```
        Do Until sr.EndOfStream
          temp = sr.ReadLine
        Loop
        MessageBox.Show(temp)
        sr.Close()
```

Second:
```
        Dim states() As String = IO.File.ReadAllLines("USStates.txt")
        MessageBox.Show(states.Last)
```

Third:
```
        Dim states() As String = IO.File.ReadAllLines("USStates.txt")
        MessageBox.Show(states(states.Count – 1))
```

2. The file ABC.txt will be present in the program's *bin\Debug* folder. However, the file will be empty. In order for the string *abc* to be placed in the file, the statement **`sw.Close()`** must be executed.

8.3 XML

As we have seen, CSV files are quite useful. However, the expansion of the Internet mandated a standard text file format for transmitting data. The World Wide Web Consortium recommends a format called XML (eXtensible Markup Language) to be that standard.

Consider the CSV file USStates.txt. Let's look at a text file consisting of the first two lines of USStates.txt. That is, the two lines of the new file are

```
Delaware,DE,1954,759000
Pennsylvania,PA,44817,12296000
```

We are familiar with these records and know what each field represents. However, if we showed this file to someone else, they might not know how to interpret the information. One way to present it in great detail and in an organized format would be as follows:

```
U.S. States
  state
    name: Delaware
    abbreviation: DE
    area: 1954
    population: 749000
  state
    name: Pennsylvania
    abbreviation: PA
    area: 44817
    population: 1229600
```

Format of XML Files

VideoNote
XML

The XML format for the state data, which has the look and feel of the presentation above, is as follows:

```
<?xml version='1.0'?>
<!-- This file contains data on two of the 50 U.S. states.-->
<us_states>
  <state>
    <name>Delaware</name>
    <abbreviation>DE</abbreviation>
    <area>1954</area>
    <population>749000</population>
  </state>
```

```
    <state>
      <name>Pennsylvania</name>
      <abbreviation>PA</abbreviation>
      <area>44817</area>
      <population>1229600</population>
    </state>
  </us_states>
```

Comments on XML format

1. Don't be concerned about the colorization. If the file is created in the Visual Basic IDE and given the extension ".xml", Visual Basic will automatically color it.
2. The first line identifies the format of the file.
3. The second line is a comment and is treated like any Visual Basic comment. It may appear anywhere in the file, is colored green, and will be totally ignored by any program accessing the file. Comments start with `<!--` and end with `-->`.
4. A line such as `<area>1954</area>` is called an **element** and the bracketed entities are called **tags**. Specifically, `<area>` is the **start tag** for the element and `</area>` is the **end tag**. The two tags are identical except for the presence of a forward slash (/) following the less-than sign in the end tag. The word inside the brackets, in this case *area*, is called the **name of the element**. The text surrounded by the two tags, in this case 1954, is called the **content** of the element.
5. Element names are case sensitive. For instance, the tag `<Area>` is different than the tag `<area>`. Also, start and end tags must have the same case.
6. Element names should convey the meaning of the content of the element. Element names cannot start with a number or punctuation character and can contain numbers and characters. However, they cannot contain spaces and cannot start with the letters *xml*.
7. In the XML file above, the content of the elements *name*, *abbreviation*, *area*, and *population* is text. However, the content of an element can be other elements. For instance, in the XML file above,

   ```
   <state>
     <name>Delaware</name>
     <abbreviation>DE</abbreviation>
     <area>1954</area>
     <population>749000</population>
   </state>
   ```

   is such an element. Its start tag is `<state>` and its end tag is `</state>`. The elements that constitute its content are said to be its **children**. For instance, we say that *name* is a child of *state*. We also say that *state* is a **parent** of *name*, and that *name* and *abbreviation* are **siblings**.
8. The element *us_states* is called the **root element** of the file. An XML file can have only one root element.

LINQ to XML

LINQ can be used with XML files in much the same way as with CSV files—with three differences:

1. Instead of being loaded into an array, the file is loaded into an XElement object with a statement of the form

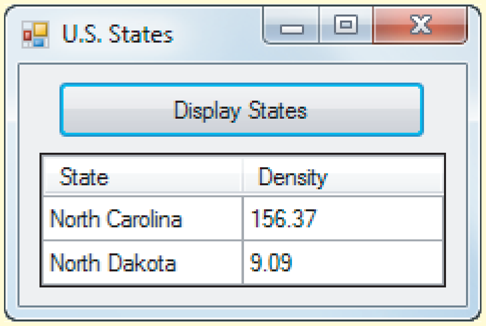
   ```
   Dim xmlElementName As XElement = XElement.Load(filespec)
   ```

2. The From clause references the elements to be considered—namely, the children of each child of the root element.
3. Instead of using the Split method to extract the value of a field, queries use an expression of the form `<childName>.Value`.

Example 1 The file USStates.xml extends the records shown above to all 50 states. The following program uses the file to display the population densities of the states that begin with the word *North*. The states are displayed in decreasing order of their densities.

```
Private Sub btnDisplay_Click(...) Handles btnDisplay.Click
  Dim stateData As XElement= XElement.Load("USStates.xml")
  Dim query = From st In stateData.Descendants("state")
              Let name = st.<name>.Value
              Let pop = CInt(st.<population>.Value)
              Let area = CInt(st.<area>.Value)
              Let density = pop / area
              Let formattedDensity = FormatNumber(density)
              Order By density Descending
              Where name.StartsWith("North")
              Select name, formattedDensity
  dgvStates.DataSource = query.ToList
  dgvStates.CurrentCell = Nothing
  dgvStates.Columns("name").HeaderText = "State"
  dgvStates.Columns("formattedDensity").HeaderText = "Density"
End Sub
```

[Run, and click on the button.]



State	Density
North Carolina	156.37
North Dakota	9.09

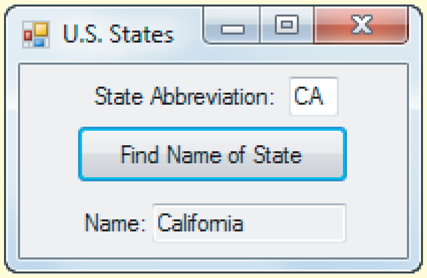
Example 2 The following rewrite of Example 7 in Section 8.1 uses an XML file rather than a CSV file. The variable *stateData* is given class-level scope so that the XML file will not be reread from disk each time a new state abbreviation is entered.

```
Dim stateData As XElement = XElement.Load("USStates.xml")

Private Sub btnFind_Click(...) Handles btnFind.Click
  Dim query = From st In stateData.Descendants("state")
              Let name = st.<name>.Value
              Let abbr = st.<abbreviation>.Value
              Where abbr = mtbAbbr.Text.ToUpper
              Select name
```

```
    If query.Count = 1 Then
      txtName.Text = query.First
    Else
      Dim str As String = " is not a valid state abbreviation."
      MessageBox.Show(mtbAbbr.Text.ToUpper & str, "Error")
      mtbAbbr.Clear()
      mtbAbbr.Focus()
    End If
  End Sub
```

[Run, enter a state abbreviation into the masked text box, and click on the button.]



Comments

1. The content of an element can contain any characters except for "`<`" and "`&`". These characters must be replaced with "`&lt;`" and "`&amp;`", respectively. For instance, the following two lines might appear in XML files:

```
<college>William &amp; Mary</college>
<inequality> x &lt; 7</inequality>
```

Practice Problems 8.3

1. Suppose the query in Example 1 is replaced by

```
Dim query = From st In stateData.Descendants("state")
            Let name = st.<name>.Value
            Let pop = st.<population>.Value
            Order By pop Descending
            Select name, pop
```

Explain why this query will not display the states in descending order of their population.

2. Consider Example 2. Suppose the last line of the query is changed to

```
Select name, abbr
```

What change would have to be made to the following line?

```
txtName.Text = query.First
```

EXERCISES 8.3

In Exercises 1 through 6, determine if the name is a proper name for an element of an XML file.

1. 7up 2. vice president 3. _77 4. Fred 5. xmlTitle 6. ?mark

In Exercises 7 through 10, determine if the expression is a proper element.

7. <begin>Hello</end>
8. <Team>Oakland Raiders</team>
9. <city>New York<city>
10. <first name>John</first name>
11. The first two lines of the file AgeAtInaug.txt are

```
George Washington,57
John Adams,61
```

where each record has two fields—name of president and his age when inaugurated. Create an XML file containing these two records.

12. The first two lines of the file Justices.txt are

```
Samuel,Alito,George W. Bush,NJ,2006,0
Henry,Baldwin,Andrew Jackson,PA,1830,1844
```

where each record has six fields—first name, last name, appointing president, the state from which the justice was appointed, year appointed, and year they left the court. (For sitting judges, the last field is set to 0.) Create an XML file containing these two records.

In Exercises 13 through 20, write a program to extract and display the requested information from the file USStates.xml.

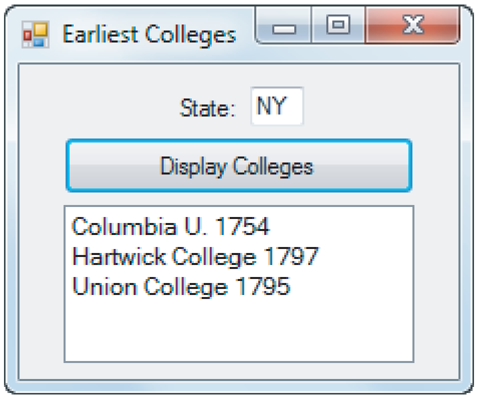
13. The total population of the United States in the year 2000.
14. The total area of the United States.
15. The population density of the United States.
16. The state (or states) with the longest name.
17. The states with area greater than 100,000 square miles. Display both the name and area of each state, with the states in decreasing order by area.
18. The states with population less than one million. Display both the name and population of each state, with the states in increasing order by population.
19. The state (or states) whose name contains the most different vowels.
20. The states whose abbreviation is different than the first two letters of their names. Display both the name and abbreviation of each state in alphabetical order by the abbreviation.

The file Colleges.xml contains data on the colleges founded before 1800. The first thirteen lines of the file are shown in Fig. 8.7. Use this file in Exercises 21 through 24.

```
<?xml version='1.0'?>
<!-- This file contains data on the earliest U.S. colleges -->
<colleges>
  <college>
    <name>Harvard U.</name>
    <state>MA</state>
    <yearFounded>1636</yearFounded>
  </college>
  <college>
    <name>William &amp; Mary</name>
    <state>VA</state>
    <yearFounded>1693</yearFounded>
  </college>
```

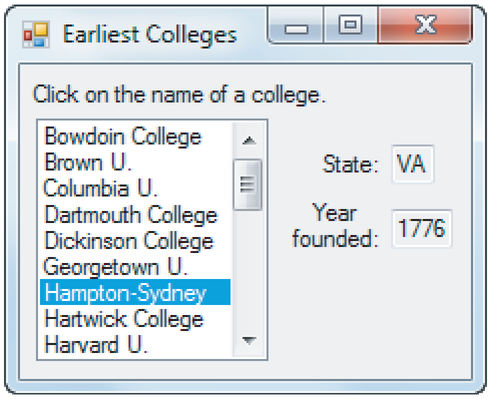
FIGURE 8.7 Beginning of the file Colleges.xml.

21. Write a program that displays the colleges alphabetically ordered (along with their year founded) that are in the state specified in a masked text box. See Fig. 8.8.



```
Columbia U. 1754
Hartwick College 1797
Union College 1795
```

FIGURE 8.8 Possible outcome of Exercise 21.

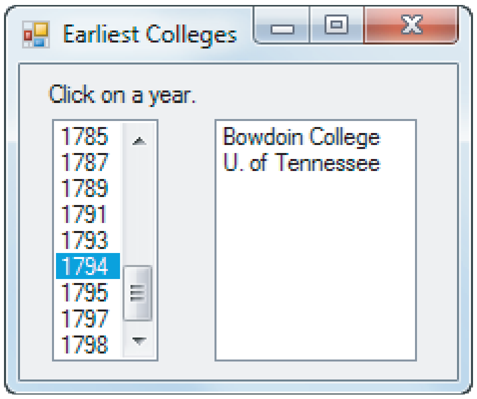


```
Bowdoin College
Brown U.
Columbia U.
Dartmouth College
Dickinson College
Georgetown U.
Hampton-Sydney
Hartwick College
Harvard U.
```

FIGURE 8.9 Possible outcome of Exercise 22.

22. Write a program that provides information about a college selected from a list box by the user. The colleges should be displayed in alphabetical order. See Fig. 8.9.

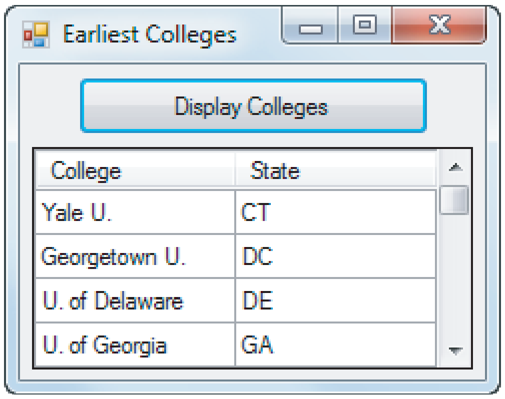
23. Write a program that fills a list box with the years before 1800 in which colleges were founded. When the user selects a year, the colleges founded that year should be displayed in another list box. See Fig. 8.10.



```
1785
1787
1789
1791
1793
1794
1795
1797
1798
```

```
Bowdoin College
U. of Tennessee
```

FIGURE 8.10 Possible outcome of Exercise 23.



College	State
Yale U.	CT
Georgetown U.	DC
U. of Delaware	DE
U. of Georgia	GA

FIGURE 8.11 Outcome of Exercise 24.

24. Write a program that displays the colleges and their states in a DataGridView control where the colleges are ordered alphabetically by their state abbreviations and secondarily by the year they were founded. See Fig. 8.11.

25. The CSV file Senate111.txt contains a record for each member of the 111th U.S. Senate. (The 111th U.S. Senate was installed in 2009.) Each record contains three fields—name, state, and party affiliation. Some records in the files are

```
John McCain,Arizona,R
Joseph Lieberman,Connecticut,I
Kirsten Gillibrand,New York,D
```

(a) Write a program that uses the file Senate111.txt and creates an XML file containing the same information.

(b) Write a program that uses the XML file from part (a) to display the names, states, and party affiliation of all the senators in a DataGridView ordered by their state. The two senators from each state should be ordered by their first names.

26. The file Top25HR.xml contains statistics for the top 25 home-run hitters of all time in major league baseball. The first nine lines of the file are shown in Fig. 8.12.

(a) Write a program that displays the contents of this file in a DataGridView control in descending order by the number of home runs hit.

(b) Write a program that uses the file Top25HR.xml and creates a CSV file containing the same information.

```
<?xml version='1.0'?>
<!-- This file contains data on the all-time top 25 home -->
<!-- run hitters in major league baseball prior to 2010. -->
<home_run_hitters>
  <player>
    <name>Babe Ruth</name>
    <atBats>8399</atBats>
    <homeRuns>714</homeRuns>
  </player>
```

FIGURE 8.12 Beginning of the file Top25HR.xml.

Solutions to Practice Problems 8.3

1. The problem is with the clause

```
Let pop = st.<population>.Value
```

Since there are no arithmetic operators or numeric conversion functions in the clause, local type inference will interpret *pop* to be a string variable. When the program is run, the first state listed will be Rhode Island, whose population is 998,000. The program will run as intended only if the clause is

```
Let pop = CDbl(st.<population>.Value)
```

2. The line would have to be changed to

```
txtName.Text = query.First.name
```

8.4 A Case Study: Recording Checks and Deposits

The purpose of this section is to take you through the design and implementation of a quality program for personal checkbook management. That a user-friendly checkbook management program can be written in less than four pages of code clearly shows Visual Basic's ability to improve the productivity of programmers. It is easy to imagine an entire finance program, similar to programs that have generated millions of dollars of sales, being written in only a few weeks by using Visual Basic!

Design of the Program

Though many commercial programs are available for personal financial management, they include so many bells and whistles that their original purposes—keeping track of transactions and reporting balances—have become obscured. The program in this section was designed specifically as a checkbook program. It keeps track of expenditures and deposits and produces a report. The program showcases many of the techniques and tools available in Visual Basic.