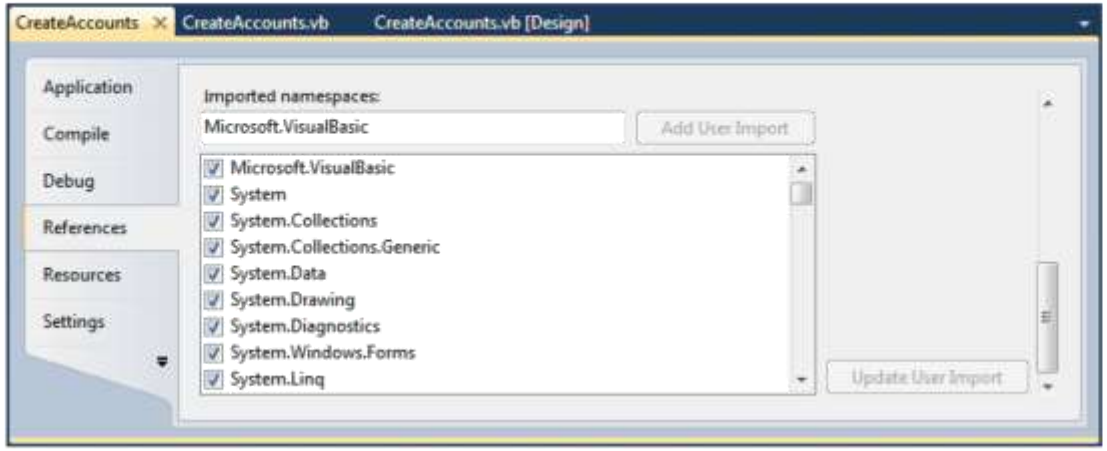


Fig. 8.8 | Viewing the namespaces that are pre-`Imported` into a Windows Forms application.

8.5.3 Opening the File

- When the user selects File > New..., method `NewToolStripMenuItem_Click` (Fig. 8.9) is called to handle the New... menu item's `Click` event.
- This method opens the file.
- First, line 12 calls method `CloseFile` (Fig. 8.11, lines 102–111) in case the user previously opened another file during the current execution of the application.
- `CloseFile` closes the file associated with this application's `StreamWriter`.

```
8       ' create a new file in which accounts can be stored
9       Private Sub NewToolStripMenuItem_Click(ByVal sender As System.Object,
10         ByVal e As System.EventArgs) Handles NewToolStripMenuItem.Click
11
12         CloseFile() ' ensure that any prior file is closed
13         Dim result As DialogResult ' stores result of Save dialog
14         Dim fileName As String ' name of file to save data
15
16         ' display dialog so user can choose the name of the file to save
17         Using fileChooser As New SaveFileDialog()
18            result = fileChooser.ShowDialog()
19            fileName = fileChooser.FileName ' get specified file name
20         End Using ' automatic call to fileChooser.Dispose() occurs here
21
22         ' if user did not click Cancel
23         If result <> Windows.Forms.DialogResult.Cancel Then
24            Try
25               ' open or create file for writing
26               fileWriter = New StreamWriter(fileName, True)
27
```

Fig. 8.9 | Using the `SaveFileDialog` to allow the user to select the file into which records will be written. (Part 1 of 2.)

```
28              ' enable controls
29              CloseToolStripMenuItem.Enabled = True
30              addAccountButton.Enabled = True
31              accountNumberTextBox.Enabled = True
32              firstNameTextBox.Enabled = True
33              lastNameTextBox.Enabled = True
34              balanceTextBox.Enabled = True
35           Catch ex As IOException
36              MessageBox.Show("Error Opening File", "Error",
37                 MessageBoxButtons.OK, MessageBoxIcon.Error)
38           End Try
39        End If
40     End Sub ' NewToolStripMenuItem_Click
41
```

Fig. 8.9 | Using the `SaveFileDialog` to allow the user to select the file into which records will be written. (Part 2 of 2.)

8.5.3 Opening the File

- Next, lines 17–20 display the Save As dialog and get the file name specified by the user.
- First, line 17 creates the SaveFileDialog object (namespace `System.Windows.Forms`) named `fileChooser`.
- Line 18 calls its ShowDialog method to display the `SaveFileDialog` (Fig. 8.6(c)).
- This dialog prevents the user from interacting with any other window in the program until the user closes it by clicking either Save or Cancel, so it's a *modal dialog*.

8.5.3 Opening the File

- The user selects the location where the file should be stored and specifies the file name, then clicks Save.
- Method `ShowDialog` returns a DialogResult enumeration constant specifying which button (Save or Cancel) the user clicked to close the dialog.
- This is assigned to the `DialogResult` variable `result` (line 18).
- Line 19 uses `SaveFileDialog` property FileName to obtain the location and name of the file.

8.5.4 Managing Resources with the `Using` Statement

- Lines 17–20 introduce the Using statement, which simplifies writing code in which you obtain, use and release a resource.
- In this case, the resource is a `SaveFileDialog`.
- Windows and dialogs are limited system resources that occupy memory and should be returned to the system (to free up that memory) as soon as they're no longer needed.
- In all our previous applications, this happens when the program terminates.

8.5.4 Managing Resources with the `Using` Statement

- In a long running program, if resources are not returned to the system when they're no longer needed, then a resource leak occurs and the resources are not available for use in this or other programs.
- Objects that represent such resources typically provide a `Dispose` method that must be called to return the resources to the system.
- The `Using` statement in lines 17–20 creates a `SaveFileDialog` object, uses it in lines 18–19, then automatically calls its `Dispose` method to release the object's resources as soon as `End Using` is reached, thus guaranteeing that the resources are returned to the system and the memory they occupy is freed up.

8.5.4 Managing Resources with the `Using` Statement

- Line 23 tests whether the user clicked `Cancel` by comparing `result` to the constant `Windows.Forms.DialogResult.Cancel`.
- If not, line 26 creates a `StreamWriter` object that we'll use to write data to the file.
- The two arguments are a `String` representing the location and name of the file, and a `Boolean` indicating what to do if the file already exists.
- If the file doesn't exist, this statement creates the file.
- If the file does exist, the second argument (`True`) indicates that new data written to the file should be appended at the end of the file's current contents.

8.5.4 Managing Resources with the `Using` Statement

- If the second argument is `False` and the file already exists, the file's contents will be discarded and new data will be written starting at the beginning of the file.
- Lines 29–34 enable the `Close` menu item and the `TextBox`es and `Button` that are used to enter records into the program.
- Lines 35–37 catch an IOException if there is a problem opening the file.
- If so, the program displays an error message.
- If no exception occurs, the file is opened for writing.
- Most file-processing operations have the potential to throw exceptions, so such operations are typically placed in `Try` statements.

8.5.5 Adding an Account to the File

- After typing information in each `TextBox`, the user clicks the Add Account `Button`, which calls method `addAccountButton_Click` (Fig. 8.10) to save the data into the file.
- If the user entered a valid account number (that is, an integer greater than zero), lines 56–59 write the record to the file by invoking the `StreamWriter`'s WriteLine method, which writes a sequence of characters to the file and positions the output cursor to the beginning of the next line in the file.
- We separate each field in the record with a comma in this example (this is known as a comma-delimited text file), and we place each record on its own line in the file.

8.5.5 Adding an Account to the File

- If an `IOException` occurs when attempting to write the record to the file, lines 64–66 `Catch` the exception and display an appropriate message to the user.
- Similarly, if the user entered invalid data in the `accountNumberTextBox` or `balanceTextBox` lines 67–69 catch the FormatExceptions thrown by class `Convert`'s methods and display an appropriate error message.
- Lines 73–77 clear the `TextBox`es and return the focus to the `accountNumberTextBox` so the user can enter the next record.

```
42      ' add an account to the file
43      Private Sub addAccountButton_Click(ByVal sender As System.Object,
44         ByVal e As System.EventArgs) Handles addAccountButton.Click
45
46         ' determine whether TextBox account field is empty
47         If accountNumberTextBox.Text <> String.Empty Then
48            ' try to store record to file
49            Try
50               ' get account number
51               Dim accountNumber As Integer =
52                  Convert.ToInt32(accountNumberTextBox.Text)
53
54               If accountNumber > 0 Then ' valid account number?
55                  ' write record data to file separating fields by commas
56                  fileWriter.WriteLine(accountNumber & "," &
57                     firstNameTextBox.Text & "," &
58                     lastNameTextBox.Text & "," &
59                     Convert.ToDecimal(balanceTextBox.Text))
60               Else
61                  MessageBox.Show("Invalid Account Number", "Error",
62                     MessageBoxButtons.OK, MessageBoxIcon.Error)
63               End If
```

Fig. 8.10 | Writing an account record to the file. (Part 1 of 2.)

```
64            Catch ex As IOException
65               MessageBox.Show("Error Writing to File", "Error",
66                  MessageBoxButtons.OK, MessageBoxIcon.Error)
67            Catch ex As FormatException
68               MessageBox.Show("Invalid account number or balance",
69                  "Format Error", MessageBoxButtons.OK, MessageBoxIcon.Error)
70            End Try
71         End If
72
73         accountNumberTextBox.Clear()
74         firstNameTextBox.Clear()
75         lastNameTextBox.Clear()
76         balanceTextBox.Clear()
77         accountNumberTextBox.Focus()
78      End Sub ' addAccountButton_Click
79
```

Fig. 8.10 | Writing an account record to the file. (Part 2 of 2.)

8.5.6 Closing the File and Terminating the Application

- When the user selects File > Close, method `CloseToolStripMenuItem_Click` (Fig. 8.11, lines 81–91) calls method `CloseFile` (lines 102–111) to close the file.
- Then lines 85–90 disable the controls that should not be available when a file is not open.

```
80      ' close the currently open file and disable controls
81      Private Sub CloseToolStripMenuItem_Click(ByVal sender As System.Object,
82         ByVal e As System.EventArgs) Handles CloseToolStripMenuItem.Click
83
84         CloseFile() ' close currently open file
85         CloseToolStripMenuItem.Enabled = False
86         addAccountButton.Enabled = False
87         accountNumberTextBox.Enabled = False
88         firstNameTextBox.Enabled = False
89         lastNameTextBox.Enabled = False
90         balanceTextBox.Enabled = False
91      End Sub ' CloseToolStripMenuItem_Click
92
93      ' exit the application
94      Private Sub ExitToolStripMenuItem_Click(ByVal sender As System.Object,
95         ByVal e As System.EventArgs) Handles ExitToolStripMenuItem.Click
96
97         CloseFile() ' close the file before terminating application
98         Application.Exit() ' terminate the application
99      End Sub ' ExitToolStripMenuItem_Click
100
```

Fig. 8.11 | Closing the file and terminating the application. (Part 1 of 2.)

```
101     ' close the file
102     Sub CloseFile()
103        If fileWriter IsNot Nothing Then
104           Try
105              fileWriter.Close() ' close StreamWriter
106           Catch ex As IOException
107              MessageBox.Show("Error closing file", "Error",
108                 MessageBoxButtons.OK, MessageBoxIcon.Error)
109           End Try
110        End If
111     End Sub ' CloseFile
112  End Class ' CreateAccounts
```

Fig. 8.11 | Closing the file and terminating the application. (Part 2 of 2.)

8.5.6 Closing the File and Terminating the Application

- When the user clicks the Exit menu item, method `ExitToolStripMenuItem_Click` (lines 94–99) responds to the menu item's `Click` event by exiting the application.
- Line 97 closes the `StreamWriter` and the associated file, then line 98 terminates the program.
- The call to method `Close` (line 105) is located in a `Try` block.
- Method Close throws an `IOException` if the file cannot be closed properly.
- In this case, it's important to notify the user that the information in the file or stream might be corrupted.

8.6 Building Menus with the Windows Forms Designer

- In the test-drive of the Credit Inquiry application (Section 8.4) and in the overview of the Create Accounts application (Section 8.5), we demonstrated how menus provide a convenient way to organize the commands that you use to interact with an application without "cluttering" its user interface.
- Menus contain groups of related commands.

8.6 Building Menus with the Windows Forms Designer

- When a command is selected, the application performs a specific action (for example, select a file to open, exit the application, etc.).
- Menus make it simple and straightforward to locate an application's commands.
- They can also make it easier for users to use applications.
- For example, many applications provide a File menu that contains an Exit menu item to terminate the application.

8.6 Building Menus with the Windows Forms Designer

- If this menu item is always placed in the File menu, then users become accustomed to going to the File menu to terminate an application.
- When they use a new application and it has a File menu, they'll already be familiar with the location of the Exit command.
- The menu that contains a menu item is that menu item's parent menu.
- In the Create Accounts application, File is the parent menu that contains three menu items—New…, Close and Exit.

8.6 Building Menus with the Windows Forms Designer

- Adding a ***`MenuStrip` to the `Form`***
  - Before you can place a menu on your application, you must provide a MenuStrip to organize and manage the application's menus.
  - Double click the `MenuStrip` control in the Toolbox.
  - This creates a menu bar (the `MenuStrip`) across the top of the `Form` (below the title bar; Fig. 8.12) and places a `MenuStrip` icon in the component tray (the gray area) at the bottom of the designer.
  - You can access the `MenuStrip`'s properties in the Properties window by clicking the `MenuStrip` icon in the component tray.
  - We set the `MenuStrip`'s (Name) property to `applicationMenuStrip`.

8.6 Building Menus with the Windows Forms Designer

- Adding a ***ToolStripMenuItem*** to ***MenuStrip***
  - You can now use Design mode to create and edit menus for your application.
  - To add a menu, click the Type Here `TextBox` (Fig. 8.12) in the menu bar and type the menu's name.
  - For the File menu, type `&File` (we'll explain the & in a moment) then press *Enter*.
  - This creates a `ToolStripMenuItem` that the IDE automatically names `FileToolStripMenuItem`.
  - Additional Type Here `TextBox`es appear, allowing you to add menu items to the menu or add more menus to the menu bar (Fig. 8.13).

8.6 Building Menus with the Windows Forms Designer

- Most menus and menu items provide access shortcuts (or keyboard shortcuts) that allow users to open a menu or select a menu item by using the keyboard.
- For example, most applications allow you to open the File menu by typing *Alt* + *F*.
- The letter that's used as the shortcut is underlined in the GUI when you press the *Alt key*.
- To specify the shortcut key, type an ampersand (&) before the character to be underlined—so `&File` underlines the F in File.

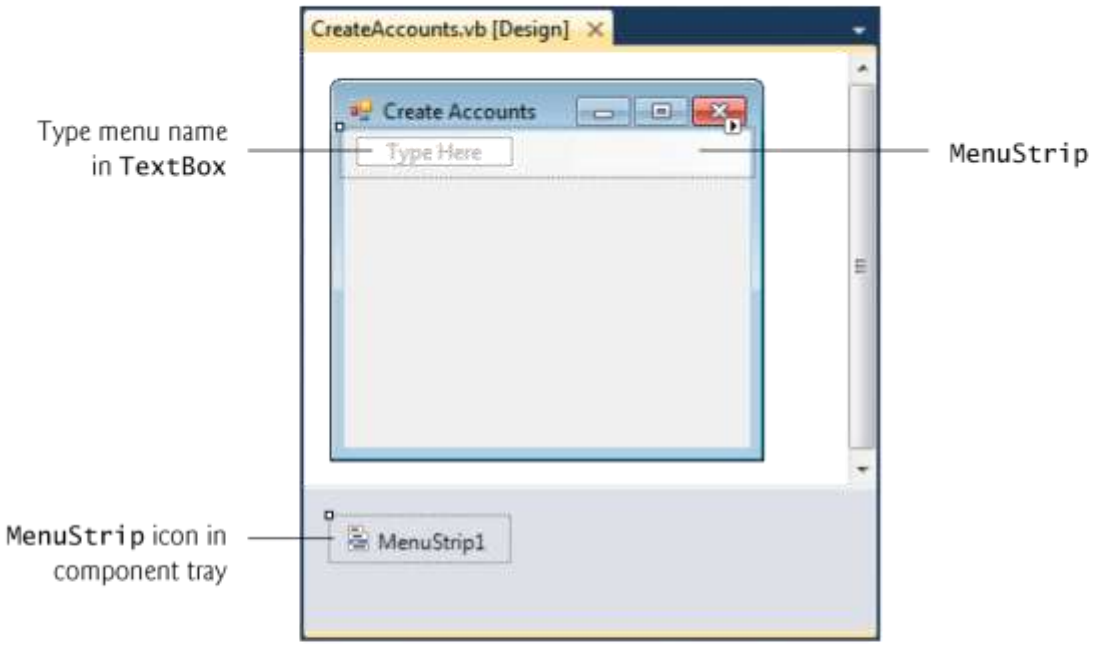


Fig. 8.12 | Editing menus in Visual Studio.

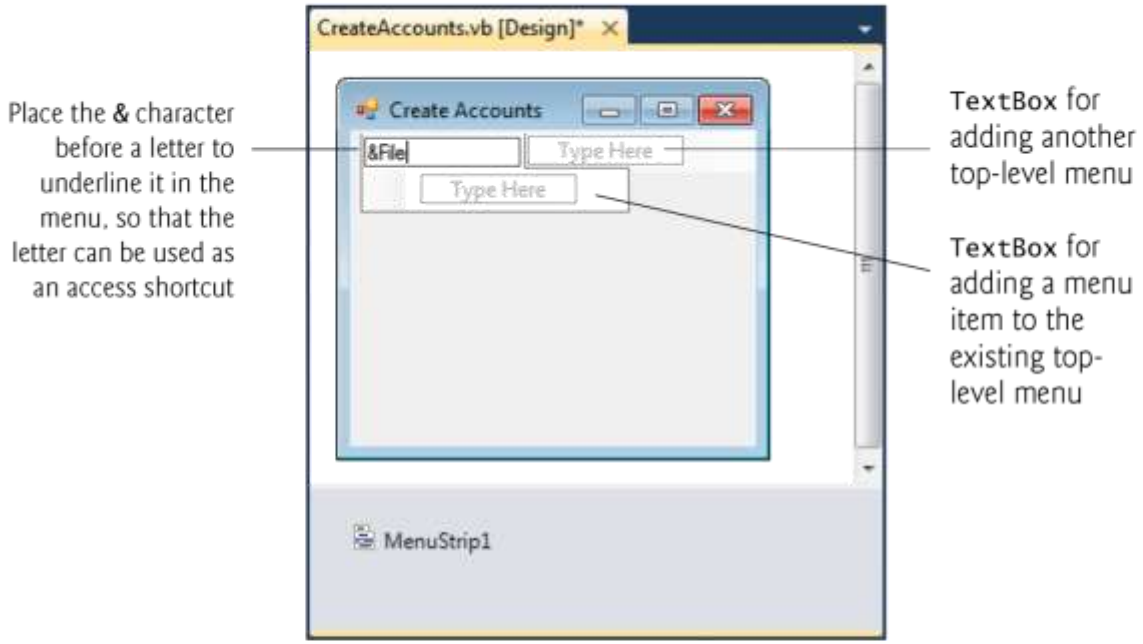


Fig. 8.13 | Adding `ToolStripMenuItems` to a `MenuStrip`.

8.6 Building Menus with the Windows Forms Designer

- Adding Menu Items to the *File Menu*
  - To add the New…, Close and Exit menu items to the File menu, type `&New...`, `&Close` and `E&xit` (one at a time) into the `TextBox` that appears below the File menu.
  - When you press *Enter after each, a new* `TextBox` *appears below that item so you can add another menu item.*
  - Placing the `&` before the `x` in `Exit` makes the `x` the access key—`x` is commonly used as the access key for the Exit menu item.
  - The menu editor automatically names the `ToolStripMenuItems` for the New…, Close and Exit menu items as `NewToolStripMenuItem`, `CloseToolStripMenuItem` and `ExitToolStripMenuItem`, respectively.

Look-and-Feel Observation 8.1

By convention, place an ellipsis (...) after the name of a menu item that, when selected, displays a dialog (e.g. *New...*).

8.6 Building Menus with the Windows Forms Designer

- Creating Event Handlers for the Menu Items
  - Like `Button`s, menu items have `Click` events that notify the program when an item is selected.
  - To create the event handler for a menu item so the application can respond when the menu item is selected, double click the menu item in the Windows Forms Designer then insert your event handling code in the new method's body.
  - In fact, the same event handler method can be used for `Button`s and menu items that perform the same task.

8.7 Credit Inquiry Application: Reading Data Sequentially from a Text File

- Now that we've presented the code for creating the file of accounts, let's develop the code for the Credit Inquiry application which reads that file.
- Much of the code in this example is similar to the Create Accounts application, so we'll discuss only the unique aspects of the application.

8.7.1 Implementing the Credit Inquiry Application

- The declaration of class `CreditInquiry` begins in Fig. 8.14.
- Line 4 imports the `System.IO` namespace, which contains the `StreamReader` class that we'll use to read from the text file in this example.
- Line 7 declares the instance variable `fileName` in which we store the file name selected by the user (that is, credit manager) in the Open dialog (Fig. 8.3(c)).
- Lines 9–13 declare the enumeration `AccountType`, which creates constants that represent the types of accounts that can be displayed.

```
1  ' Fig. 8.14: CreditInquiry.vb
2  ' Read a file sequentially and display contents based on
3  ' account type specified by user (credit, debit or zero balances).
4  Imports System.IO ' using classes from this namespace
5
6  Public Class CreditInquiry
7     Private fileName As String ' name of file containing account data
8
9     Enum AccountType ' constants representing account types
10       CREDIT
11       DEBIT
12       ZERO
13    End Enum ' AccountType
14
```

Fig. 8.14 | Declaring the `fileName` instance variable and creating the `AccountType` enumeration that's used to specify the type of account to display.

8.7.2 Selecting the File to Process

- When the user selects File > Open…, the event handler `OpenToolStripMenuItem_Click` (Fig. 8.15, lines 16–33) executes.
- Line 22 creates an OpenFileDialog, and line 23 calls its ShowDialog method to display the Open dialog, in which the user selects the file to open.
- Line 24 stores the selected file name in `fileName`.

```
15      ' opens a file in which accounts are stored
16      Private Sub OpenToolStripMenuItem_Click(ByVal sender As System.Object,
17         ByVal e As System.EventArgs) Handles OpenToolStripMenuItem.Click
18
19         Dim result As DialogResult ' stores result of Open dialog
20
21         ' create dialog box enabling user to open file
22         Using fileChooser As New OpenFileDialog()
23            result = fileChooser.ShowDialog()
24            fileName = fileChooser.FileName ' get specified file name
25         End Using ' automatic call to fileChooser.Dispose() occurs here
26
27         ' if user did not click Cancel, enable Buttons
28         If result <> Windows.Forms.DialogResult.Cancel Then
29            creditBalancesButton.Enabled = True
30            debitBalancesButton.Enabled = True
31            zeroBalancesButton.Enabled = True
32         End If
33      End Sub ' OpenToolStripMenuItem_Click
34
```

Fig. 8.15 | Event handlers for the **Open...** and **Exit** menu items. (Part 1 of 2.)

```
35      ' exit the application
36      Private Sub ExitToolStripMenuItem_Click(ByVal sender As System.Object,
37         ByVal e As System.EventArgs) Handles ExitToolStripMenuItem.Click
38
39         Application.Exit() ' terminate the application
40      End Sub ' ExitToolStripMenuItem_Click
41
```

Fig. 8.15 | Event handlers for the **Open...** and **Exit** menu items. (Part 2 of 2.)

8.7.3 Specifying the Type of Records to Display

- When the user clicks the Credit Balances, Debit Balances or Zero Balances `Button`, the program invokes the corresponding event-handler method—`creditBalancesButton_Click` (Fig. 8.16, lines 43–47), `debitBalancesButton_Click` (lines 50–54) or `zeroBalancesButton_Click` (lines 57–61).
- Each of these methods calls method `DisplayAccounts` (Fig. 8.17), passing a constant from the `AccountType` enumeration as an argument.
- Method `DisplayAccounts` then displays the matching accounts.

```
42      ' display accounts with credit balances
43      Private Sub creditBalancesButton_Click(ByVal sender As System.Object,
44         ByVal e As System.EventArgs) Handles creditBalancesButton.Click
45
46         DisplayAccounts(AccountType.CREDIT) ' displays credit balances
47      End Sub ' creditBalancesButton_Click
48
49      ' display accounts with debit balances
50      Private Sub debitBalancesButton_Click(ByVal sender As System.Object,
51         ByVal e As System.EventArgs) Handles debitBalancesButton.Click
52
53         DisplayAccounts(AccountType.DEBIT) ' displays debit balances
54      End Sub ' debitBalancesButton_Click
55
56      ' display accounts with zero balances
57      Private Sub zeroBalancesButton_Click(ByVal sender As System.Object,
58         ByVal e As System.EventArgs) Handles zeroBalancesButton.Click
59
60         DisplayAccounts(AccountType.ZERO) ' displays zero balances
61      End Sub ' zeroBalancesButton_Click
62
```

Fig. 8.16 | Each `Button` event handler calls method `DisplayAccounts` and passes the appropriate `AccountType` as an argument to specify which accounts to display.

8.7.4 Displaying the Records

- Method `DisplayAccounts` (Fig. 8.17, lines 64–104) receives as an argument an `AccountType` constant specifying the type of accounts to display.
- The method reads the entire file one record at a time until the end of the file is reached, displaying a record only if its balance matches the type of accounts specified by the user.
- Opening the File
  - Line 65 declares the `StreamReader` variable `fileReader` that will be used to interact with the file.
  - Line 72 opens the file by passing the `fileName` instance variable to the `StreamReader` constructor.

```
63      ' display accounts of specified type
64      Sub DisplayAccounts(ByVal accountType As AccountType)
65         Dim fileReader As StreamReader = Nothing
66
67         ' read and display file information
68         Try
69            accountsTextBox.Text = "The accounts are:" & vbCrLf
70
71            ' open file for reading
72            fileReader = New StreamReader(fileName)
73
74            ' read file and display lines that match the balance type
75            Do While Not fileReader.EndOfStream ' while not end of file
76               Dim line As String = fileReader.ReadLine() ' read line
77               Dim fields() As String = line.Split(","c) ' split into fields
78
79               ' get data from fields array
80               Dim accountNumber As Integer = Convert.ToInt32(fields(0))
81               Dim firstName As String = fields(1)
```

Fig. 8.17 | Method `DisplayAccounts` opens the file, reads one record at a time, displays the record if it matches the selected `AccountType` and closes the file when all records have been processed. (Part 1 of 4.)

```
82              Dim lastName As String = fields(2)
83              Dim balance As Decimal = Convert.ToDecimal(fields(3))
84
85              If ShouldDisplay(balance, accountType) Then
86                 accountsTextBox.AppendText(accountNumber & vbTab &
87                    firstName & vbTab & lastName & vbTab &
88                    String.Format("{0:C}", balance) & vbCrLf)
89              End If
90           Loop
91        Catch ex As IOException
92           MessageBox.Show("Cannot Read File", "Error",
93              MessageBoxButtons.OK, MessageBoxIcon.Error)
94        Finally ' ensure that file gets closed
95           If fileReader IsNot Nothing Then
96              Try
97                 fileReader.Close() ' close StreamReader
```

Fig. 8.17 | Method `DisplayAccounts` opens the file, reads one record at a time, displays the record if it matches the selected `AccountType` and closes the file when all records have been processed. (Part 2 of 4.)

```
98              Catch ex As IOException
99                 MessageBox.Show("Error closing file", "Error",
100                   MessageBoxButtons.OK, MessageBoxIcon.Error)
101             End Try
102          End If
103       End Try
104    End Sub ' DisplayAccounts
105
106    ' determine whether to display given account based on the balance
107    Function ShouldDisplay(ByVal balance As Double,
108       ByVal type As AccountType) As Boolean
109
110       If balance < 0 AndAlso type = AccountType.CREDIT Then
111          Return True ' record should be displayed
112       ElseIf balance > 0 AndAlso type = AccountType.DEBIT Then
113          Return True ' record should be displayed
114       ElseIf balance = 0 AndAlso type = AccountType.ZERO Then
115          Return True ' record should be displayed
116       End If
```

Fig. 8.17 | Method `DisplayAccounts` opens the file, reads one record at a time, displays the record if it matches the selected `AccountType` and closes the file when all records have been processed. (Part 3 of 4.)

```
117
118         Return False ' record should not be displayed
119     End Function ' ShouldDisplay
120 End Class ' Credit Inquiry
```

Fig. 8.17 | Method `DisplayAccounts` opens the file, reads one record at a time, displays the record if it matches the selected `AccountType` and closes the file when all records have been processed. (Part 4 of 4.)

8.7.4 Displaying the Records

- Reading and Processing Records
  - The company could have many separate files containing account information.
  - So this application does not know in advance how many records will be processed.
  - In file processing, we receive an indication that the end of the file has been reached when we've read the entire contents of a file.
  - For a `StreamReader`, this is when its EndOfStream property returns `True` (line 75).

8.7.4 Displaying the Records

- As long as the end of the file has not been reached, line 76 uses the `StreamReader`'s ReadLine method (which returns a `String`) to read one line of text from the file.
- Recall from Section 8.5 that a each line of text in the file represents one "record" and that the record's fields are delimited by commas.

8.7.4 Displaying the Records

- To access the record's data, we need to break the `String` into its separate fields.
- This is known as tokenizing the `String`.
- Line 77 breaks the line of text into fields using `String` method Split, which receives a delimiter as an argument.
- In this case, the delimiter is the character literal `","c`—indicating that the delimiter is a comma.
- A character literal looks like a `String` literal that contains one character and is followed immediately by the letter `c`.
- Method `Split` returns an array of `String`s representing the tokens, which we assign to array variable `fields`.

8.7.4 Displaying the Records

- Preparing to Display the Record
  - Lines 80–83 assign the tokens to the local variables `accountNumber`, `firstName`, `lastName` and `balance`.
  - Line 85 calls method `ShouldDisplay` (lines 107–119) to determine whether the current record should be displayed.
  - If so, lines 86–88 display the record.
  - If the balance is negative, the currency format specifier (`C`) formats the value in parentheses (Fig. 8.4(b)).
  - Method `ShouldDisplay` receives the balance and the `AccountType` as arguments.
  - If the balance represents the specified `AccountType`, the method returns `True` and the record will be displayed by method `DisplayAccounts`.

8.7.4 Displaying the Records

- Ensuring that the File is Closed Properly
  - When performing file processing, exceptions can occur.
  - In this example, if the program is unable to open the file or unable to read from the file, `IOExceptions` will occur.
  - For this reason, file-processing code normally appears in a `Try` block.
  - Regardless of whether a program experiences exceptions while processing a file, the program should close the file when it's no longer needed.
  - Suppose we put the statement that closes the `StreamReader` after the `Do While...Loop` at line 91.

8.7.4 Displaying the Records

- If no exceptions occur, the `Try` block executes normally and the file is closed.
- However, if an exception occurs, the `Try` block terminates immediately—*before the* `StreamReader` *can be closed.*
- We could duplicate the statement that closes the `StreamReader` in the `Catch` block, but this would make the code more difficult to modify and maintain.
- We could also place the statement that closes the `StreamReader` after the `Try` statement; however, if the `Try` block terminated due to a `Return` statement, code following the `Try` statement would never execute.
- To address these problems, Visual Basic's exception-handling mechanism provides the optional Finally block, which—if present—is *guaranteed to execute regardless of whether the* `Try` *block executes successfully or an exception occurs.*

8.7.4 Displaying the Records

- This makes the `Finally` block an ideal location in which to place resource release code for resources that are acquired and manipulated in the corresponding `Try` block (such as files).
- By placing the statement that closes the `StreamReader` in a `Finally` block, we ensure that the file will always be closed properly.
- Local variables in a `Try` block cannot be accessed in the corresponding `Finally` block.
- For this reason, variables that must be accessed in both a `Try` block and its corresponding `Finally` block should be declared before the `Try` block, as we did with the `StreamReader` variable (line 65).

8.7.4 Displaying the Records

- Relationship Between the `Using` and `Try` Statements
  - In Section 8.5.4, we discussed how a `Using` statement manages resources.
  - The `Using` statement is actually a shorthand notation for a `Try` statement with a `Finally` block.
  - For example, the `Using` statement in Fig. 8.15 (lines 22–25) is equivalent to the following code

```
Dim fileChooser As New OpenFileDialog()

Try
   result = fileChooser.ShowDialog()
   ' get specified file name
   fileName = fileChooser.FileName
Finally
   fileChooser.Dispose()
End Try
```