Chương 11

Tạo các lớp

- Tìm hiểu cách tạo lớp trong VB.NET.
- Tìm hiểu sự khác nhau giữa VB6 và VB.NET về các lớp.
- Tìm hiểu cách thêm các đặc tính vào lớp.
- Tìm hiểu cách thêm các phương thức vào lớp.

Lớp (class)

OOP (Object-Oriented Programming – Lập trình hướng đối tượng) là một phương thức thiết kế và xây dựng phần mềm. Nó là một sự phát triển logic tiếp theo lập trình có cấu trúc để cải thiện khả năng duy trì và khả năng dùng lại mã của bạn. Nói một cách khác, OOP là một phương thức thiết kế các thành phần phần mềm riêng lẻ (các lớp) với các ứng xử kết hợp (các phương thức) và các giới hạn dữ liệu (các đặc tính) để giúp bạn nối các

thành phần này lại với nhau tạo thành một ứng dụng hoàn chỉnh.

Ưu điểm của OOP là nó cho phép bạn lập nhóm dữ liệu thành các biến tinh xảo được chứa bên trong một cấu trúc trong chương trình. Dữ liệu này tách riêng và phân biệt với bất kỳ cấu trúc khác trong ứng dụng của bạn. Không có cấu trúc hoặc thủ tục nào có thể gây trở ngại với dữ liệu ấy trong bất kỳ cấu trúc hoặc thủ tục nào khác mà không đi qua một giao diện nhất định trên cấu trúc đó.

Mỗi đối tượng xác định tập thông điệp (giao diện) của nó mà nó sẽ đáp ứng. Các đối tượng khác có thể gửi các thông điệp tới các đối tượng này và cho chúng thực hiện một ứng xử nào đó hoặc nhận thông tin. Bằng cách này, các đối tượng không ảnh hưởng tới thông tin hoặc tiến trình của các đối tượng khác trực tiếp. Khi bạn xây dựng một lớp, bạn sẽ thấy cách xây dựng các thông điệp này bằng cách dùng các đặc tính và các phương thức.

Bảng 1 trình bày các thuật ngữ OOP mà bạn xem lại trước khi đọc tài liệu này.

Class – Lớp	Trình chứa dữ liệu và mã. Dữ liệu được coi như các đặc tính. Mã được coi như các phương thức.
Object – Đối tượng	Thể nghiệm của một lớp trong bộ nhớ. Một thể nghiệm được tạo bằng cách dùng câu lệnh Dim và từ khóa New.
Constructor – Cấu tử	Thủ tục được gọi ra tự động khi một đối tượng được lập thể nghiệm lần đầu tiên. Trong VB6, cấu tử được gọi là Class_Initialize. Trong VB.NET, cấu tử được gọi là New.
Destructor – Hủy tử	Thủ tục tự động được gọi ra khi

	một đối tượng bị hủy. Trong VB6, hủy tử được gọi là Class_Terminate. Trong VB.NET, hủy tử được gọi là Finalize.
Properties – Đặc tính	Dữ liệu được đưa vào một đối tượng và được một đối tượng khác truy tìm. Mỗi đối tượng duy nhất trong bộ nhớ có thể đặt các đoạn dữ liệu khác nhau vào cùng đặc tính có tên.
Method – Phương thức	Hoạt động có thể được thực hiện bởi một đối tượng.

Bảng 1: *Các thuật ngữ OOP.*

Sử dụng tốt các lớp

Lớp là điểm chủ yếu và linh hồn của ngôn ngữ hướng đối tượng. Bạn sẽ thấy việc sử dụng các lớp bất kỳ khi nào bạn viết ngay cả một chương trình đơn giản nhất trong VB.NET. Microsoft.NET Framework khai thác các lớp này nhiều hơn và bạn cũng vậy. Bên dưới là một số lớp thông dụng:

- Bao bọc một loạt các thao tác bạn thực hiện trên bảng CSDL, thí dụ, bổ sung, soạn thảo, xóa và truy tìm dữ liệu.
- Bao bọc tất cả các tham số bạn chuyển tới một lời gọi Windows API phức tạp. Điều này làm cho dễ hiểu hơn và đáp ứng cho các lập trình viên mới tiếp cận.
- Bao bọc một tập các điều tác và dữ liệu để xử lý các file văn bản, chẳng hạn như đọc, ghi và lập chỉ mục các dòng văn bản bên trong file.

- Bao bọc tất cả các biến toàn cục trong một chương trình vào các đặc tính bên trong một lớp.

Tạo lớp

Chúng ta hãy tạo một lớp biểu thị một dòng văn bản. Để làm điều này, bạn tạo một đặc tính để lưu giữ dòng văn bản và đặc tính chỉ đọc để cho trở lại chiều dài văn bản. Bạn cũng sẽ tạo một phương thức để cho trở lại từ đầu tiên trong dòng văn bản. Khi bạn thực hiện tất cả các bước này, bạn sẽ tìm hiểu cách tạo một lớp chính xác. Bạn sẽ xây dựng một form như được trình bày ở hình 1 để kiểm tra lớp Line khi bạn xây dựng nó.

Hình 1: *Form này cho phép bạn kiểm tra từng đặc tính và phương thức của lớp Line.*

Các bước tạo form

1. Mở Visual Studio.NET.
2. Lựa Visual Basic Project từ hiển thị cây ở bên trái màn hình.
3. Lựa Windows Application là template của project.

4. Đặt tên của ứng dụng là **ClassCreation**.

5. Tô sáng form có tên Form1.vb trong cửa sổ Solution Explorer và đặt lại tên là **frmLineTest.vb**. Bạn có thể đặt lại tên form bằng cách nhấp nó trong cửa sổ Solution Explorer, nhấp nút chuột phải và chọn Rename từ menu.

6. Tạo form được trình bày ở hình 1 bằng cách thêm các control thích hợp và xác lập các đặc tính của các control này như được trình bày ở bảng 2.

Kiểu control	Đặc tính	Giá trị
Label	Name	Label1
	Text	Line of Text
TextBox	Name	txtLine
	Text	The rain in Spain stays mainly in the plain
CommandButton	Name	btnDisplay
	Text	Display Length
TextBox	Name	txtLength
	ReadOnly	True
CommandButton	Name	btnGetWord
	Text	Get Word
TextBox	Name	txtWord
	Text	
	ReadOnly	True

Bảng 2: *Sử dụng các control này để xây dựng form kiểm tra lớp Line.*

Tạo lớp Line

Tạo lớp Line bằng các bước sau:

1. Lựa Project ➢ Add Class để hiển thị hộp thoại Add New Item.

2. Xác lập đặc tính Name là **Line.vb** và nhấp OK.

3. Bây giờ bạn sẽ thấy một file mới xuất hiện trong project của bạn và cửa sổ mã bên trong môi trường Visual Studio.NET. Trong cửa sổ Code, sẽ có mã như sau:

```
Public Class Line

End Class
```

Tất cả các đặc tính và các phương thức bạn tạo cho lớp này phải được nhập vào giữa các dòng mã này.

Tạo các đặc tính

Để tạo một đặc tính bên trong một lớp, bạn có thể tạo biến Public hoặc có thể tạo biến Private và thể hiện biến riêng (private) bằng cách dùng câu lệnh Property. Còn có một số lý do tại sao bạn chỉ cần thể hiện các đặc tính bằng câu lệnh Property.

- Bạn có thể tạo đặc tính chỉ đọc hoặc chỉ ghi ngược lại với biến Public để chỉ có thể đọc hoặc ghi.
- Bạn có thể thêm xử lý lỗi bên trong câu lệnh Property để kiểm tra các giá trị không hợp lệ được xác lập. Bạn không thể kiểm tra các giá trị không hợp lệ khi xác lập biến

Public vì không có mã chạy để đáp ứng xác lập biến Public.

- Bạn có thể thể hiện các giá trị đã tính như các đặc tính dù cho chúng không được lưu trữ như dữ liệu thực bên trong lớp. Một thí dụ về điều này là đặc tính Length. Có thể bạn không muốn lưu trữ chiều dài dòng văn bản, khi nó có thể thay đổi.

Bây giờ tạo hai đặc tính có tên Line và Length cho lớp Line. Đầu tiên bạn tạo một biến riêng để lưu giữ dòng dữ liệu bạn lưu trữ bên trong lớp. Kế tiếp, bạn sẽ tạo các câu lệnh Property cho hai đặc tính mới này. Sửa đổi lớp trong project của bạn như mã được trình bày bên dưới.

```
Public Class Line

    Private mstrLine As String

    Property Line() As String
        Get
           Return mstrLine
        End Get
        Set(ByVal Value As String)
            mstrLine = Value
        End Set
    End Property

    ReadOnly Property Length() As Integer
       Get
           Return mstrLine.Length
       End Get
    End Property
End Class
```

Cú pháp để tạo đặc tính khá dễ, nhưng hoàn toàn khác với những gì bạn đã thực hiện trong VB6. Đầu tiên sử dụng từ khóa Property được theo sau bởi tên của đặc tính kế tiếp là kiểu dữ liệu của đặc tính này sẽ cho trở lại hoặc được xác lập. Để cho trở lại dữ liệu được chứa trong biến riêng mstrLine, bạn sử dụng

khối Get...End Get. Khối này giống hàm Property Get cũ trong VB6. Khối mã này thi hành mã bất kỳ ở giữa khối Get...End Get và cho trở lại giá trị chuỗi. Trong VB.NET, bạn có thể sử dụng câu lệnh Return để cho trở lại dữ liệu.

Khối Set...End Set giống thủ tục Property Let cũ bạn dùng trong VB6. Khối này nhận tham số có tên Value. Value là tên mặc định được Visual Studio tạo tự động cho bạn. Bạn có thể thay đổi nó nếu muốn. Bạn có thể dùng giá trị được chuyển tới khối Set này và đặt nó vào biến riêng của mình. Đây là điểm, nơi bạn có thể đặt bất kỳ mã xử lý lỗi nào bạn muốn. Thí dụ, bạn có thể kiểm tra chắc chắn dữ liệu được chuyển tới tham số Value không được điền vào với một từ như Test. Nếu một từ không hợp lệ được chuyển vào, bạn có thể chọn phát sinh lỗi là nó không phải giá trị hợp lệ cho đặc tính này.

Các đặc tính chỉ đọc

Ở mã trên, bạn cũng đã tạo đặc tính chỉ đọc bằng cách dùng Keyword ReadOnly ở trước câu lệnh Property. Khi sử dụng từ khóa này, Visual Studio.NET thêm khối Get...End Get vào câu lệnh Property. Thực ra, nếu bạn cố thêm khối Set...End Set, bạn sẽ nhận một lỗi của trình biên dịch. Nếu bạn cần tạo một đặc tính chỉ đọc trong VB6, bạn không tạo thủ tục Property Let, nhưng không có lỗi nào được phát sinh nếu bạn bỏ sót nó. Đúng ra bạn không thể xác lập đặc tính vào thời gian chạy.

Trong project bạn đã tạo, bạn có một lớp và một form. Bây giờ bạn sẽ viết mã vào form để tạo một đối tượng Line mới, đặt một dòng văn bản vào đặc tính Line của đối tượng và sau đó đặt chiều dài vào đặc tính Text của văn bản txtLength trên form của bạn.

1. Nhấp đôi form frmLineTest trong cửa sổ Solution Explorer để đưa form vào chế độ thiết kế.

2. Nhấp đôi nút có các từ Display Length ở trên nó. VB.NET tạo thủ tục sự kiện btnDisplay_Click cho bạn trong mã đằng sau form này. Bạn chỉ cần điền các dòng mã được trình bày bên dưới vào thân của thủ tục.

```
Private Sub btnDisplay_Click( _
  ByVal sender As Object, _
  ByVal e As System.EventArgs) Handles btnDisplay.Click

    Dim oLine As Line

    oLine = New Line()

    oLine.Line = txtLine.Text
    txtLength.Text = oLine.Length.ToString()
End Sub
```

Bên trong thủ tục sự kiện này, bạn tạo một biến có tên oLine bằng cách dùng câu lệnh Dim. Biến này được định nghĩa như một tham chiếu lớp Line. Thực ra bạn tạo tham chiếu đối tượng mới bằng cách dùng từ khóa New, như được trình bày ở dòng kế tiếp sau câu lệnh Dim. Từ khóa New phải được theo sau bằng tên của lớp bạn muốn lập thể nghiệm. Một sự khác biệt nữa giữa VB.NET và VB6 là bạn không còn dùng từ khóa Set khi tạo một đối tượng mới.

Trong VB.NET, bạn được phép kết hợp hai dòng này thành một như được trình bày ở mã bên dưới.

```
Dim oLine As Line = New Line()
```

VB.NET (và tất cả ngôn ngữ .NET) cho phép bạn khai báo và khởi tạo một biến bất kỳ trên cùng dòng. Trong VB6, bạn không thể làm điều này, vì câu lệnh Dim không phải là dòng mã khả thi. Trong VB.NET, Dim là một dòng mã khả thi, vì vậy cú pháp này hoàn toàn hợp lệ.

Bây giờ chúng ta hãy khảo sát hai dòng kế tiếp của thủ tục sự kiện này:

```
oLine.Line = txtLine.Text
txtLength.Text = oLine.Length.ToString()
```

Dòng đầu tiên thiết lập đặc tính Line trong đối tượng của bạn bằng giá trị được chứa trong đặc tính Text của hộp văn bản txtLine trên form. Nó chuyển dữ liệu trong đặc tính Text tới biến Value ở khối Set...End Set trong đối tượng Line.

Bây giờ bạn sẵn sàng báo lại chiều dài của chuỗi được chứa trong hộp văn bản txtLine. Nhớ là bạn xác lập giá trị của hộp văn bản này bằng chuỗi "The rain in Spain stays mainly in the plain". Đây là giá trị được chứa trong biến mstrLine bên trong đối tượng của bạn. Bạn có thể gọi ra đặc tính Length trên đối tượng dòng và nó sẽ cho trở lại chiều dài của chuỗi cá biệt này. Vì đặc tính Length là một giá trị Integer, bạn cần chuyển đổi giá trị đó thành chuỗi trước khi có thể đặt nó vào đặc tính Text của hộp văn bản txtLength. Bạn hoàn tất điều này bằng cách áp dụng phương thức ToString vào đặc tính Length.

Tạo đặc tính chỉ ghi

Đặc tính chỉ ghi là một đặc tính xác định khối Set...End Set, chứ không phải Get...End Get. Bạn được phép đặt dữ liệu vào đặc tính này chỉ vào thời gian chạy và không thể truy tìm dữ liệu từ đặc tính này.

Nếu bạn muốn mở rộng lớp Line này để có thể đọc dòng văn bản từ trong một file trên đĩa, bạn có thể chuyển tên file vào lớp này. Bạn có thể thực hiện điều này bằng cách dùng đặc tính chỉ ghi. Sau là một thí dụ về đặc tính chỉ ghi thể hiện như thế nào.

```
WriteOnly Property FileName() As String
    Set(ByVal Value As String)
        mstrFileName = Value
```

```
    End Set
End Property
```

Cú pháp này cũng khác với VB6, trong đó bạn dùng Keyword WriteOnly như tiền tố của câu lệnh Property. Visual Studio.NET tự động tạo khối Set...End Set cho bạn. Nếu bạn cố thêm khối Get...End Get, trình biên dịch đưa ra một lỗi.

Tạo phương thức

Phương thức trong một lớp có thể là một thủ tục để thực hiện sự sắp xếp điều tác dữ liệu nào đó bên trong lớp. Hoặc một phương thức có thể là một hàm để thực hiện một điều tác nào đó trên dữ liệu và cho dữ liệu trở lại từ lớp ấy. Để có thể gọi một phương thức từ một thể nghiệm của lớp này, phương thức ấy phải được khai báo là Public. Nếu phương thức được khai báo là Private, thì chỉ các phương thức khác bên trong lớp ấy mới có thể gọi phương thức đó. Việc tạo phương thức trong VB.NET hoàn toàn giống như trong VB6.

Chúng ta hãy tạo một phương thức trong lớp Line để tách dòng văn bản được chuyển vào lớp thành mỗi từ tách riêng. Phương thức này là một hàm cho trở lại từ đầu tiên được chứa trong dòng văn bản bạn chuyển vào. Nếu bạn đã dùng chuỗi được cho khi tạo form, thì dòng văn bản sẽ là "The rain in Spain stays mainly in the plain". Trong trường hợp này, từ đầu tiên được cho trở lại sẽ là "The."

1. Mở lớp Line.vb ở chế độ thiết kế.

2. Ngay dưới các câu lệnh Property, tạo phương thức sau.

```
Public Function GetWord() As String
    Dim astrWords() As String
```

```
        astrWords = Split(mstrLine, " ")

        Return astrWords(0)
    End Function
```

Phương thức GetWord chuyển đổi chuỗi thành một mảng các từ riêng biệt bằng cách dùng hàm Split. Hàm Split có từ VB6 và nó được chuyển một chuỗi và một dấu định biên để tìm. Split bắt đầu với kí tự đầu tiên trong chuỗi và tiếp tục tìm qua chuỗi cho tới khi tìm thấy dấu định biên. Một khi tìm thấy, nó đưa cả chuỗi tới điểm đó và tạo một phần tử mảng mới. Sau đó nó tiếp tục xử lý chuỗi theo cách này cho tới khi toàn bộ chuỗi đã được đưa vào các phần tử mảng.

Sau khi chuyển đổi chuỗi thành một mảng, bạn có thể cho trở lại một từ bất kỳ trong câu lệnh bạn chuyển vào lớp Line. Thí dụ này chỉ cho trở lại từ đầu tiên bằng cách xử lý phần tử đầu tiên của mảng. Trong VB.NET, tất cả các mảng có cơ số zêrô giống như mặc định trong VB6. Sự khác nhau ở đây là bạn không thể làm cho bất kỳ thứ gì khác hơn mảng cơ số zêrô trong VB.NET và bạn có thể thực hiện trong VB6.

Lưu ý:

Một cách khác bạn có thể viết mã để tách các từ trong biến mstrLine vào một mảng là dùng phương thức Split trên đối tượng String như được trình bày ở bên dưới.

```
astrWords = mstrLine.Split(" ".ToCharArray())
```

Thực hiện theo các bước bên dưới để thử phương thức GetWord mới này bạn vừa viết.

1. Mở form frmLineTest ở chế độ thiết kế.

2. Nhấp đôi nút có các từ "Get Word" trên nó. Điều này mở cửa sổ mã với thủ tục sự kiện btnGetWord_Click được tạo cho bạn.

3. Thêm mã sau vào thủ tục sự kiện này như được trình bày ở thí dụ bên dưới.

```
Private Sub btnGetWord_Click(ByVal sender As Object,_
 ByVal e As System.EventArgs)

    Dim oLine As Line = New Line()

    oLine.Line = txtLine.Text
    txtWord.Text = oLine.GetWord()

End Sub
```

Đầu tiên thủ tục sự kiện này tạo một thể nghiệm mới của lớp Line vào biến đối tượng oLine. Kế tiếp bạn thiết lập đặc tính Line trên lớp line bằng cách nhận giá trị từ đặc tính Text của hộp văn bản txtLine. Cuối cùng bạn gọi phương thức GetWord mới và nó cho trở lại giá trị chuỗi. Giá trị này được gán vào đặc tính Text của hộp văn bản txtWord bạn đã tạo trên form frmLineTest trước đây ở phần này.

Chuyển dữ liệu vào một cấu tử

Trong VB6, bạn không thể chuyển dữ liệu khi bạn đã tạo một thể nghiệm mới của một lớp. Điều này có nghĩa là bạn phải tạo đối tượng, phải nhớ gọi phương thức khởi tạo và chuyển dữ liệu tới phương thức đó. Nếu bạn quên khởi tạo đối tượng, đối tượng của bạn không thành công. Mặc dù có thể bạn nhớ gọi phương thức khởi tạo, nếu các lập trình viên khác dùng lớp của bạn, họ có thể không biết cách gọi nó.

Vấn đề này đã được giải quyết trong VB.NET, vì bây giờ phương thức cấu tử được chuyển dữ liệu. Phương thức cấu tử được gọi là New trong VB.NET. Theo mặc định, VB.NET tạo phương thức New này cho bạn mà bạn không phải làm gì. Bạn sẽ không thấy mã này bên trong module mã của mình, nhưng nó hoàn toàn ở đó. Nếu bạn muốn chuyển mã nào đó vào phương thức New này, bạn phải tạo nó cụ thể.

Bạn có thể đặt cấu tử New bất kỳ nơi nào bên trong lớp của mình, nhưng bạn nên chọn một vị trí chuẩn, chẳng hạn như, luôn luôn gần trên đầu định nghĩa lớp của bạn. Ở phần bài tập kế tiếp, bạn tạo một cấu tử New để nhận giá trị String. Bằng cách thực hiện điều này, bạn sẽ có thể khai báo, khởi tạo một biến đối tượng và thiết lập dữ liệu cho đối tượng Line chỉ bằng một bước.

1. Mở lớp Line.vb trong đối tượng của bạn.

2. Thêm Sub New, như được trình bày ở dưới vào lớp Line của bạn.

```
Public Class Line
    Private mstrLine As String

    Public Sub New(ByVal Value As String)
        mstrLine = Value
    End Sub
```

Cấu tử New là một thủ tục bạn viết để nhận một hoặc nhiều tham số. Trong mã bạn vừa viết, bạn đã tạo một tham số được gọi là Value. Sau đó bạn đã dùng giá trị ấy và gán nó vào biến riêng mstrLine. Bây giờ bạn cần thay đổi các thủ tục trong form của mình, vì vậy bạn chuyển giá trị từ hộp văn bản txtLine vào cấu tử.

1. Mở form frmLineTest.vb của bạn.

2. Bạn sẽ phải thay đổi cả thủ tục sự kiện btnDisplay_Click lẫn các thủ tục sự kiện btnGetWord_Click để khai báo biến oLine như được trình bày ở dưới.

```
Dim oLine As Line = New Line(txtLine.Text)
```

Từ khóa New báo VB.NET tạo một thể nghiệm Line mới và chuyển dữ liệu được chứa trong đặc tính Text của hộp văn bản tới cấu tử New.

Bạn cần thay đổi cả hai thủ tục sự kiện trước khi bạn có thể chạy project. Bằng cách thay đổi thủ tục New để nhận tham số, bất kỳ khai báo nào của lớp Line bây giờ phải chuyển dữ liệu nào đó khi khai báo đối tượng.

Sau khi thay đổi các thủ tục sự kiện, bạn sẵn sàng chạy thử chương trình lại để chắc chắn bạn đã thực hiện chính xác.

1. Ấn **F5** để chạy ứng dụng này.
2. Nếu bạn nhận các lỗi bất kỳ, sửa lỗi và thử lại.

Sử dụng các lớp cho tất cả các tác vụ

Nên tạo các lớp cho các tác vụ lập trình của bạn. VB.NET sẽ vẫn để bạn tạo các hàm độc lập và các thủ tục bên trong một module. Nhưng có một vài ưu điểm về sử dụng các lớp (và các đối tượng) hơn các kĩ thuật lập trình có cấu trúc truyền thống:

- Các tính toán hoặc các điều tác (được gói riêng) vào một giao diện dễ dùng. Điều này giúp các lập trình viên mới tiếp cận khắc phục các vấn đề dễ dàng mà không phải biết mọi thứ hoạt động như thế nào. Nó cũng giúp các lập trình viên giảm thời gian phát triển đáng kể.

- Nó dễ dàng thay đổi các công việc bên trong một lớp để tạo cho nó thực hiện tốt hơn. Một chương trình bất kỳ sử dụng lớp này sẽ tăng được hiệu suất.

- Có thể có nhiều thể nghiệm của một lớp hoạt động trong bộ nhớ cùng một lúc. Nó không cần tạo nhiều biến Public khi bạn muốn so sánh nhiều giá trị từ một nguồn dựa trên các giá trị từ một nguồn khác.

- Có thể thừa kế từ một lớp và mở rộng tính năng mà không phải thay đổi lớp gốc. Sự thừa kế được đề cập ở một tài liệu tách riêng, có tên "Thừa kế từ lớp cơ sở".

Sự khác biệt giữa VB.NET và VB6

Có một số sự khác biệt về cách các lớp được tạo trong VB.NET với VB6. Bên dưới là một danh sách về các sự khác biệt chính.

- Bây giờ bạn sử dụng Public Class...End Class để định nghĩa một lớp. Trong VB6, mỗi lớp phải ở trong một file tách riêng của nó.

- Bạn có thể có nhiều lớp trên mỗi file.

- Các khai báo của các đặc tính hoàn toàn khác với VB6.

- Bạn không còn dùng từ khóa Set để lập thể nghiệm mới của một lớp hoặc thiết lập tham chiếu đối tượng.

- Bây giờ bạn có thể chuyển các tham số tới cấu tử của lớp

Tóm tắt

Bạn đã tạo lớp có đặc tính đọc/ghi, đặc tính chỉ đọc và một phương thức. Bạn cũng đã chuyển dữ liệu vào cấu tử của lớp này. Như đã nói ở đầu chương này, OOP thực hiện khả năng kết hợp các thành phần của phần mềm lại với nhau để giúp bạn tạo các ứng dụng dễ dàng hơn. Bằng cách tạo lớp Line, bạn không phải nhớ chi tiết cách tính chiều dài của một dòng văn bản hoặc sử dụng hàm Split để cho trở lại từ đầu tiên của dòng văn bản. Thay vào đó, bạn sử dụng lớp Line, chuyển cho nó dòng ấy và sử dụng các đặc tính và các phương thức để thực hiện các điều này cho bạn. Đây là một cách tiếp cận lập trình rất tốt.

Câu hỏi ôn tập

1. Đúng hay sai: Bạn chỉ có thể có một lớp trong một file?
2. Bạn dùng từ khóa nào để tạo đặc tính chỉ đọc?
3. Bạn dùng từ khóa nào để tạo đặc tính chỉ ghi?
4. Tên của thủ tục bạn viết cho cấu tử lớp là gì?

Trả lời câu hỏi ôn tập

1. Sai.
2. ReadOnly.
3. WriteOnly.
4. New.

Chương 12

Quá tải các phương thức

- Tìm hiểu tại sao quá tải một phương thức.
- Tạo các phương thức quá tải.
- Tạo các cấu tử của riêng bạn cho một lớp

Quá tải

Bạn đã bao giờ muốn tạo một nhóm các hàm mà về thực chất mỗi hàm thực hiện cùng một thứ chỉ với các tham số của chúng được thay đổi? Bạn có thể thực hiện điều này bằng cách dùng các tham số tùy chọn hoặc bằng cách đổi tên hàm. Cả hai kĩ thuật đều có tác dụng, nhưng chúng không hay lắm. Trong ngôn ngữ OOP, như Visual Basic.NET, bạn được phép tạo các phương thức trong một lớp có cùng tên nhưng các danh sách đối số khác nhau. Visual Basic.NET có thể tính được phương thức nào gọi trong thời gian biên dịch dựa vào các kiểu tham số bạn chuyển. Kĩ thuật này được gọi là *overload* (quá tải) một phương thức. Ưu điểm của việc sử dụng cùng tên là giao diện người dùng luôn luôn nhất quán; chỉ có các nhập liệu khác nhau. Tính năng thay đổi theo mỗi phương thức quá tải mới.

Lớp Line bạn đã tạo trong tài liệu "Creating Classes" sử dụng phương thức GetWord để cho trở lại từ đầu tiên trong dòng văn bản của nó, bạn sẽ tạo hai phiên bản của phương thức GetWord này. Ở phiên bản thứ nhất, bạn sẽ chuyển vào một số biểu thị vị trí của từ bạn muốn truy tìm từ dòng ấy. Ở phiên bản thứ hai, bạn sẽ chuyển vào chuỗi tìm kiếm và phương thức sẽ cho trở lại từ đầu tiên trong dòng văn bản chứa chuỗi đó.

Signature của một phương thức bao gồm tên của nó, các tham số và kiểu trả lại — nói cách khác, các đối số phân biệt "hương vị" của phương thức này với phương thức kia. Các signature của ba phương thức này thể hiện như sau:

```
Function GetWord() As String
Function GetWord(ByVal intPosition As Integer) As
String
Function GetWord(ByVal strSearch As String) As String
```

Ở mỗi phương thức trên, tên của phương thức (GetWord) vẫn giống như nhau, nhưng danh sách đối số khác nhau. Trình biên dịch có thể phân giải từng tên dựa vào "các signature" khác nhau này. Lưu ý, giá trị trả lại của mỗi phương thức đều giống nhau. Không nhất thiết phải duy trì giá trị trả lại giống nhau, nhưng thông thường bạn sẽ không thay đổi giá trị trở lại trên các phương thức quá tải. Chỉ thay đổi kiểu trả lại trên một phương thức sẽ không quá tải một phương thức. Nếu bạn cố làm điều này, Visual Basic.NET cho bạn một thông báo lỗi cho biết không thể quá tải một phương thức bằng cách thay đổi kiểu trả lại. Bạn phải thay đổi ít nhất một đối số cho một phương thức để có signature khác và được quá tải.

Lưu ý:

Signature của một phương thức không bao gồm giá trị trả lại. Việc trả lại một kiểu dữ liệu khác từ một phương thức không đủ để trình biên dịch phân biệt giữa hai phương thức cùng tên.

Các lý do quá tải một phương thức

Việc quá tải một phương thức cho phép bạn duy trì giao diện nhất quán, nhưng cho phép bạn thay đổi tính năng dựa vào kiểu dữ liệu được chuyển vào một thủ tục cá biệt. Bạn sẽ thấy việc dùng cùng tên sẽ giúp bạn nhớ được thủ tục làm gì, trái lại phải theo kịp các tên mới hoặc quy ước đặt tên để giúp bạn duy trì mọi thứ nhất quán.

Tạo các phương thức quá tải

Chúng ta hãy tạo hai phương thức GetWord mới để quá tải phương thức GetWord gốc trong lớp Line. Phương thức GetWord mới đầu tiên sẽ nhận một tham số integer và cho trở lại từ ở vị trí bạn chuyển tới phương thức. Phương thức GetWord thứ hai sẽ nhận tham số chuỗi và sẽ tìm chuỗi đó trong dòng văn bản ban đầu. Nếu tìm thấy chuỗi tìm kiếm, từ ở vị trí đó sẽ được trả lại từ phương thức này.

Hình 1: *Màn hình được dùng để kiểm tra việc quá tải của các phương thức.*

1. Tạo project Windows Application mới bằng cách dùng Visual Studio.NET. Hoặc nếu bạn đọc tài liệu có tên "Tạo các lớp", bạn có thể sử dụng cùng project và theo các bước bên dưới để thay đổi WinForm làm việc với project này.

2. Tạo form được trình bày ở hình 1, bằng cách thêm các control thích hợp và xác lập các đặc tính của các control này như được trình bày ở bảng 1.

Kiểu Control	Đặc tính	Giá trị
Label	Name	Label1
	Text	Line of Text
TextBox	Name	txtLine
	Text	The rain in Spain stays mainly in the plain
CommandButton	Name	btnDisplay
	Text	Display Length

GroupBox	Name	fraWord
	Text	Find Occurrence of Word
Label	Name	Label2
	Text	Enter Word
TextBox	Name	txtWordNum
	Text	1
CommandButton	Name	btnGetWord
	Text	Get Word
TextBox	Name	txtWord
	Text	
	ReadOnly	True
GroupBox	Name	fraWord2
	Text	Find Word
Label	Name	Label3
	Text	Enter Word to Find
TextBox	Name	txtSearch
	Text	Spain
CommandButton	Name	btnGetWord2
	Text	Get Word
TextBox	Name	txtWord2
	Text	
	ReadOnly	True

Bảng 1: *Sử dụng các control này để xây dựng form kiểm tra phương thức GetWord quá tải trong lớp Line.*

3. Thêm Class mới bằng cách lựa Project ➢ Add Class từ menu Visual Studio.NET.

4. Khi được nhắc, xác lập Name của lớp này là Line.vb và nhấp nút OK.

5. Thêm mã sau để tạo lớp Line này:

```
Public Class Line
    Private mstrLine As String

    Property Line() As String
        Get
            Return mstrLine
        End Get
        Set
            mstrLine = Value
        End Set
    End Property

    ReadOnly Property Length() As Integer
        Get
            Return mstrLine.Length
        End Get
    End Property

    Public Function GetWord() As String
        Dim astrWords() As String

        astrWords = mstrLine.Split(" ".ToCharArray())

        Return astrWords(0)
    End Function
End Class
```

6. Thay đổi phương thức GetWord và thêm từ khóa **Overloads**, như được trình bày ở mã bên dưới.

```
Public Overloads Function GetWord() As String
```

Từ khóa Overloads được đòi hỏi khi bạn tạo một số phiên bản của cùng phương thức. Bạn sẽ tạo thêm các phiên bản của phương thức GetWord kế tiếp.

7. Tạo phương thức GetWord sẽ nhận giá trị Integer và vẫn trả lại một chuỗi. Khai báo ở mã như bên dưới.

```
Public Overloads Function GetWord(_
 ByVal intPosition As Integer) As String
    Dim astrWords() As String

    astrWords = mstrLine.Split(" ".ToCharArray())

    If intPosition > astrWords.Length Then
        Return ""
    Else
        Return astrWords(intPosition - 1)
    End If
End Function
```

Phương thức GetWord chuyển đổi chuỗi thành một mảng các từ riêng biệt bằng cách dùng phương thức Split của lớp String. Vì biến mstrLine được định nghĩa như một đối tượng chuỗi, nên nó có phương thức Split. Phương thức Split này dùng mảng các dấu định biên Char. Thay vì khai báo biến mảng Char và đặt nó với một kí tự trắng, bạn có thể khai thác kĩ thuật được gọi là *boxing*.

Boxing là tiến trình chuyển đổi nguyên nghĩa, như một kí tự trắng thành đối tượng bao hàm ý nghĩa (trong trường hợp này là đối tượng chuỗi) cho nguyên nghĩa đó. Một khi boxing này xảy ra, bạn có thể áp dụng phương thức ToCharArray của đối tượng chuỗi mới tạo để trả lại mảng một kí tự. Mảng phần tử đơn giản này được chuyển tới phương thức Split dùng như dấu định biên để tách rời các từ trong chuỗi.

Bây giờ bạn có một mảng các từ, bạn có thể cho trả lại từ tương ứng với số bạn đã chuyển vào phương thức này. Đương nhiên, bạn phải luôn luôn chắc chắn là số bạn đã chuyển không lớn hơn số từ trong mảng.

1. Mở form frmLineTest.vb.

2. Nhắp đôi vào nút Get Word đầu tiên trên form và thêm mã sau vào sự kiện Click.

```
Protected Sub btnGetWord_Click(ByVal sender As Object,
_
ByVal e As System.EventArgs) Handles btnGetWord.Click
    Dim oLine As Line = New Line()

    oLine.Line = txtLine.Text
    txtWord.Text = _
     oLine.GetWord(txtWordNum.Text.ToInt32())
End Sub
```

Vì phiên bản của phương thức GetWord này cần có kiểu dữ liệu Integer, bạn phải dùng phương thức ToInt32 để chuyển đổi giá trị trong đặc tính Text thành Integer.

3. Để chạy project, đặt một số vào hộp văn bản đầu tiên và nhấp nút Get Word. Tùy theo số bạn đã gõ vào, bây giờ một từ sẽ xuất hiện trong hộp văn bản chỉ đọc bên nút đầu tiên này.

4. Thêm một phiên bản của phương thức GetWord mới khác để nhận chuỗi tìm kiếm như một đối số. Phiên bản GetWord này cần tìm chuỗi đó bên trong dòng văn bản và cho trở lại từ, nơi tìm thấy chuỗi đó.

```
Public Overloads Function GetWord(_
 ByVal strSearch As String) As String
    Dim astrWords() As String
    Dim intLoop As Integer

    astrWords = mstrLine.Split(" ".ToCharArray())

    For intLoop = 0 To astrWords.Length - 1
        If  astrWords(intLoop).IndexOf(strSearch)  >  -1
Then
```

```
            Exit For
        End If
    Next
    If intLoop < astrWords.Length Then
        Return astrWords(intLoop)
    End If
End Function
```

Ở phương thức này, sau khi bạn tạo mảng các từ, bạn vòng lặp qua mảng và kiểm tra xem chuỗi tìm kiếm bạn đã chuyển có ở bất kỳ từ nào trong mang hay không. Để thực hiện điều này, bạn dùng phương thức IndexOf của chuỗi để xem chuỗi bạn đang tìm có phải là một phần của chuỗi hay không. Nếu phương thức IndexOf tìm thấy chuỗi đó, nó sẽ cho trở lại một số lớn hơn –1. Một khi bạn tìm thấy chuỗi, vòng lặp For thoát khỏi và từ ở vị trí đó được cho trở lại từ phương thức này.

1. Mở form frmLineTest.vb.

2. Nhắp đôi nút Get Word thứ hai và thêm mã sau vào thủ tục sự kiện Click cho nút này.

```
Protected Sub btnGetWord2_Click(ByVal sender As Object,
_
ByVal e As System.EventArgs) Handles btnGetWord2.Click
    Dim oLine As Line = New Line()

    oLine.Line = txtLine.Text
    txtWord2.Text = _
     oLine.GetWord(txtSearch.Text)
End Sub
```

Trong thủ tục sự kiện btnGetWord2_Click, bạn chuyển chuỗi tìm kiếm tới phương thức GetWord. Đặc tính Text trả lại kiểu dữ liệu String, vì vậy phương thức GetWord nhận kiểu dữ liệu String là phương thức được gọi.

Để chạy project, đặt một vài kí tự vào hộp văn bản thứ hai trên form này, sau đó nhấp nút Get Word thứ hai. Tùy theo các kí tự bạn đã gõ vào, bạn sẽ thấy một từ xuất hiện trong hộp văn bản bên nút ấy hoặc một chuỗi rỗng.

Quá tải Sub New

Khi bạn lập thể nghiệm một đối tượng mới, Visual Basic.NET gọi phương thức New. Nếu bạn không tự viết phương thức New, Visual Basic.NET xây dựng nó cho bạn. Nếu bạn muốn khởi tạo bất kỳ các biến nào bên trong một đối tượng vào thời điểm bạn tạo đối tượng mới, bạn có thể viết một phương thức New được quá tải. Sau đó bạn có thể chuyển một hoặc nhiều giá trị vào phương thức New này. Bạn sẽ thấy việc quá tải phương thức New là việc quá tải bạn thường sử dụng nhất.

Mở module lớp Line.vb và thêm mã bên dưới.

```
Public Sub New()
    ' Do Nothing
End Sub

Public Sub New(ByVal strLine As String)
    mstrLine = strLine
End Sub
```

Phương thức New đầu tiên bạn đã tạo xử lý việc tạo đối tượng Line mới khi bạn không chuyển bất kỳ các tham số nào vào nó. Không giống như các phương thức thông thường, bạn không thể dùng từ khóa Overloads trên cấu tử .NET cho là bạn sẽ quá tải phương thức này, vì vậy nó không đòi hỏi phải có từ khoa này.

Phương thức New khác nhận đối số chuỗi. Đối số chuỗi này sẽ được chuyển khi đối tượng được tạo và được gán vào biến riêng để lưu giữ dòng văn bản.

1. Mở form frmLineTest.vb.
2. Nhắp đôi một trong hai nút lệnh GetWord.
3. Thay đổi mã giống với mã bên dưới.

```
Protected Sub btnGetWord_Click(ByVal sender As Object,
_
 ByVal e As System.EventArgs) Handles btnGetWord.Click
    Dim oLine As Line = New Line(txtLine.Text)
    txtWord.Text = _
     oLine.GetWord(txtWordNum.Text.ToInt32())
End Sub
```

4. Ấn **F5** để thử mẫu này.

Lưu ý, bây giờ khai báo của đối tượng oLine chuyển giá trị từ hộp văn bản txtLine. Điều này khác với Visual Basic 6.0, trong đó câu lệnh Dim không phải là mã khả thi. Bây giờ câu lệnh Dim là mã khả thi. Thực ra, bây giờ thậm chí bạn có thể đặt điểm ngắt trên câu lệnh Dim.

Tóm tắt

Các phương thức quá tải là một cách rất tốt để duy trì mục đích của một phương thức rõ ràng khi cho phép các tham số khác nhau ảnh hưởng đến giá trị bạn nhận lại từ phương thức. Nhớ là bạn phải thay đổi các tham số, chứ không phải chỉ kiểu trả lại để quá tải một phương thức. Khi quyết định các phương thức nào quá tải, mục đích của các phương thức vẫn phải nhất quán. Chính vì bạn có thể quá tải một phương thức không nhất thiết có nghĩa là bạn sẽ quá tải. Hãy luôn luôn quan tâm đến mục đích của

phương thức và các phương thức khác nhau bạn tạo thực chất có làm một thứ giống nhau hay không.

Câu hỏi ôn tập

1. Quá tải có ý nghĩa gì?

2. Bạn dùng từ khóa nào để nhận dạng một phương thức được quá tải?

3. Đúng hay sai: Kiểu trả lại tạo ra sự khác nhau trong signature của một phương thức?

4. Đúng hay sai: Bạn phải dùng từ khóa Overloads trên Sub New?

Trả lời câu hỏi ôn tập

1. Để sử dụng cùng tên phương thức chỉ với các tham số khác nhau cho một số thủ tục khác nhau trong lớp của bạn.

2. Overloads

3. Sai.

4. Sai.

Chương 13

Thừa kế

- Tìm hiểu cách thừa kế từ một lớp cơ sở.
- Tìm hiểu thừa kế Interface.
- Tìm hiểu thừa kế Implementation.

Thừa kế

Một trong các tính năng quan trọng của các ngôn ngữ OOP (Object Oriented Programming – Lập trình hướng đối tượng) là sự thừa kế (*Inheritance*). Sự thừa kế là khả năng truy tìm toàn bộ tính năng của một lớp sẵn có và mở rộng các khả năng này mà không cần phải viết lại lớp gốc. Trước khi có VB.NET, các lập trình viên Visual Basic chưa có được khả năng này. Trong VB.NET, bạn có thể thừa kế từ các lớp đi cùng với .NET Framework cũng như các lớp bạn tạo. Ở chương này, bạn sẽ tìm hiểu cách dùng khả năng thừa kế và sẽ thấy nó có thể giảm thời gian lập trình của bạn đáng kể như thế nào.

Một thí dụ đơn giản

Trong nhiều lớp bạn tạo, bạn thấy thường cần cùng các đặc tính và các phương thức từ một lớp khác bạn đã tạo trước đây. Thí dụ, nếu bạn có lớp cơ sở được gọi là lớp Person và nó chứa đặc tính LastName và FirstName, và phương thức Print, bạn thấy là đối với lớp Employee, bạn cần các đặc tính và các phương thức tương tự. Bạn cũng có thể cần thêm các đặc tính, chẳng hạn như EmployeeID và Salary. Khi bạn thừa kế từ lớp Person (lớp cơ sở), bạn có thể thêm các đặc tính này vào lớp Employee mới này và vẫn truy xuất được tất cả các đặc tính trong lớp Person. Sự thừa kế hầu như là khả năng để một lớp tự định nghĩa như tất cả các đặc tính và các phương thức của một lớp đặc biệt, sau đó mở rộng định nghĩa của lớp cơ sở bằng cách cộng thêm các đặc tính và các phương thức.

Thuật ngữ thừa kế

Trước khi chúng ta nghiên cứu sâu chủ đề này, chúng ta hãy định nghĩa một vài thuật ngữ. Một lớp mới được tạo bởi sự thừa kế đôi khi còn được gọi là *child class* (lớp con) hoặc *subclass*. Lớp bạn thừa kế đầu tiên được gọi là *base class* (lớp cơ sở), *parent class* (lớp cha) hoặc *superclass* (lớp trên cùng). Trong một số ngôn ngữ OOP, lớp cơ sở có thể thừa kế từ hơn một lớp cơ sở. Điều này có nghĩa là nếu bạn đã có một lớp Person và lớp Car, thì lớp Driver có thể thừa kế tất cả các đặc tính và các phương thức của hai lớp này. Trong thế giới .NET, chỉ cho phép thừa kế duy nhất một lớp, vì vậy mỗi lớp con sẽ chỉ có một lớp cơ sở. Bạn được phép sử dụng thừa kế tiếp theo, trong đó một lớp thừa kế từ lớp cơ sở đã thừa kế một lớp cơ sở khác.

Các lý do sử dụng thừa kế

Sự thừa kế được ưa dùng vì bạn không muốn phải viết lại mã giống nhau nhiều lần. Nếu bạn có hai lớp tách riêng và mỗi lớp phải bổ sung đặc tính FirstName và LastName, bạn sẽ sao chép mã. Nếu bạn muốn thay đổi sự bổ sung của một trong các đặc tính này, bạn phải tìm tất cả các lớp đã bổ sung các đặc tính này để thay đổi. Không những mất thời gian mà bạn còn dễ có thể tạo ra các lỗi trong các lớp khác nhau.

Một điều cần nhớ khi bạn xét việc sử dụng thừa kế là mối quan hệ giữa hai lớp sẽ là kiểu quan hệ "thuộc về". Thí dụ, Employee là một Person và Manager là một Person, như vậy hai lớp này có thể thừa kế lớp Person. Nhưng bạn không thể có lớp Leg thừa kế lớp Person, vì Leg không phải là người.

Ghi đè

Một khi bạn bắt đầu thừa kế tính năng của một lớp cơ sở, bạn có thể thấy là phương thức generic bạn đã viết trong lớp cơ sở chỉ thực hiện một phần những gì bạn cần ở lớp thừa kế. Thay vì tạo một phương thức hoàn toàn mới với một tên mới để thực hiện đầy đủ tính năng bạn muốn, bạn có thể ghi đè phương thức của lớp cơ sở vào lớp mới này.

Khi ghi đè, bạn có tùy chọn ghi đè toàn bộ phương thức của lớp cơ sở hoặc bạn có thể viết mã thực hiện điều gì đó trong lớp cơ sở và sau đó gọi phương thức của lớp cơ sở, rồi viết thêm mã sau khi phương thức của lớp cơ sở thi hành xong.

Tạo lớp thừa kế

Sự thừa kế cho bạn khả năng sử dụng tất cả các đặc tính và các phương thức từ một lớp bên trong một lớp khác. Thay vì phải chép và dán mã từ lớp này sang lớp kia, bạn dùng từ khóa *Inherits* để có tính năng từ lớp cơ sở.

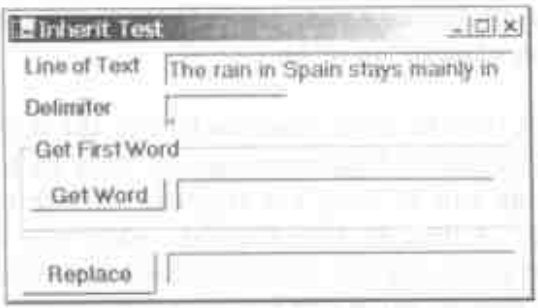
Có một số kiểu thừa kế khác nhau mà bạn có thể dùng trong Visual Basic.NET. Thừa kế Implementation và thừa kế Interface là hai kiểu thừa kế bạn sẽ dùng khi làm việc với các lớp. Một kiểu thừa kế khác là thừa kế Visual, nó cho phép bạn thừa kế hình thức và năng khiếu của một form trong một form khác. Kiểu thừa kế Visual sẽ được đề cập ở một tài liệu khác.

Thừa kế Implementation

Ở chương này, bạn sẽ tạo một lớp mới được gọi là LineDelim, sẽ thừa kế toàn bộ tính năng của lớp Line được tạo trong tài liệu "Tạo các lớp". Bạn sẽ mở rộng lớp Line bằng cách thêm hai đặc tính và một phương thức. Đặc tính đầu tiên sẽ cho phép bạn nhận và xác lập kí tự định biên vào lớp. Dấu định biên này sẽ được dùng để thay thế tất cả các kí tự trắng trong dòng ấy tới kí tự dấu định biên này. Đặc tính thứ hai là OriginalLine, sẽ được dùng để lưu giữ dòng văn bản gốc trước khi chèn dấu định biên mới vào dòng văn bản. Phương thức mới bạn sẽ tạo là ReplaceAll(), sẽ thay thế tất cả các kí tự trắng trong dòng có kí tự dấu định biên. Sau đó bạn sẽ tìm hiểu cách ghi đè phương thức GetWord sao cho nó sử dụng dấu định biên này thay vì các kí tự trắng để ngắt dòng văn bản và tìm từ đầu tiên.

Xây dựng form mẫu

Form mẫu được trình bày ở hình 1 sẽ được dùng để kiểm tra lớp thừa kế bạn sẽ tạo.



Hình 1: *Form mẫu để kiểm tra Inheritance.*

1. Tạo form được trình bày ở hình 1 bằng cách lựa Project ➤ Add Windows Form.

2. Đặt tên của form này là frmLineTest.vb và nhấp OK.

3. Kế tiếp, tạo các control thích hợp trên form và xác lập các đặc tính như được trình bày ở bảng 1.

Kiểu control	Đặc tính	Giá trị
Label	Name	Label1
	Text	Line of Text
TextBox	Name	txtLine
	Text	The rain in Spain stays mainly in the plain
TextBox	Name	txtDelim
	Text	,
GroupBox	Name	fraWord

	Text	Get First Word
CommandButton	Name	btnFirst
	Text	Get Word
TextBox	Name	txtFirstWord
	Text	
	ReadOnly	True
CommandButton	Name	btnReplace
	Text	Replace
TextBox	Name	txtReplace
	Text	
	ReadOnly	True

Bảng 1: *Form để kiểm tra sự thừa kế.*

Xác định lớp Line

Xây dựng lớp Line để thừa kế.

1. Lựa Project ➢ Add Class từ menu.

2. Gõ mã được trình bày bên dưới vào.

```
Public Class Line
    Private mstrLine As String

    Property Line() As String
        Get
            Return mstrLine
        End Get
        Set(ByVal Value As String)
            mstrLine = Value
        End Set
    End Property

    ReadOnly Property Length() As Integer
        Get
            Return mstrLine.Length
```

```
        End Get
    End Property

    Public Function GetWord() As String
        Dim astrWords() As String

        astrWords = mstrLine.Split(" ".ToCharArray())

        Return astrWords(0)
    End Function
End Class
```

Tạo Subclass

Bây giờ bạn có form và đã tạo lớp cơ sở, giờ là lúc bắt đầu tiến trình thừa kế.

1. Lựa Project ➤ Add Class và đặt tên lớp là LineDelim.vb. Nhắp OK.

2. Sửa đổi mã Visual Basic.NET tạo cho bạn khi bạn thêm lớp mới như mẫu được trình bày bên dưới.

```
Public Class LineDelim
    Inherits Line

End Class
```

Bạn đã thêm câu lệnh Inherits Line để có thể sử dụng tất cả các đặc tính và các phương thức của lớp Line bên trong lớp mới được tạo này.

1. Mở form frmLineTest.vb.

2. Nhắp đôi nút GetWord.

3. Thêm mã sau vào thủ tục sự kiện Click cho nút này:

```
Protected Sub btnFirst_Click(ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles btnFirst.Click
    Dim oLine As LineDelim = New LineDelim()

    oLine.Line = txtLine.Text
    txtFirstWord.Text = oLine.GetWord()
End Sub
```

4. Chạy project và nhấp nút GetWord của form. Bạn sẽ thấy từ "The" xuất hiện trong hộp văn bản chỉ đọc bên nút này.

Đây là một câu lệnh Inherits rất mạnh. Chỉ với một câu lệnh, bây giờ bạn có tất cả các đặc tính và các phương thức của lớp Line sẵn có trong lớp LineDelim. Mặc dù lớp mới này chưa làm gì mới mẻ, nhưng nó trình bày tất cả mã hoạt động từ lớp Line thừa kế.

Thêm tính năng

Bây giờ bạn có thể mở rộng lớp LineDelim với các đặc tính và các phương thức bổ sung. Thực hiện các bước sau để thêm hai đặc tính mới vào lớp LineDelim này.

Thêm hai biến Private được trình bày bên dưới sau câu lệnh Inherits bạn đã thêm vào ở đoạn trước.

```
Private mstrDelim As String = " "
Private mstrOriginal As String
```

Gõ mã bên dưới vào để thêm Property Statements thích hợp cho mỗi biến Private này. Bạn có thể đặt mã này ngay sau hai dòng bạn vừa nhập ở trên.

```
Public Property Delimiter() As String
    Get
        Return mstrDelim
    End Get
```

```
    Set(ByVal Value As String)
        mstrDelim = Value
    End Set
End Property

Public ReadOnly Property OriginalLine() As String
    Get
        Return mstrOriginal
    End Get
End Property
```

Bây giờ bạn có thể sử dụng đặc tính Delimiter để vừa xác lập và nhận giá trị từ biến Private, mstrDelim.

Đôi khi có thể bạn không muốn người khác thay đổi một trong các đặc tính của mình. Trong trường hợp này, bạn có thể tạo cho đặc tính ấy chỉ đọc. Để làm điều này, bỏ câu lệnh Set và thêm thuộc tính ReadOnly vào câu lệnh Property. Bạn có thể thấy một thí dụ về điều này ở khai báo của đặc tính OriginalLine được trình bày ở mã trên.

Kế tiếp, tạo phương thức được gọi là ReplaceAll, sẽ thay thế tất cả các kí tự trắng trong dòng văn bản có dấu định biên được chuyển vào đặc tính Delimiter.

```
Public Function ReplaceAll() As String
    mstrOriginal = MyBase.Line

    Return MyBase.Line.Replace(" ", mstrDelim)
End Function
```

Phương thức ReplaceAll truy tìm dòng văn bản gốc từ phương thức Line của lớp cơ sở. Bạn đã dùng cú pháp MyBase.Line để truy tìm đặc tính của lớp cơ sở. Hàm ReplaceAll đặt giá trị của đặc tính MyBase.Line vào biến Private, *mstrOriginal* mà bạn vừa tạo cho lớp này. Phương thức Replace của kiểu dữ liệu String thay thế tất cả các thể nghiệm của một kí tự chuỗi có kí tự dấu định biên mới, *mstrDelim* mà bạn đã xác lập trong đặc tính Delimiter.

Từ khóa MyBase

Từ khóa MyBase có thể được dùng từ một lớp con bất kỳ để đưa ra một lời gọi một đặc tính hoặc phương thức bất kỳ trong lớp cơ sở. Thậm chí bạn có thể dùng nó để gọi phương thức của lớp cơ sở bạn đã ghi đè trong lớp subclass (lớp con). Thí dụ, nếu bạn dùng một phương thức trong lớp cơ sở gọi ReplaceAll và bạn đã ghi đè phương thức đó vào lớp con, thì từ bên trong phương thức ReplaceAll trong lớp con, bạn có thể gọi phương thức ReplaceAll trong lớp cơ sở.

1. Mở form frmLineTest.vb.

2. Nhấp đôi nút Replace để đưa ra thủ tục sự kiện Click.

3. Viết mã bên dưới vào sự kiện Click cho nút btnReplace.

```
Protected Sub btnReplace_Click( _
 ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles btnReplace.Click
    Dim oLine As LineDelim = New LineDelim()

    oLine.Delimiter = txtDelim.Text
    oLine.Line = txtLine.Text
    txtReplace.Text = oLine.ReplaceAll()
End Sub
```

Mã này sẽ xác lập đặc tính Delimiter là giá trị được nhập vào hộp văn bản txtDelimiter trên form mẫu. Sau đó bạn gọi ra phương thức ReplaceAll để đổi tất cả các kí tự trắng trong dòng văn bản thành kí tự dấu định biên mới.

4. Ấn **F5** để chạy project.

5. Nhấp nút Replace và bạn sẽ thấy một dấu phẩy ở giữa mỗi từ của câu trong hộp văn bản được định vị bên nút Replace.

Ghi đè một phương thức

Bây giờ bạn đã thêm đặc tính Delimiter, bạn nên thay đổi phương thức GetWord trong lớp LineDelim để sử dụng dấu định biên đó thay vì một kí tự trắng mà lớp Line sử dụng. Vì bạn không cần thay đổi lớp cơ sở, bạn sẽ phải ghi đè các khả năng của phương thức GetWord trong lớp LineDelim. Trước khi bạn có thể tạo một phương thức GetWord mới trong lớp LineDelim, bạn cần thêm một từ khóa vào khai báo của phương thức GetWord trong lớp Line.

1. Mở cửa sổ mã cho lớp Line.vb từ cửa sổ Solution Explorer.

2. Định vị khai báo cho phương thức GetWord (khai báo không có các tham số) như được trình bày bên dưới:

```
Public Overloads Function GetWord() As String
```

3. Thêm từ khóa **Overridable** vào khai báo hàm, như được trình bày ở listing bên dưới (không có từ khóa này, bạn sẽ không thể ghi đè phương thức này).

```
Public Overridable Overloads Function GetWord() As
String
```

4. Mở lớp LineDelim.vb và thêm phương thức GetWord mới bằng cách dùng mã được trình bày bên dưới.

```
Public Overloads Overrides Function GetWord() As String
    Dim astrWords() As String

    astrWords                                          =
MyBase.Line.Split(mstrDelim.ToCharArray())
```

```
    Return astrWords(0)
End Function
```

Từ khóa Overrides trong khai báo hàm cần thiết khi bạn muốn thay đổi tính năng của một phương thức của lớp cơ sở. Bây giờ phương thức GetWord trong lớp LineDelim sẽ sử dụng giá trị của đặc tính Delimiter để tách các từ trong một câu.

Một hiệu ứng phụ của việc ghi đè là chỉ một trong các phương thức GetWord mà mã của bạn có thể nhận biết nó là phiên bản duy nhất của phương thức. Mã của bạn không thể gọi các phiên bản khác của phương thức GetWord. Để làm cho thấy được tất cả chúng, bạn phải ghi đè mỗi phương thức, giống như bạn đã thực hiện lớp LineDelim.

1. Chạy project bằng cách ấn **F5**.

2. Đặt dấu phẩy vào giữa mỗi từ trong câu và điền Delimiter vào hộp văn bản với một dấu phẩy.

3. Nhắp nút Get Word.

4. Từ đầu tiên trong câu bây giờ xuất hiện trong hộp văn bản bên nút Get Word.

Các lớp trừu tượng

Một lớp trừu tượng (abstract) trong một lớp hoặc một giao diện không có đầy đủ tính năng được định nghĩa. Kiểu lớp này mong đợi các lớp khác thừa kế nó và tạo phần tính năng còn lại để được hữu dụng. Lớp Person là một điển hình, nó chỉ có một số đặc tính và khái niệm cơ bản về các kiểu phương thức sẽ được tạo, chứ không phải là cách tạo các phương thức.

Lớp Person được định nghĩa là một lớp trừu tượng chỉ có thể được thừa kế, chứ không phải là đối tượng thực sự được tạo vào thời gian chạy. Để thực hiện điều này, bạn sẽ dùng từ khóa **MustInherit** khi tạo một lớp trừu tượng. Mỗi lớp thừa kế từ lớp này, như lớp Employee, sẽ tạo tất cả các đặc tính và các phương thức thích hợp với tính năng riêng biệt cần được điền vào.

Thí dụ, lớp Employee sẽ tạo phương thức Print thực, trong khi lớp Person chỉ xác định phương thức Print phải tồn tại; không có mã kết hợp với phương thức Print trong lớp Person. Các lớp trừu tượng rất tốt cho việc bảo đảm các trình thiết kế các lớp con bổ sung tất cả các giao diện thường được một ứng dụng cần.

Bên dưới là một thí dụ về lớp Person phải được thừa kế. Nó chứa các giao diện cho hai đặc tính và hai phương thức phải được ghi đè.

```
Public MustInherit Class Person
    MustOverride Property FirstName() As String
    MustOverride Property LastName() As String

    MustOverride Sub Print()
    MustOverride Function Talk() As String
End Class
```

Dù đây là điển hình, nhưng rất có thể bạn sẽ bổ sung các đặc tính, tuy nhiên, hãy để nguyên các phương thức là **MustOverride**. Bên dưới là phiên bản thứ hai của lớp Person này để bổ sung các đặc tính vẫn còn để nguyên các phương thức được bổ sung.

```
Public MustInherit Class Person
    Private mstrFirstName As String
    Private mstrLastName As String

    Property FirstName() As String
        Get
```

```
            Return mstrFirstName
        End Get
        Set(ByVal Value As String)
            mstrFirstName = Value
        End Set
    End Property

    Property LastName() As String
        Get
            Return mstrLastName
        End Get
        Set(ByVal Value As String)
            mstrLastName = Value
        End Set
    End Property

    Public MustOverride Sub Print()
    Public MustOverride Function Talk() As String

    Public Function Test() As String
        Return "Test"
    End Function
End Class
```

Khi bạn đã khai báo lớp Person, bây giờ nó có thể được thừa kế. Ở mã bên dưới, một lớp được gọi là Employee sẽ thừa kế lớp Person. Sau đó nó truy xuất đặc tính FirstName và LastName, phải bổ sung mã cho các phương thức.

```
Public Class Employee
    Inherits Person

    Public Overrides Sub Print()

    End Sub

    Public Overrides Function Talk() As String

    End Function
End Class
```

Sau đó bạn có thể dùng lớp Person lần nữa, một thừa kế của nó trong lớp Manager.

```
Public Class Manager
    Inherits Person

    Public Overrides Sub Print()

    End Sub

    Public Overrides Function Talk() As String

    End Function
End Class
```

Thừa kế Interface

Khi bạn muốn tạo một lớp trừu tượng thuần túy, bạn sử dụng từ khóa **Interface** thay vì **Class**. Bạn cho giao diện một tên và định nghĩa mỗi đặc tính và phương thức bạn mong đợi lớp con bổ sung. Lý do bạn làm điều này là không có gì trong lớp cơ sở sẽ tạo ra ý nghĩa bổ sung — nó chỉ chứa dữ liệu generic mà không có các phương thức. Bạn sẽ tạo một contract (hợp đồng) cho biết bất kỳ lớp con nào sử dụng giao diện phải theo đúng các nguyên tắc nào đó.

1. Chúng ta hãy thêm một lớp mới vào project bạn đã tạo.

2. Lựa Project ➢ Add Class từ menu Visual Studio.

3. Thêm mã sau vào lớp này:

```
Interface IPerson
    Property FirstName() As String
    Property LastName() As String

    Sub Print()
    Sub Talk()
End Interface
```

Bạn có thể thấy là bạn sẽ định nghĩa các đặc tính và các thủ tục con giống như bạn định nghĩa chúng bình thường. Sự khác biệt duy nhất là bạn không điền bất kỳ mã nào cho chúng. Bây giờ, chúng ta hãy xét cách sử dụng Interface này trong một định nghĩa lớp.

Để có cùng file lớp bạn đã tạo ở bước trước đây, thêm mã bên dưới.

```
Public Class Employee
    Implements Iperson

    Private mstrFirstName As String
    Private mstrLastName As String

    Property FirstName() As String _
     Implements Person.FirstName
        Get
            Return mstrFirstName
        End Get
        Set
            mstrFirstName = Value
        End Set
    End Property

    Property LastName() As String _
     Implements Person.LastName
        Get
            Return mstrLastName
        End Get
        Set
            mstrLastName = Value
        End Set
    End Property

    Sub Print() Implements Person.Print
        ' Some code goes here
    End Sub

    Sub Talk() Implements Person.Talk
        ' Some code goes here
    End Sub
```

```
End Class
```

Dòng đầu tiên sau định nghĩa của lớp Employee là Implements Person. Từ khóa này cho biết bạn theo đúng contract được xác lập trước trong Person Interface. Bây giờ bạn có thể định nghĩa từng đặc tính và phương thức trong contract đó. Sau mỗi câu lệnh Property, bạn phải có từ khóa Implements và chỉ định tên của Interface, một dấu chấm (.) và tên của phương thức/đặc tính bạn sẽ bổ sung. Visual Basic.NET sẽ truy tìm mỗi giao diện này và cho tới khi từng giao diện được tạo, nó sẽ cho bạn biên dịch ứng dụng.

Nếu bạn muốn chạy mã, bạn cần tạo các thủ tục con thích hợp, vì chúng vẫn còn trống ở mẫu trên. Sau khi bạn tạo mỗi thủ tục này, bạn sẽ có thể khai báo đối tượng Employee mới và sử dụng đối tượng Employee giống như bất kỳ đối tượng nào khác bạn thường tạo và sử dụng.

Chọn kiểu thừa kế để sử dụng

Việc quyết định sử dụng thừa kế Implementation hoặc Interface không phải lúc nào cũng đơn giản. Trong nhiều trường hợp, bạn có thể phải thực hiện một phần nhỏ của cả hai thừa kế. Thí dụ, bạn có thể thêm định nghĩa phương thức phải được ghi đè bởi một lớp con vào lớp Line. Bạn bổ sung nó bằng cách dùng từ khóa MustOverride trong định nghĩa thủ tục.

```
Public MustOverride Sub Init()
```

Khi bạn thêm định nghĩa này vào Class, nó hành động giống một Interface. Trong lớp con, phương thức Init phải được định nghĩa và nó phải dùng từ khóa Overrides. Sau là một thí dụ về cách bạn có thể định nghĩa phương thức Init này:

```
Public Overrides Sub Init()
    strDelim = " "
    mstrLine = "Test Line"
End Sub
```

Một lần nữa, lưu ý việc sử dụng từ khóa Overrides để cho trình biên dịch biết phương thức này là phương thức ghi đè phương thức Init trong lớp cha.

Ngưng thừa kế

Trong một số thể nghiệm, có thể bạn không muốn các lớp khác được phát sinh từ lớp của mình. Nếu ở vào trường hợp này, bạn có thể dừng thừa kế trên một lớp bằng cách dùng từ khóa NotInheritable.

```
Public Class NotInheritable Employee
    ' Class definition
End Class
```

Thừa kế Visual

Ngoài thừa kế Implementation, bạn còn có thể tạo một form với giao diện chuẩn và sau đó thừa kế giao diện đó tới một form khác. Để thấy rõ điều này hoạt động như thế nào, thực hiện các bước sau.

1. Tạo một project mới trong Visual Studio và xác lập Name là **VisualInheritance.sln**.

2. Đặt lại tên Form1.vb là **frmDataEntry.vb**.

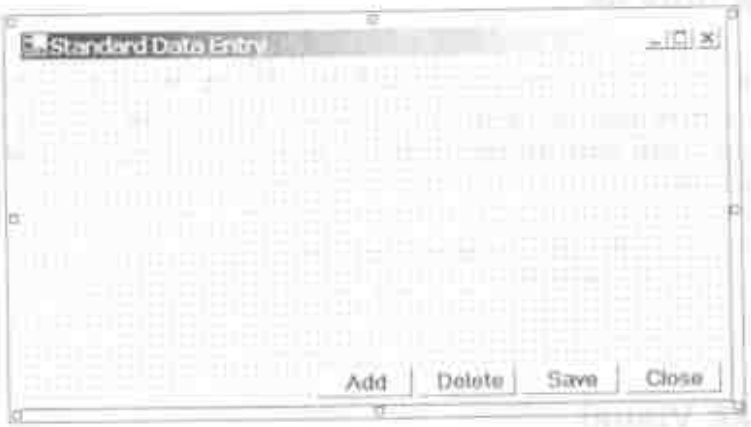
3. Xác lập đặc tính Name của form này là **frmDataEntry**.

4. Thêm bốn control nút vào góc phải dưới của form này như được trình bày ở hình 2.

5. Xác lập các tên là btnAdd, btnDelete, btnSave và btnClose tương ứng từ trái sang phải.

6. Tô sáng tất cả bốn nút bằng chuột.

7. Xác lập đặc tính Anchor là **Bottom, Right**.



Hình 2: *Các form Standard Data Entry là các ứng viên tốt cho các form cơ sở.*

8. Nhấp đôi nút Close và viết mã sau.

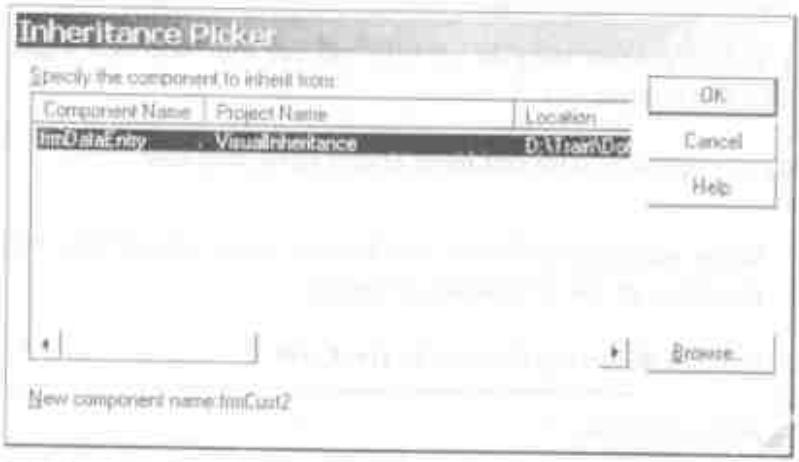
```
Private Sub btnClose_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles BtnClose.Click
    Me.Close()
End Sub
```

Tạo form thừa kế

Bây giờ bạn sẵn sàng thừa kế từ form mục nhập dữ liệu chuẩn này. Trước khi bạn có thể thừa kế từ form này, bạn phải xây dựng form. Thực ra, sau khi có bất kỳ thay đổi nào đối với form cơ sở, bạn sẽ phải xây dựng lại project ấy.

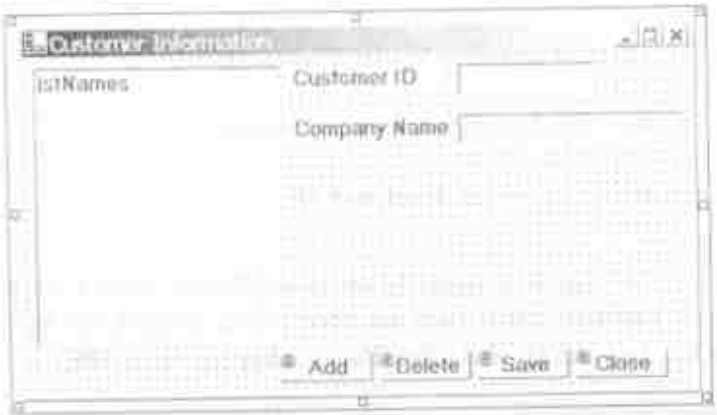
1. Lựa Build | Build từ menu Visual Studio.NET.
2. Lựa Project | Add Inherited Form từ menu.
3. Đặt tên của form mới là **frmCust.vb**.

Bây giờ bạn sẽ thấy hộp thoại Inheritance Picker, như ở hình 3, nó sẽ trình bày một danh sách các form trong project mà bạn có thể thừa kế. Lựa thành phần **frmDataEntry** và nhấp **OK**.



***Hình 3:** Hộp thoại Inheritance Picker cho bạn lựa một form khác trong project để thừa kế.*

Sau khi nhấp OK trên hộp thoại Inheritance Picker, bạn sẽ có một form mới trong project của bạn có bốn control nút trên nó, nhưng chúng sẽ có một mũi tên nhỏ ở góc trái trên của mỗi nút. Điều này cho biết các control này được thừa kế từ form cơ sở. Bây giờ bạn có thể tạo giao diện riêng cho form với bốn nút này. Tạo một form giống như hình 4.



Hình 4: *Các control thừa kế sẽ hiển thị với một mũi tên ở góc trái trên của control.*

Bây giờ bạn sẵn sàng thử form khách hàng mới này.

1. Nhấp nút chuột phải vào tên Project trong cửa sổ Solution Explorer và lựa Properties từ menu.
2. Xác lập đối tượng Startup là **frmCust**.
3. Nhấp nút OK.
4. Ấn **F5** để chạy ứng dụng này.

Nếu bạn thực hiện mọi thứ chính xác, bạn có thể thay đổi kích thước form này và các nút sẽ di chuyển cùng với form. Bạn

Tóm tắt

Ở chương này, bạn đã tìm hiểu cách thừa kế từ một lớp cơ sở. Bạn đã bổ sung các đặc tính vào lớp cơ sở và sử dụng từ khóa Overrides để thay thế tính năng được định nghĩa trong lớp cơ sở. Bạn cũng đã tìm hiểu cách dùng từ khóa MyBase để gọi các phương thức trong lớp cơ sở, từ đó mở rộng những gì lớp cơ sở có thể thực hiện. Sự thừa kế không thích hợp cho tất cả các ứng dụng của bạn, nhưng bạn sẽ thấy là sử dụng chính xác nó có thể là một công cụ rất mạnh.

Câu hỏi ôn tập

1. Định nghĩa thừa kế.
2. Bạn dùng từ khóa nào để thừa kế từ một lớp cơ sở?
3. Bạn phải dùng từ khóa này ở đâu?
4. Sự khác biệt giữa thừa kế Implementation và Interface như thế nào?
5. Bạn dùng từ khóa nào để ngưng thừa kế trên một lớp?

Bài tập

- Tạo lớp LineDelim như được trình bày ở chương này.

Trả lời câu hỏi ôn tập

1. Khả năng truy hồi toàn bộ tính năng của một lớp sẵn có và mở rộng hoặc thay thế tính năng đó.

2. Inherits.

3. Trên dòng đầu tiên sau khai báo Public Class.

4. Implementation cho bạn tính năng, Interface chỉ cho bạn định nghĩa.

5. NotInheritable.

Chương 14

Tạo thư viện lớp

- Tìm hiểu kiến trúc thành phần của .NET.
- Tìm hiểu cách xây dựng các thành phần và các đối tượng .NET.
- Tìm hiểu cách các đối tượng .NET trình bày một giao diện.

Thư viện lớp (Class Library)

Vì kiến trúc .NET dựa vào đối tượng, nên việc tạo các lớp và các thành phần được xây dựng từ các lớp đó được xem là rất quan trọng đối với các lập trình viên .NET. Một kiến thức toàn diện về kiến trúc thành phần của .NET là yêu cầu thiết yếu để lập trình .NET thành công.

Tất cả các đối tượng .NET đều biểu lộ các thuộc tính quan trọng (các đặc tính, các phương thức và các sự kiện) hỗ trợ các

nguyên tắc về sự trừu tượng hóa hướng đối tượng quan trọng (sự thể hiện các dịch vụ được đối tượng cung cấp) và sự gói riêng (sự tách rời giao diện lập trình với công việc được đối tượng thực hiện).

Vì kiến trúc của các đối tượng VB.NET bạn chịu trách nhiệm bổ sung giao diện cần thiết cho các lập trình viên khác sử dụng các dịch vụ ứng dụng của bạn. Phần lớn thời gian phát triển của bạn là thiết kế các đối tượng và viết mã định nghĩa các đối tượng và các thành phần được các ứng dụng của bạn sử dụng và trình bày.

Tìm hiểu các thư viện lớp .NET

Hầu hết các lập trình viên có sự hiểu biết tự nhiên về các đối tượng. Xét cho cùng, bạn bị vây quanh bởi các đối tượng của tất cả các loại và tương tác với các đối tượng đó trong đời sống hàng ngày. Bạn cũng thường xử lý với các đối tượng VB. Nhưng bạn đã từng thực sự nghĩ về nơi tất cả các đối tượng phát xuất chưa? Hầu hết chúng chẳng có gì khác hơn là một thành phần (hoặc thư viện lớp) trong form của một OCX hoặc DLL. Class Library trong .NET tiêu biểu là .DLL, chứa một hoặc nhiều lớp mà bạn có thể dùng từ một ứng dụng khác.

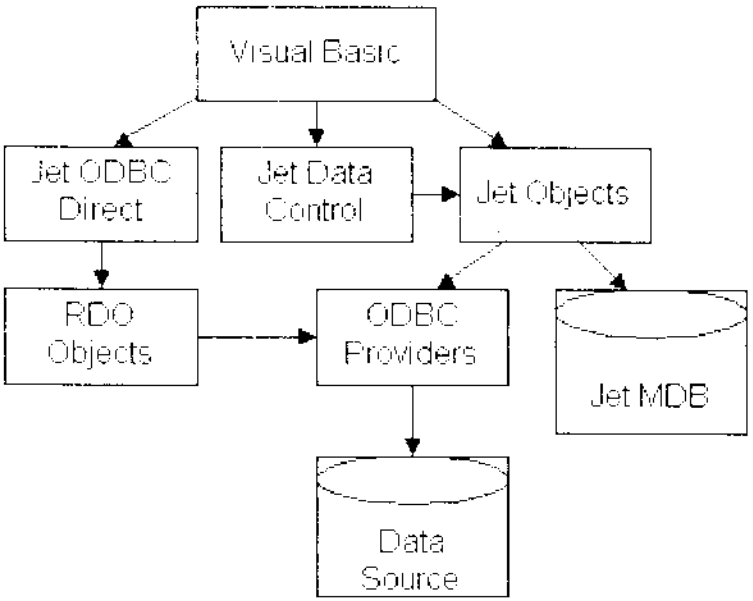
Việc lập trình hướng đối tượng .NET đơn giản không có gì nhiều hơn việc tạo lớp, thêm các đặc tính, các phương thức và các sự kiện được lớp đòi hỏi, và bao gồm lớp ấy trong các ứng dụng khác. Tuy nhiên, .NET Class Library là một lớp biên dịch sẵn hoặc một tập các lớp được thể hiện bằng .DLL. Vào thời gian chạy, một .NET DLL được gọi ra và được nạp vào bộ nhớ để được một ứng dụng người dùng nào đó sử dụng. Các thành phần .NET này rất thường được xây dựng và kiểm tra như các project .NET độc lập. Sau khi biên dịch thành một .NET DLL, các thành phần này có thể được thêm vào nhiều ứng dụng .NET như các bộ cung

cấp dịch vụ plug-in. Điều này tốt hơn chỉ sử dụng các lớp tách riêng như các file .VB và gộp chúng vào mỗi project.

Mỗi .NET DLL có thể chứa nhiều lớp. Điều này có nghĩa là mỗi DLL có thể trình bày một số lớp khác nhau hoặc có thể chỉ có một lớp. Trong trường hợp này, bạn có thể xây dựng một .NET DLL hỗ trợ truy xuất dữ liệu từ xa. DLL này có thể trình bày các lớp tách riêng cho CSDL và một số các DataSet khác nhau để biểu thị mỗi bảng trong CSDL. Hoặc trong một trường hợp khác, bạn có thể dùng lớp Line và LineDelim để cư trú một DLL và được một ứng dụng sử dụng.

Hình 1 được trình bày ở dưới cho bạn một thí dụ về một DLL đơn giản trình bày ba tập lớp khác nhau. Cùng DLL này có thể được nhiều ứng dụng khác sử dụng trong form của các file EXE. Đương nhiên, một DLL khác cũng có thể sử dụng DLL này.

Trong Visual Studio.NET, các .NET DLL chạy bên trong không gian xử lý của ứng dụng chủ và dùng chung bộ nhớ và thời gian của bộ xử lý với các ứng dụng chủ của chúng. Vào thời gian chạy, thành phần (là một phần hợp ngữ của ứng dụng chủ và được tham chiếu bởi manifest (bản kê) của nó) được nạp từ đĩa và được thêm vào không gian xử lý của ứng dụng chủ. Vì không có các lời gọi thủ tục từ xa được phát sinh để sắp xếp truyền thông giữa thành phần ấy và ứng dụng chủ của nó, việc xác lập và đọc các giá trị đặc tính, gọi ra các phương thức và đáp ứng các sự kiện phát sinh bởi thành phần ấy xảy ra rất nhanh.



Hình 1: *Các DLL chứa nhiều lớp và có thể được dùng từ nhiều EXE hoặc các DLL khác.*

Xây dựng thành phần

Ở chương này bạn sẽ xây dựng một lớp .NET đơn giản và gói nó vào một DLL. Một khi DLL này được xây dựng, bạn sẽ tham chiếu thành phần này từ .NET EXE. Mặc dù thí dụ này không phức tạp lắm, nhưng nó đủ minh họa hầu hết các nguyên tắc về phát triển thành phần trong .NET.

Sự đòi hỏi tiêu biểu của các hệ thống CSDL phân phối là đồng bộ hóa và điều hòa các cập nhật được thực hiện ở các vị trí từ xa. Trong một ứng dụng CSDL .NET chủ, cùng bản ghi có thể được cập nhật ở nhiều site và được gửi tới server để lưu trữ. Hoặc các bản ghi tương tự có thể được tạo ở nhiều site và được gửi tới server để xử lý. Trong các trường hợp như thế, quan trọng là

phải biết chính xác khi nào mỗi bản ghi đã được cập nhật hoặc được tạo sao cho các bản ghi mới có thể được xử lý khác hơn các bản ghi cũ.

Trong hầu hết các trường hợp, ứng dụng gán tem thời gian (timestamp) vào các bản ghi CSDL khi dữ liệu được sửa đổi hoặc được bổ sung. Server có thể so sánh tem thời gian trên một bản ghi với tem thời gian trên một bản ghi khác và xử lý các bản ghi phù hợp. Tuy nhiên, khi dữ liệu đã được cập nhật trong các timezone (múi giờ), cần phải sử dụng cơ sở thời gian thích hợp cho các tem thời gian. Điều này thường được thực hiện bằng cách dùng giờ của máy tính server làm tem thời gian thay vì giờ địa phương ở mỗi vị trí.

Đây là một việc sử dụng lý tưởng của thành phần .NET. .NET server có thể biểu thị thời gian của nó như một dịch vụ web có thể được một ứng dụng người dùng bất kỳ đọc và sử dụng. Việc cung cấp "server time" (thời gian máy chủ) như một dịch vụ web giúp điều hòa các thay đổi dữ liệu xảy ra ở các vị trí ở xa về địa lý.

Tạo thư viện lớp

Thực hiện theo các bước sau để xây dựng ứng dụng Class Library. Thư viện lớp này sẽ chứa một lớp để cho trở lại giờ và ngày tháng của máy chủ:

1. Khởi động Visual Studio.NET.
2. Lựa Class Library Template.
3. Xác lập Name của thư viện mới này là **DotNetLib**.

4. Nhấp OK.

5. Đổi tên của Public Class đã phát sinh từ "Class1" thành **ServerTime**.

6. Nhập mã sau vào Class này để định nghĩa lớp ServerTime.

```
Public Class ServerTime
  Private mdtTime As DateTime

  ReadOnly Property TimeStamp() As String
    Get
      mdtTime = Now()

      Return CStr(mdtTime)
    End Get
  End Property
End Class
```

Biến Private có tên *mdtTime* hoàn toàn không cần thiết trong lớp này. Chúng ta có thể cho trở lại giờ và ngày tháng hiện hành mà không phải gán nó vào biến *mdtTime* trước. Nhưng vì các project VB thường gia tăng nhanh vượt đặc tả ban đầu của chúng, trong tương lai có thể phải cần đến biến mức module được nhiều đặc tính và/hoặc các phương thức bên trong lớp ServerTime dùng chung.

Ở điểm này bạn có một DLL hoạt động sẵn sàng được một ứng dụng khác sử dụng.

Tạo ứng dụng người dùng

Bây giờ bạn tạo một Windows Application sẽ tham chiếu thư viện lớp DotNetLib này và sử dụng lớp ServerTime.

1. Lựa File > Add Project từ menu Visual Studio.NET.
2. Lựa template Windows Application.
3. Xác lập Name là **ServerConsumer**.
4. Nhấp OK.
5. Nhấp ứng dụng ServerConsumer trong cửa sổ Solution Explorer.
6. Nhấp nút chuột phải và lựa Set as Startup Project từ menu.
7. Nhấp folder (thư liệu) References dưới project mới này.
8. Nhấp nút chuột phải và lựa Add Reference...
9. Bạn sẽ thấy một hộp thoại như ở hình 2.
10. Lựa tab Projects và nhấp nút Select.
11. Nhấp OK để thêm tham chiếu project Class Library vào project Windows Application này.

Hình 2: *Sử dụng hộp thoại Add Reference để tham chiếu project Class Library.*

Bây giờ bạn sẵn sàng sử dụng lớp ServerTime trong project khác từ project ứng dụng Windows mới này.

12. Thêm TextBox vào WinForm đã phát sinh và đặt tên nó là **txtServerTime**.

13. Xóa chuỗi trong đặc tính Text để nó trống.

14. Thêm một nút vào WinForm và đặt tên nó là **btnGetServerTime**.

15. Xác lập đặc tính Text của nút này là **Get Server Time**.

16. Bây giờ form của bạn giống như hình 3.

Hình 3: *Form Server Time sẽ nhận thời gian máy chủ và đặt nó vào hộp văn bản.*

17. Nhấp đôi nút trên để thêm mã bên dưới vào đằng sau WinForm (các kí tự nối tiếp (_) được thêm vào khai báo của thủ tục sự kiện để dễ phân biệt):

```
Private Sub btnGetServerTime_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles btnGetServerTime.Click

    Dim st As DotNetLib.ServerTime

    st = New DotNetLib.ServerTime()

    txtServerTime.Text = st.TimeStamp

End Sub
```

Vì lớp ServerTime được gộp trong một project khác, nên bạn phải có tiền tố cho lớp ServerTime với tên của Namespace trong project kia. Theo mặc định, Namespace giống tên solution.

1. Kiểm tra ứng dụng này để xem bạn đã gõ mọi thứ vào chính xác hay không.

2. Ấn F5 để chạy ứng dụng này.

3. Nhấp nút Get Server Time và bạn sẽ thấy Date and Time xuất hiện trong Text Box.

Tạo NameSpace

Một tính năng mới và hữu ích khác trong các ứng dụng .NET là khả năng chỉ định NameSpaces bên trong project thành phần. Bạn nhớ là một DLL đơn giản có thể trình bày nhiều lớp. Cần biết Visual Studio.NET cung cấp một cách thuận tiện cho bạn tạo nhóm các lớp này như NameSpaces bên trong DLL. Điều này có thể đơn giản hóa rất nhiều tác vụ tham chiếu các lớp khác nhau bên trong DLL.

Bạn chỉ định một NameSpace bằng cách đánh dấu vùng mã với câu lệnh NameSpace và End NameSpace. Thí dụ, mã của lớp ServerTime và các đặc tính của nó có thể được gói bên trong một NameSpace được gọi là SystemInfo:

```
Namespace ServerInfo
    Public Class ServerTime
        Private mdtTime As DateTime

        ReadOnly Property TimeStamp() As String
            Get
                mdtTime = Now()

                Return CStr(mdtTime)
            End Get
        End Property
    End Class
End Namespace
```

Bây giờ một câu lệnh khai báo đối tượng ServerTime trở thành:

```
Dim st As DotNetLib.ServerInfo.ServerTime
```

ServerInfo NameSpace có thể không có hoặc có thêm các lớp, chẳng hạn như ServerDiskNames, ServerDatabases, vân vân.

Câu lệnh khai báo NameSpace (NameSpace ServerInfo) tương tự có thể xuất hiện trong nhiều module lớp bên trong một project. Visual Studio.NET sắp xếp các module lớp này và liên kết chúng lại như một DLL vào thời gian biên dịch.

Các Namespace cung cấp trình phát triển VB.NET với tính mềm dẻo lạ thường khi thiết kế các ứng dụng. Như trong trường hợp bạn có thể có lớp trùng tên bên trong các khoảng trống tên khác nhau. Bạn có thể có một lớp dọn sạch trong mỗi một khoảng trống tên và mỗi lớp dọn sạch xử lý các tác vụ dọn sạch (đóng các file CSDL, giải phóng các biến đối tượng, giải phóng các khóa bản ghi, vân vân) cho khoảng trống tên đó. Thậm chí bạn có thể có cùng các đặc tính cho mỗi lớp dọn sạch, dù mỗi lớp có các thủ tục dọn sạch khác nhau.

Sử dụng Imports

Thay vì phải gõ toàn bộ Namespace cùng với tên Class, bạn có thể dùng câu lệnh Imports. Bên dưới là một mẫu:

```
Imports DotNetLib.SystemInfo
```

Bây giờ khi bạn khai báo lớp ServerTime, bạn chỉ cần dùng câu lệnh sau:

```
Dim st As ServerTime
```

Thêm các lớp khác

Bạn sẽ dùng lớp Line và LineDelim bạn đã tạo trước đây trong tài liệu này và thêm chúng vào DotNetLib DLL này. Bằng cách này, tất cả các lớp của bạn có thể cư trú bên trong một project và một DLL. Điều này thuận tiện vì bây giờ bạn chỉ cần gộp một tham chiếu project vào tất cả các ứng dụng và truy xuất một số lớp dùng lại được. Để thêm các lớp này, thực hiện các bước sau:

1. Nhắp project DotNetLib.
2. Nhắp nút chuột phải và chọn Add > Add Existing Item
3. Tìm lớp clsLine.vb của bạn chứa cả lớp Line lẫn lớp LineDelim.
4. Nhắp file này và nhắp OK để thêm nó vào project của bạn.

Khi bạn thêm các file bằng cách này, file được chép từ vị trí gốc của nó vào folder của project này. Vì vậy bản copy gốc vẫn còn ở vị trí cũ và bây giờ project này có riêng bản copy cục bộ.

Giờ bạn có thể dùng lớp ServerTime, Line hoặc LineDelim bằng cách tham chiếu DotNetLib.ServerTime, DotNetLib.Line hoặc DotNetLib.LineDelim trong một project bất kỳ có solution DotNetLib này như một tham chiếu.

Cập nhật thông tin hợp ngữ

Những thông tin nào đó được trình biên dịch đưa vào thành phần ở bước build, có thể rất hữu dụng vào thời gian chạy. Thí dụ, có thể có nhiều phiên bản khác nhau của thành phần ServerTime được cài đặt trên một máy tính cá biệt. Một ứng dụng của người dùng có thể cần biết chính xác phiên bản nào được nạp trước khi nó cố sử dụng các phương thức và các đặc tính của thành phần ấy.

Mỗi hợp ngữ của project .NET được kèm theo một module AssemblyInfo. Module này bao gồm một số khai báo có thể được thay đổi vào thời gian thiết kế và được liên kết với thành phần ở bước biên dịch. Hình 4 trình bày module AssemblyInfo cho thành phần ServerTime với một số thuộc tính hợp ngữ (AssemblyTitle, AssemblyDescription và AssemblyCompany) được điền vào.

Bạn mở cửa sổ mã hợp ngữ (assembly code) từ Solution Explorer. Nhắp đôi module có tên AssemblyInfo.vb trong Solution Explorer và nhập các giá trị bạn cần vào mỗi thuộc tính của hợp ngữ.

Lưu ý, thuộc tính AssemblyVersion ở gần cuối module AssemblyInfo.vb. Theo mặc định, giá trị này được xác lập là "1.0.*". Dấu sao cho biết bạn muốn VB.NET tự động tăng số hiệu phiên bản lên một số (thí dụ, tăng "1.0.7" thành "1.0.8") mỗi lần DLL được biên dịch. Bạn có thể lựa ghi mã cứng (hard-code) một số phiên bản hoàn tất bằng cách nhập giá trị của nó vào thuộc tính này.

Hình 4: Chỉ định các thuộc tính hợp ngữ.

Sau khi DLL được biên dịch từ mã nguồn của thành phần, thông tin này có thể được đọc vào Windows Explorer bằng cách nhấp nút chuột phải vào DLL và lựa tab Version để xem các thuộc tính hợp ngữ (xem hình 5).

Hình 5: *Xem các thuộc tính hợp ngữ bằng Windows Explorer.*

Dưới Hood

Đôi khi hiểu được một file .SLN và .VBPROJ rất hữu dụng. Các file này chỉ là các file văn bản chứa thông tin mà môi trường Visual Studio sử dụng mỗi lần bạn nạp một trong các file này. Dù hai file này có định dạng khác nhau, nhưng bạn vẫn có thể đọc thông tin và hiểu rõ mỗi đoạn dữ liệu sẽ làm gì.

File Solution

Chúng ta hãy mở file DotNetLib.sln để khảo sát. Các vùng quan trọng nhất của .SLN này ở ngay trên đầu, nơi bạn có thể thấy listings Project. Nó trình bày tên của mỗi project được file solution này tham chiếu cũng như đường dẫn tới nơi chúng được định vị.

```
Microsoft Visual Studio Solution File, Format Version
7.00
Project("{F184B08F-C81C-45F6-A57F-5ABD9991F28F}")      =
"DotNetLib", "DotNetLib.vbproj", "{7B365305-6DF4-4F4D-
AA6B-0FE65B604563}"
EndProject
Project("{F184B08F-C81C-45F6-A57F-5ABD9991F28F}")      =
"ServerConsumer",
"..\ServerConsumer\ServerConsumer.vbproj",   "{2533BEE1-
4B39-4A31-A7CC-2C10FAAA9AC0}"
EndProject
Global
   GlobalSection(SolutionConfiguration) = preSolution
      ConfigName.0 = Debug
      ConfigName.1 = Release
   EndGlobalSection
   GlobalSection(ProjectDependencies) = postSolution
   EndGlobalSection
   GlobalSection(ProjectConfiguration) = postSolution
      {7B365305-6DF4-4F4D-AA6B-
0FE65B604563}.Debug.ActiveCfg = Debug|.NET
      {7B365305-6DF4-4F4D-AA6B-
0FE65B604563}.Debug.Build.0 = Debug|.NET
      {7B365305-6DF4-4F4D-AA6B-
0FE65B604563}.Release.ActiveCfg = Release|.NET
      {7B365305-6DF4-4F4D-AA6B-
0FE65B604563}.Release.Build.0 = Release|.NET
      {2533BEE1-4B39-4A31-A7CC-
2C10FAAA9AC0}.Debug.ActiveCfg = Debug|.NET
      {2533BEE1-4B39-4A31-A7CC-
2C10FAAA9AC0}.Debug.Build.0 = Debug|.NET
      {2533BEE1-4B39-4A31-A7CC-
2C10FAAA9AC0}.Release.ActiveCfg = Release|.NET
      {2533BEE1-4B39-4A31-A7CC-
2C10FAAA9AC0}.Release.Build.0 = Release|.NET
```

```
    EndGlobalSection
    GlobalSection(ExtensibilityGlobals) = postSolution
    EndGlobalSection
    GlobalSection(ExtensibilityAddIns) = postSolution
    EndGlobalSection
EndGlobal
```

File VBPROJ

File VBPROJ là một file XML để mô tả việc thiết lập project. Bạn sẽ thấy một số đoạn rất khác nhau trong file này. Bạn có một danh sách các xác lập cấu hình Debug và Release. Vào sâu hơn bạn sẽ thấy danh sách References tới các DLL khác cần có trong project này. Bạn cũng sẽ thấy danh sách các file thiết lập project đặc biệt này.

```
<VisualStudioProject>
    <VisualBasic
        ProjectType = "Local"
        ProductVersion = "7.0.9254"
        SchemaVersion = "1.0"
        ProjectGuid     =     "{7B365305-6DF4-4F4D-AA6B-
3FE6F3B6-1563}"
    >
        <Build>
            <Settings
                ApplicationIcon = ""
                AssemblyKeyContainerName = ""
                AssemblyName = "DotNetLib"
                AssemblyOriginatorKeyFile = ""
                AssemblyOriginatorKeyMode = "None"
                DefaultClientScript = "JScript"
                DefaultHTMLPageLayout = "Grid"
                DefaultTargetSchema = "IE50"
                DelaySign = "false"
                OutputType = "Library"
                OptionCompare = "Binary"
                OptionExplicit = "On"
                OptionStrict = "Off"
                RootNamespace = "DotNetLib"
                StartupObject = "DotNetLib.(None)"
            >
```

```
    <Config
        Name = "Debug"
        BaseAddress = "285212672"
        ConfigurationOverrideFile = ""
        DefineConstants = ""
        DefineDebug = "true"
        DefineTrace = "true"
        DebugSymbols = "true"
        IncrementalBuild = "true"
        Optimize = "false"
        OutputPath = "bin\"
        RegisterForComInterop = "false"
        RemoveIntegerChecks = "false"
        TreatWarningsAsErrors = "false"
        WarningLevel = "1"
    />
    <Config
        Name = "Release"
        BaseAddress = "285212672"
        ConfigurationOverrideFile = ""
        DefineConstants = ""
        DefineDebug = "false"
        DefineTrace = "true"
        DebugSymbols = "false"
        IncrementalBuild = "false"
        Optimize = "false"
        OutputPath = "bin\"
        RegisterForComInterop = "false"
        RemoveIntegerChecks = "false"
        TreatWarningsAsErrors = "false"
        WarningLevel = "1"
    />
</Settings>
<References>
    <Reference
        Name = "System"
        AssemblyName = "System"
    />
    <Reference
        Name = "System.Data"
        AssemblyName = "System.Data"
    />
    <Reference
        Name = "System.XML"
        AssemblyName = "System.Xml"
    />
</References>
```

```
            <Imports>
                <Import                 Namespace
"Microsoft.VisualBasic" />
                <Import Namespace = "System" />
                <Import                 Namespace              =
"System.Collections" />
                <Import Namespace = "System.Data" />
                <Import                 Namespace              =
"System.Diagnostics" />
            </Imports>
        </Build>
        <Files>
            <Include>
                <File
                    RelPath = "AssemblyInfo.vb"
                    SubType = "Code"
                    BuildAction = "Compile"
                />
                <File
                    RelPath = "clsLine.vb"
                    SubType = "Code"
                    BuildAction = "Compile"
                />
                <File
                    RelPath = "clsServerTime.vb"
                    SubType = "Code"
                    BuildAction = "Compile"
                />
            </Include>
        </Files>
    </VisualBasic>
</VisualStudioProject>
```

Tóm tắt

Chương này mô tả tiến trình thiết kế và bổ sung các thành phần .NET đơn giản như các DLL. Mỗi .NET DLL có thể chứa nhiều lớp, mỗi lớp trình bày một số đặc tính, phương thức và các sự kiện. Trong chương này bạn đã biết cách phát triển một module lớp thực hiện tác vụ đơn giản nhưng hữu dụng, cách thiết lập project thành phần dựa vào module lớp, sau đó là cách kết

hợp thành phần đó vào project WinForms của VB.NET. Các thành phần Visual Studio không cần được đăng ký trên máy tính của người dùng, hợp ngữ của người dùng phải có tham chiếu với DLL thành phần để vận hành.

Câu hỏi ôn tập

1. Một .NET DLL đơn giản có thể chứa bao nhiêu lớp?

   (a) Một và chỉ một.

   (b) Không hơn 2.

   (c) Nhiều.

2. Mã của một lớp bên trong module mã được định biên bởi các câu lệnh nào?

3. Một đặc tính được chỉ định là chỉ đọc bằng cách nào?

4. Tại sao bạn cần thêm nhiều project bên trong một solution?

5. Đoạn mã sau có gì sai?

```
ReadOnly Property TimeStamp() As String
  Get
    Try
      mTime = Now()
      Return CStr(mTime)
    Catch
      Return "Error!"
   End Get
End Property
```

6. Tại sao bạn muốn xây dựng cấu tử lớp riêng?

Bài tập

- Tạo thành phần ServerTime ở chương này.
- Sử dụng thành phần ServerTime đó từ một ứng dụng khác.

Trả lời câu hỏi ôn tập

1. (c) đúng. Một .NET DLL có thể chứa nhiều lớp khi cần. Nếu cần, nó có thể sắp xếp các lớp này trong NameSpaces tách riêng bên trong DLL.

2. Các câu lệnh Class..End Class định biên một lớp bên trong một module mã. Vì VS.NET hỗ trợ nhiều cặp câu lệnh Class..End Class bên trong một module, mỗi module trong project Visual Studio.NET có thể chứa nhiều lớp.

3. Từ khóa Readonly được dùng như một phần của khai báo đặc tính chỉ đọc:

```
ReadOnly Property TimeStamp() As String
```

4. Các project thành phần không thể được khởi động như các ứng dụng Windows độc lập. Thêm project thứ hai vào môi trường Visual Studio cung cấp một cách kiểm tra một thành phần thuận tiện trước khi nó thực sự được chuyển tới người dùng.

5. Không có câu lệnh End Try nào đóng khối được mở bằng câu lệnh Try. Thông thường, Visual Studio làm cho dễ thấy các câu lệnh bị thiếu bằng cách gạch dưới các câu lệnh

phải được so khớp bởi các câu lệnh khác. Còn phải có một câu lệnh Catch là thành phần của khối Try..End Try.

6. Một cấu tử là một thủ tục tiến hành khi một đối tượng được lập thể nghiệm từ một lớp. Đối số được chuyển tới cấu tử là cách thuận tiện để khởi tạo đối tượng khi nó được tạo. Thậm chí không có đối số, cấu tử có thể được đòi hỏi trong một số ứng dụng để khởi tạo các đối tượng, được xây dựng từ các lớp của một thành phần.

Chương 15

Khái niệm về control ListBox

- Tìm hiểu cách dùng control List Box

Hộp danh sách

Control hộp danh sách (list box) được dùng khi bạn cần hiển thị một danh sách các mục với người dùng và cần lựa một hoặc nhiều mục. Nếu có nhiều mục quá không vừa cửa sổ cho danh sách, control sẽ tự động thêm một thanh cuộn dọc vào bên phải của control để bạn có thể di chuyển qua toàn bộ danh sách.

Một control hộp danh sách có một tập hợp Items để lưu giữ một hoặc nhiều đối tượng. Nếu bạn quen dùng các hộp danh sách của VB6, bạn biết là chỉ được phép thêm các chuỗi vào hộp danh sách. Trong VB.NET, bạn có thể nhập một kiểu đối tượng bất kỳ vào hộp danh sách. Hộp danh sách sẽ nhận bất kỳ kiểu đối tượng nào và sẽ gọi phương thức ToString của mỗi đối tượng để xác định hiển thị những thông tin văn bản gì cho mỗi mục trong danh sách.

Hình 1: Việc nạp một hộp danh sách có thể được thực hiện bằng nhiều cách.

Thêm các mục vào hộp danh sách

Có một vài phương thức khác bạn có thể dùng để nạp hộp danh sách có dữ liệu. Dù bạn chọn dùng phương thức nào đi nữa, tất cả chúng đều được dùng trên tổ hợp Items trong hộp danh sách.

Phương thức Add

Đầu tiên bạn sẽ tìm hiểu cách sử dụng phương thức Add như được trình bày ở mã sau.

```
Private Sub btnAdd_Click(ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles btnLoad1.Click
    lstNames.Items.Clear()

    lstNames.Items.Add("Paul")
```

```
    lstNames.Items.Add("Ken")
    lstNames.Items.Add("Michael")
End Sub
```

Phương thức Add trên tổ hợp Items sẽ thêm một mục vào tổ hợp Items và đặt dữ liệu vào cuối danh sách. Nếu đặc tính Sorted được xác lập là True, thì dữ liệu sẽ tự động được sắp xếp sau khi được thêm vào tổ hợp Items.

Sử dụng phương thức Insert

Nếu bạn đã xác lập đặc tính Sorted là False và muốn thêm các mục ở vị trí riêng biệt, bạn có thể sử dụng phương thức Insert trên tổ hợp Items

```
Private Sub btnInsert_Click(ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles btnLoad2.Click
    lstNames.Items.Clear()

    lstNames.Sorted = False
    lstNames.Items.Insert(0, "Paul")
    lstNames.Items.Insert(1, "Ken")
    lstNames.Items.Insert(0, "Michael")

    lstNames.SetSelected(0, True)
End Sub
```

Ở mã trên, bạn thấy chuỗi "Paul" đã được thêm vào mục nhập đầu tiên trong hộp danh sách, kế tiếp là "Ken" được thêm vào vị trí thứ hai. Cuối cùng là "Michael" được thêm vào vị trí đầu tiên. Điều này khiến cho hai mục khác được chuyển xuống dưới trong tổ hợp Items.

Sử dụng phương thức AddRange

Thay vì thêm dữ liệu mỗi lần vào một mục, nếu bạn có một mảng dữ liệu mà bạn muốn đặt vào hộp danh sách, bạn có thể dùng phương thức AddRange của tổ hợp Items. Nó mong đợi một mảng giá trị kích thước đơn bạn muốn hiển thị trong hộp danh sách.

```
Private Sub btnAddRange_Click(
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnAddRange.Click
    Dim astrValues() As String = _
     {"Paul", "Ken", "Michael"}

    lstNames.Items.Clear()

    lstNames.Items.AddRange(astrValues)
End Sub
```

Truy tìm giá trị từ hộp danh sách

Một khi dữ liệu được nạp vào hộp danh sách, rất có thể bạn sẽ cần truy tìm dữ liệu đó khi bạn nhấp vào một mục. Bạn có thể viết mã trong thủ tục sự kiện SelectedIndexChanged của hộp danh sách để truy tìm mục ấy trong hộp danh sách. Đặc tính SelectedIndex được điền vào bởi control List Box với số mục bạn đã nhấp. Sau đó bạn có thể sử dụng nó như một chỉ mục vào tổ hợp Items để truy tìm mục của hộp danh sách cá biệt.

```
Private Sub lstNames_SelectedIndexChanged(_
 ByVal sender As Object, ByVal e As System.EventArgs) _
 Handles lstNames.SelectedIndexChanged

    txtName.Text = lstNames.Items( _
     lstNames.SelectedIndex).ToString()

End Sub
```

Các hộp danh sách nhiều lựa chọn

Theo mặc định control List Box sẽ chỉ cho phép bạn lựa mỗi lần một mục. Tuy nhiên, bạn có thể xác lập đặc tính SelectionMode để thay đổi ứng xử này. Bảng 1 trình bày các giá trị khác nhau mà đặc tính SelectionMode có thể có.

Giá trị	Mô tả
None	Chỉ có thể lựa mỗi lần một mục.
One	(Mặc định). Mỗi lần chỉ có thể lựa một mục.
MultiSimple	Nhấp chuột hoặc ấn phím dấu cách sẽ lựa hoặc thôi lựa một mục.
MultiExtended	Nhấp chuột, sau đó ấn giữ phím Shift và di chuyển chuột xuống trong danh sách sẽ tô sáng tất cả các mục từ mục đã lựa tới vị trí mới này. Ấn phím Ctrl khi nhấp chuột sẽ lựa hoặc thôi lựa một mục.

Bảng 1: *Đặc tính SelectionMode xác định cách lựa các mục bên trong một hộp danh sách hoạt động.*

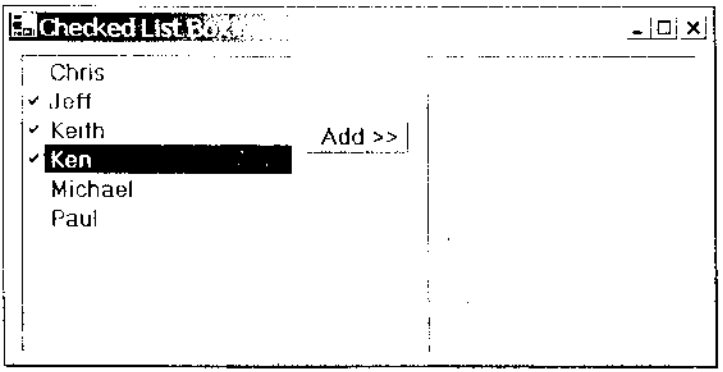
Hình 2: *Các hộp danh sách Multi-Select có thể lựa nhanh nhiều mục từ một hộp danh sách.*

Bạn có thể nạp các hộp danh sách nhiều lựa chọn bằng cách dùng bất kỳ một trong các phương thức nạp bạn đã tìm hiểu ở chương này. Nếu bạn xác lập đặc tính SelectionMode là MultiExtended, sau đó bạn có thể dùng chuột cùng với bàn phím như được mô tả ở bảng 1 để lựa nhiều mục trong hộp danh sách này. Một khi bạn đã lựa các mục đó, bạn có thể nhấp nút Add trên form để di chuyển các mục hộp danh sách này sang hộp danh sách khác. Bên dưới là mã bạn sẽ viết dưới nút Add này để di chuyển dữ liệu.

```
Private Sub btnAdd_Click(ByVal sender As Object, ByVal
e As System.EventArgs) Handles btnAdd.Click
    Dim intLoop As Integer

    For intLoop = lstUnSelected.Items.Count - 1 To 0 _
     Step -1
        If lstUnSelected.GetSelected(intLoop) Then
            lstSelected.Items.Add( _
             lstUnSelected.Items(intLoop))

            lstUnSelected.Items.RemoveAt(intLoop)
        End If
    Next
End Sub
```

Hộp danh sách được chọn

Vấn đề ở hộp danh sách nhiều lựa chọn là người dùng không biết chỉ cần xem nó và có thể lựa một hoặc nhiều mục. Do đó họ có thể không biết có thể lựa một hoặc nhiều mục. Nếu bạn muốn người dùng dễ nhận ra điều đó, bạn có thể dùng control CheckedListBox từ hộp công cụ VB.NET. Nó là một control tách từ control ListBox và sẽ trình bày dữ liệu với người dùng bằng một hộp nhỏ bên mỗi mục. Khi người dùng nhấp một mục trong hộp danh sách, một dấu kiểm sẽ xuất hiện bên mục dữ liệu như được trình bày ở hình 3.



Hình 3: *Các hộp danh sách được chọn đôi khi dễ nhận biết hơn các hộp danh sách nhiều chọn lựa.*

Control CheckedListBox hỗ trợ đặc tính SelectionMode giống như hộp danh sách thông thường. Do đó bạn cũng có thể dùng cùng các tổ hợp chuột và bàn phím với control này như bạn có thể làm với hộp danh sách thông thường. Tuy nhiên, đối với người dùng không biết điều này, họ sẽ có thể lựa riêng từng mục và thấy từng mục được lựa bởi một hộp kiểm bên các mục đó.

Để di chuyển dữ liệu từ một hộp danh sách được chọn sang hộp danh sách bên kia, bạn nhấp nút Add như được trình bày ở hình 3 và sẽ gọi ra thủ tục sự kiện Click cho nút đó. Hãy dùng tổ hợp CheckedItems để truy tìm tổ hợp các mục đã được lựa. Lặp qua tổ hợp này để nhấp kéo từng mục được lựa và thêm chúng vào control hộp danh sách đã chọn khác ở bên phải của form này.

```
Private Sub btnAdd_Click(ByVal sender As Object,
 ByVal e As System.EventArgs) Handles btnAdd.Click
    Dim oItems As CheckedListBox.CheckedItemCollection
    Dim intLoop As Integer

    ' Move items from checked to unchecked list
    oItems = clstUnSelected.CheckedItems
    For intLoop = 0 To oItems.Count - 1
        clstSelected.Items.Add(oItems(intLoop))
```

```
    Next

    ' Clear items using object
    oItems = clstUnSelected.CheckedItems
    For intLoop = oItems.Count - 1 To 0 Step -1
        clstUnSelected.Items.Remove(oItems(intLoop))
    Next
End Sub
```

Để xóa các mục trong hộp danh sách đã lựa ban đầu, bạn dùng mục Remove như phương thức này mong đợi bạn chuyển cho nó tham chiếu với đối tượng chính xác trong hộp danh sách bạn muốn xóa.

Nạp các đối tượng vào hộp danh sách

Cho tới điểm này bạn đã nạp các chuỗi vào một control hộp danh sách. Control hộp danh sách .NET hỗ trợ khả năng nạp hầu như bất kỳ đối tượng nào vào mỗi mục. Đây là một tính năng rất tốt mà bạn có thể tạo một lớp với một, hai hoặc nhiều đặc tính, lập thể nghiệm lớp này và nạp đối tượng đó vào hộp danh sách. Một trong các đặc tính có thể được dùng để hiển thị phần chuỗi trong hộp danh sách khi phần đối tượng còn lại chỉ được dùng để lưu giữ dữ liệu khác về mục đó trong hộp danh sách. Hình 4 trình bày màn hình thông tin Product tiêu biểu, trong đó hộp danh sách đang lưu giữ đối tượng sản phẩm với bốn đặc tính về đối tượng có thể được truy tìm và được hiển thị trong các control hộp văn bản thích hợp.

Hình 4: *Bạn có thể lưu trữ các đối tượng vào control danh sách.*

Nếu bạn đã sử dụng VB6 trước đây, bạn biết là chỉ có thể nạp một chuỗi duy nhất vào control hộp danh sách. Control hộp danh sách trong VB6 còn hỗ trợ đặc tính ItemData mà nó có thể lưu trữ giá trị long integer. Bạn có thể lưu trữ một giá trị số nguyên dài cho mỗi giá trị bạn đã nạp vào tổ hợp List. Tuy nhiên, vấn đề là bạn chỉ có thể lưu trữ vào đặc tính này. Nếu bạn muốn duy trì một danh sách các đối tượng, bạn phải tạo một mảng khác của các đối tượng đó, nạp dữ liệu chuỗi vào hộp danh sách, sau đó sử dụng ItemData để lưu giữ chỉ mục vị trí mà đối tượng đó được định vị trong mảng. Điều này phải làm nhiều tác vụ.

Chìa khóa để tạo công việc này là tạo một lớp có các đặc tính đầy đủ để lưu giữ dữ liệu bạn muốn đặt vào hộp danh sách. Đương nhiên, một đặc tính của lớp này sẽ được dùng để hiển thị dữ liệu trong phần văn bản của hộp danh sách và do đó phải thuộc kiểu dữ liệu chuỗi. Đặc tính khác hoặc các đặc tính có thể được dùng để lưu giữ bất kỳ dữ liệu nào bạn muốn. Thí dụ, nếu bạn tạo một đối tượng biểu diễn mỗi dòng trong một bảng CSDL,

thì bạn có thể dùng một số đặc tính khác để lưu giữ thông tin quan trọng nhất.

Ở các chương trước, bạn đã tạo một form Products với bốn hộp văn bản trên nó, gồm Product ID, Product Name, Unit Price và Units in Stock. Chúng ta hãy tạo một lớp lưu giữ các đoạn thông tin này.

```
Public Class Product
    Private mintProductID As Integer
    Private mstrProductName As String
    Private mdecUnitPrice As Decimal
    Private msrtUnitsInStock As Short

    Property ProductID() As Integer
        Get
            Return mintProductID
        End Get
        Set(ByVal Value As Integer)
            mintProductID = Value
        End Set
    End Property

    Property ProductName() As String
        Get
            Return mstrProductName
        End Get
        Set(ByVal Value As String)
            mstrProductName = Value
        End Set
    End Property

    Property UnitPrice() As Decimal
        Get
            Return mdecUnitPrice
        End Get
        Set(ByVal Value As Decimal)
            mdecUnitPrice = Value
        End Set
    End Property

    Property UnitsInStock() As Short
        Get
            Return msrtUnitsInStock
```

```
        End Get
        Set(ByVal Value As Short)
            msrtUnitsInStock = Value
        End Set
    End Property

    Public Overrides Function ToString() As String
        Return mstrProductName
    End Function
End Class
```

Lưu ý, ở lớp Product trên có một hàm ghi đè có tên ToString. Bạn đã thấy phương thức này được dùng để chuyển đổi các kiểu numeric thành chuỗi. Tất cả các lớp bạn tạo tự động thừa kế lớp cơ sở Object. Lớp cơ sở này luôn luôn có phương thức ToString, theo mặc định nó cho trở lại tên của chính lớp ấy. Control List Box mong đợi bất kỳ đối tượng nào được đặt vào tổ hợp Items của nó để có phương thức ToString cho nó gọi. Do đó, bạn phải cung cấp phương thức ToString để cho trở lại giá trị chuỗi bạn muốn đặt vào hiển thị hộp danh sách.

Lưu ý:

Thay vì tạo phương thức ToString, bạn cũng có thể xác lập đặc tính DisplayMember là tên của đặc tính trong đối tượng của bạn muốn có lời gọi ListBox để truy tìm dữ liệu.

Để nạp một loạt các sản phẩm vào hộp danh sách, bạn chỉ cần tạo từng đối tượng sản phẩm và thêm đối tượng mới đó vào control hộp danh sách. Bên dưới là mã tạo ba đối tượng sản phẩm, xác lập các đặc tính của chúng và sau đó thêm các đối tượng đó vào control hộp danh sách.

```
Private Sub ListLoad()
    Dim oProd As Product

    oProd = New Product()
```

```
    With oProd
        .ProductID = 1
        .ProductName = "Yellow Cake Mix"
        .UnitPrice = CDec(10.95)
        .UnitsInStock = 10
    End With
    lstProducts.Items.Add(oProd)


    oProd = New Product()
    With oProd
        .ProductID = 2
        .ProductName = "Sponge"
        .UnitPrice = CDec(5.54)
        .UnitsInStock = 20
    End With
    lstProducts.Items.Add(oProd)

    oProd = New Product()
    With oProd
        .ProductID = 3
        .ProductName = "Coffee"
        .UnitPrice = CDec(4.96)
        .UnitsInStock = 25
    End With
    lstProducts.Items.Add(oProd)
End Sub
```

Khi bạn cần truy tìm một đối tượng từ control hộp danh sách, bạn sẽ viết lại mã trong thủ tục sự kiện SelectedIndexChanged. Bạn sẽ cần trích mục ấy từ hộp danh sách bằng cách dùng đặc tính SelectedIndex và sử dụng nó như chỉ mục vào tập hợp Items. Chuyển đổi đối tượng trả lại từ hộp danh sách thành đối tượng Product bằng cách dùng hàm CType.

```
Private Sub lstProducts_SelectedIndexChanged( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles lstProducts.SelectedIndexChanged
    Dim oProd As Product

    oProd = CType(lstProducts.Items( _
     lstProducts.SelectedIndex), Product)
    With oProd
```

```
            txtProductID.Text = .ProductID.ToString()
            txtProductName.Text = .ProductName
            txtUnitPrice.Text = .UnitPrice.ToString()
            txtUnitsInStock.Text = .UnitsInStock.ToString()
        End With
    End Sub
```

Sau khi truy tìm đối tượng Products, bạn có thể dùng các đặc tính của đối tượng này để đặt dữ liệu vào các control văn bản thích hợp trên form này.

Tóm tắt

Ở chương này, bạn đã tìm hiểu cách dùng control List Box bên trong VB.NET. Control danh sách này mạnh hơn control danh sách trước đây trong VB6 nhiều. Bây giờ bạn có khả năng lưu trữ các đối tượng bên trong control danh sách. Còn một control mới cho phép bạn đặt dấu kiểm vào bên mục đã lựa. Ngoài ra, control Combo Box trong .NET cũng hỗ trợ tất cả các tính năng tương tự được thảo luận trong chương này về các hộp danh sách.

Câu hỏi ôn tập

1. Bạn dùng phương thức nào để nạp dữ liệu khi bạn không quan tâm đến thứ tự nó xuất hiện trong control danh sách?

2. Phương thức nào sẽ cho phép bạn chỉ định vị trí chính xác bạn muốn đặt một mục dữ liệu?

3. Phương thức nào cho phép bạn thêm một mảng các mục vào hộp danh sách?

4. Bạn cần phương thức nào để tạo trên các lớp của riêng bạn sẽ được hộp danh sách dùng để truy tìm dữ liệu hiển thị?

5. Đặc tính nào được xác lập với số hiệu mục được lựa khi bạn nhấp vào một mục trong hộp danh sách?

Bài tập

1. Tạo một form với hai hộp danh sách bên nhau như được trình bày ở hình 2.

2. Tạo lớp Products như lớp được trình bày ở chương này.

3. Nạp hộp danh sách ở bên trái với 3 tới 4 đối tượng sản phẩm.

4. Viết mã để có thể lựa nhiều đối tượng và di chuyển chúng vào hộp danh sách ở bên phải form.

Trả lời câu hỏi ôn tập

1. Add.

2. Insert.

3. AddRange.

4. ToString.

5. SelectedIndex.

Chương 16

ADO.NET

- Giới thiệu về các lớp ADO.NET.
- So sánh ADO.NET với ADO.

ADO.NET

ADO.NET là một mô hình đối tượng cho bạn xây dựng các tập dữ liệu trong bộ nhớ. Bạn có thể tạo các tập dữ liệu từ mã cứng (hard-code) của bạn trong chương trình, truy tìm từ một file văn bản, truy tìm từ một máy chủ trao đổi (exchange server) hoặc từ một hệ thống CSDL. ADO.NET không cần quan tâm tới vị trí dữ liệu đến từ đâu, miễn là bạn có thể đặt nó vừa vào các dòng và các cột bên trong các đối tượng của nó.

ADO.NET có một số lớp mà bạn sẽ cần tìm hiểu và sử dụng. Bảng 1 cung cấp một danh sách và mô tả tóm lược về mỗi lớp bạn sẽ gặp ở một số chương kế tiếp.

Lớp	Mô tả
DataSet	Lớp này giống một CSDL trong bộ nhớ. Nó có thể chứa một hoặc nhiều đối tượng DataTable. Bạn cũng có thể có khả năng xác lập các mối quan hệ giữa các đối tượng DataTable này. Lớp này có thể lưu giữ tập dữ liệu gốc và bất kỳ các sửa đổi nào xảy ra với dữ liệu này. Sau đó bạn có thể dùng các đối tượng ADO.NET khác để gửi các thay đổi này trở lại nguồn dữ liệu. Lớp này không biết cách truyền thông với bất kỳ kiểu nguồn dữ liệu nào.
DataTable	Lớp này lưu giữ các dòng dữ liệu. Mỗi dòng được thiết lập bởi các cột và mỗi cột có thể lưu giữ một số dữ liệu riêng. Lớp này không biết cách truyền thông với bất kỳ kiểu nguồn dữ liệu nào.
DataView	Lớp này là khung xem riêng của DataTable. Đối tượng này thường được dùng để xác lập các thứ tự sắp xếp hoặc các bộ lọc trên đối tượng DataTable gốc. Lớp này không biết cách truyền thông với bất kỳ kiểu nguồn dữ liệu nào.
Connection	Lớp này được dùng để tạo một kết nối với nguồn dữ liệu. Lớp này có hai kiểu mặc định, một cho các nguồn dữ liệu OLE DB và một dành riêng cho SQL Server. Bạn có thể mở một kết nối với nguồn dữ liệu và duy trì đối tượng kết nối mở bao lâu bạn cần.
Command	Lớp này được dùng để đệ trình các câu lệnh SQL tới nguồn dữ liệu của chương trình phụ trợ (back-end). Bạn có thể dùng SQL hoặc các lời gọi các thủ tục lưu trữ cho tất cả sửa đổi và truy tìm dữ liệu. Lớp này có các

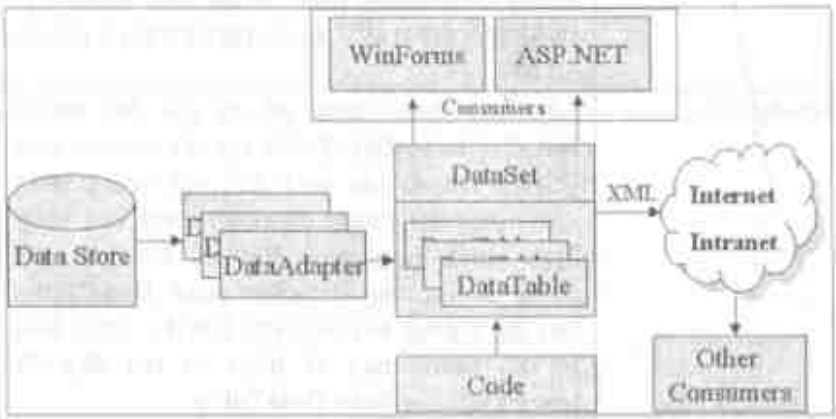
	phương thức để đệ trình SQL để sử dụng dữ liệu và các phương thức truy tìm dữ liệu.
CommandBuilder	Lớp này được dùng để xây dựng các đối tượng Command, nhất là dữ liệu SELECT, INSERT, UPDATE hoặc DELETE từ nguồn dữ liệu.
DataAdapter	Lớp này được dùng để cư trú đối tượng DataSet hoặc DataTable với dữ liệu từ một CSDL. Nó sẽ tạo một kết nối với CSDL, xây dựng đối tượng command với câu lệnh SQL thích hợp, truy tìm dữ liệu và xây dựng đối tượng DataSet hoặc DataTable, sau đó ngưng kết nối với CSDL. Xuất liệu từ đối tượng này sẽ được cư trú đầy đủ trong DataSet hoặc DataTable.
DataReader	Lớp này là con trỏ chỉ tiếp tới (forward-only) và chỉ đọc (đôi khi còn được gọi là con trỏ vòi rồng), sẽ đọc nhanh dữ liệu từ một nguồn dữ liệu. Lớp này để điền các khung lưới dữ liệu, các hộp danh sách và các hộp combo rất tốt.

Bảng 1: *Các đối tượng của ADO.NET.*

Sử dụng các lớp ADO

Hình 1 trình bày cách một số lớp trong .NET làm việc để truy tìm dữ liệu, tạo các DataTable bên trong một DataSet và cách các DataSet đó có thể được sử dụng từ một số vị trí. Các DataSet có thể được tạo từ các DataAdapter mà nó đọc từ một kho dữ liệu, chẳng hạn như SQL Server, Oracle hoặc Access. Các DataSet cũng có thể được tạo bởi mã bạn viết bằng VB.NET hoặc C#. Một khi DataSet được xây dựng, nó có thể được sử dụng bởi WinForm,

WebForm, Web Service hoặc nó có thể được gửi qua giao diện HTTP tới một máy của người dùng khác, chẳng hạn như một trang web được viết bằng Java chạy trên máy chủ Unix.



Hình 1: *Tổng quan về các lớp .NET.*

OleDb và SqlClient

Các lớp Connection, Command, CommandBuilder, DataAdapter và DataReader có hai dạng. Một là tập các lớp này có tiền tố "OleDb" và sử dụng các nhà cung cấp OLE DB để tới nguồn dữ liệu. Tập còn lại có tiền tố "Sql" và sử dụng các nhà cung cấp địa phương để truyền thông trực tiếp với SQL Server. Hai tập lớp này đến từ các namespace tách riêng. Bên dưới là các tên đầy đủ bạn sẽ dùng để khai báo các lớp OleDb.

```
Dim oConn As OleDb.OleDbConnection
        Dim oCmd As OleDb.OleDbCommand
        Dim oBuild As OleDb.OleDbCommandBuilder
        Dim oDA As OleDb.OleDb.OleDbDataAdapter
Dim oDR As OleDb.OleDb.OleDbDataReader
```

Bên dưới là các tên đầy đủ bạn sẽ dùng để khai báo các lớp Sql.

```
Dim oConn As SqlClient.SqlConnection
        Dim oCmd As SqlClient.SqlCommand
        Dim oBuild As SqlClient.SqlCommandBuilder
        Dim oDA As SqlClient.SqlDataAdapter
Dim oDR As SqlClient.SqlDataReader
```

ADO.NET so với ADO

ADO.NET là một phiên bản tăng cường của ADO. Dù có một số khác biệt cơ bản giữa hai mô hình đối tượng, nhưng kết quả cuối cùng vẫn giống nhau. Bạn cần một mô hình đối tượng để cho phép lưu trữ dữ liệu, truy tìm dữ liệu và sửa đổi dữ liệu. ADO cho bạn thực hiện với các Recordset, ADO.NET cho bạn thực hiện với các DataSet và một vài đối tượng khác.

Một trong các khác biệt chính giữa ADO.NET và ADO là ADO.NET được thiết kế là một khung xem dữ liệu đã ngưng kết nối, trong khi ADO là kiểu kết nối mô hình đối tượng. Cả hai đều có ưu điểm và khuyết điểm. Bạn sẽ có nhận định này sau khi đã làm việc với ADO.NET.

Tóm tắt

Chương này cho bạn cái nhìn tổng quát về ADO.NET và giải thích một số đối tượng khác nhau bạn có thể dùng cùng với ADO.NET.

Câu hỏi ôn tập

1. Bạn dùng namespace nào để tìm kiếm một CSDL Oracle?
2. Bạn dùng namespace nào để đi thẳng tới SQL Server?
3. Lớp nào truy tìm dữ liệu từ nguồn dữ liệu?
4. Lớp nào sẽ xây dựng một đối tượng command mới cho bạn?
5. Đúng hay sai: DataSet có thể cập nhật dữ liệu trực tiếp trong một CSDL SQL Server?

Trả lời câu hỏi ôn tập

1. OleDb.
2. SqlClient.
3. DataAdapter.
4. CommandBuilder.
5. Sai.

Chương 17

Liên kết dữ liệu

- Tìm hiểu cách liên kết control DataGrid với DataSet
- Tìm hiểu cách liên kết control ComboBox với DataSet
- Tạo truy vấn tham số hóa

Liên kết dữ liệu của ADO.NET

Liên kết dữ liệu liên quan đến tiến trình tạo nguồn dữ liệu và tự động cung cấp nguồn dữ liệu đó cho (các) control trên một form. Các form của Windows sử dụng ADO.NET dưới các lớp vỏ để thực hiện liên kết dữ liệu. Với sự liên kết dữ liệu, bạn không cần viết mã cụ thể để lập thể nghiệm một kết nối và tạo dataset (khi bạn thực hiện với một form chưa liên kết). .NET framework viết mã ADO.NET cần thiết cho bạn.

Các form Windows cho phép bạn dễ dàng liên kết hầu như bất kỳ cấu trúc nào chứa dữ liệu. Điều này có nghĩa là bạn có thể

liên kết với các kho dữ liệu truyền thống, chẳng hạn như dữ liệu được lưu trữ trong Access hoặc bảng SQL Server hay với kết quả dữ liệu được đọc từ một file, được chứa trong các control hoặc được lưu trữ trong một mảng. Dữ liệu nhập vào cấu trúc như thế nào không quan trọng đối với các mục đích của chương này.

Sau khi bạn liên kết một form với dữ liệu, thì bạn liên kết các control trên form với các phần tử dữ liệu riêng biệt. Liên kết dữ liệu truyền thống nhất bao gồm việc liên kết đặc tính Text của control Textbox với một cột datasource. Bạn có thể liên kết đồ họa của một control Image, nền của một control hoặc bất kỳ một đặc tính khác của bất kỳ control nào trên một form.

Hai kiểu liên kết dữ liệu sẵn có cho các form Windows: Simple Data Binding (Liên kết dữ liệu đơn) và Complex Data Binding (Liên kết dữ liệu phức tạp). Mỗi phương thức cung cấp các ưu điểm khác nhau.

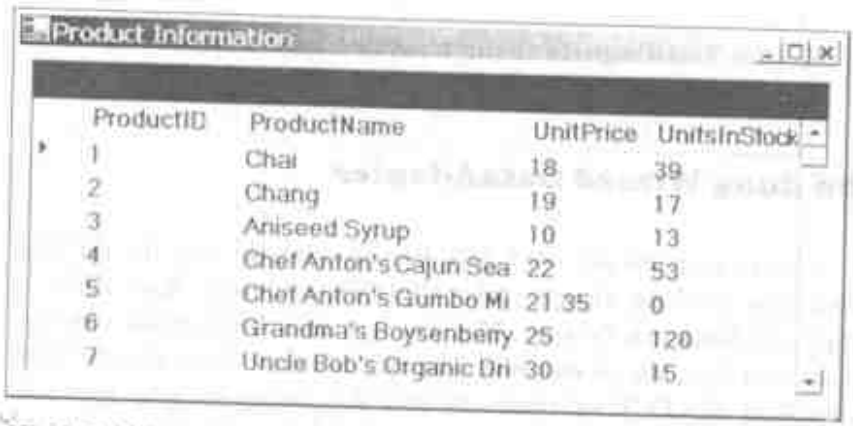
Liên kết dữ liệu đơn cho phép bạn liên kết một control với một phần tử dữ liệu đơn. Việc liên kết dữ liệu đơn được sử dụng thông dụng nhất bao gồm liên kết một phần tử dữ liệu đơn, chẳng hạn như giá trị của một cột trong một bảng, với một control trên form. Bạn sử dụng kiểu liên kết dữ liệu này cho các control chỉ biểu thị một giá trị. Việc sử dụng liên kết dữ liệu đơn bao gồm liên kết dữ liệu với các hộp văn bản và các nhãn.

Liên kết dữ liệu phức tạp cho phép bạn liên kết nhiều hơn một phần tử dữ liệu với một control. Thí dụ sử dụng dữ liệu cột, liên kết dữ liệu phức tạp bao gồm liên kết hơn một cột hoặc dòng từ nguồn bản ghi cơ bản. Các control hỗ trợ liên kết dữ liệu phức tạp bao gồm các control khung lưới dữ liệu, các hộp combo và các hộp danh sách.

Liên kết dữ liệu với DataGrid

Việc sử dụng liên kết dữ liệu cơ bản nhất có sẵn trong môi trường .NET hiển thị nội dung một bảng trong khung lưới. Thí dụ liệp quan đến các bước sau (không có chi tiết):

1. Xây dựng một form Windows.
2. Tạo và cấu hình dataset bạn muốn liên kết với form.
3. Thêm control khung lưới vào form và liên kết nó với dữ liệu.
4. Một thí dụ về form dẫn đến tiến trình này xuất hiện ở hình 1.



Hình 1: *Kết quả của việc tạo form Windows liên kết dữ liệu đơn.*

Tạo form mẫu

Sau là các bước bạn sẽ thực hiện để xây dựng form mẫu.

1. Mở Visual Studio.NET và nhấp nút New Project trên Start page.
2. Lựa Windows Application như Template bạn muốn xây dựng.
3. Xác lập Name là **DataBinding**.
4. Nhấp OK để tạo project.
5. Ấn F4 để hiển thị các đặc tính cho form. Thay đổi đặc tính Name của form là **frmProducts**.
6. Đổi Text Property thành **Product Information**.

Sử dụng Wizard DataAdapter

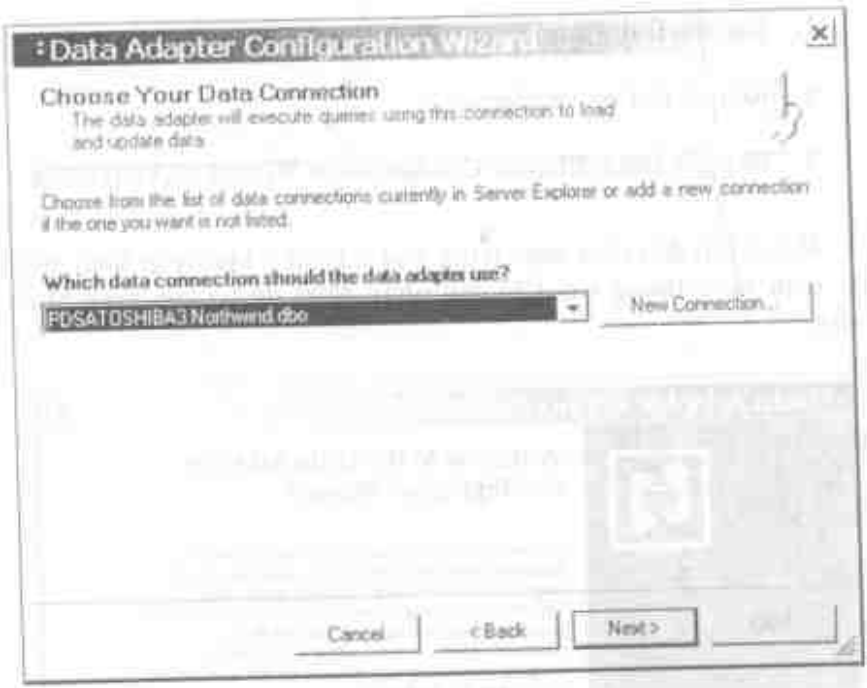
Để tạo một kết nối với CSDL bạn sẽ dùng để truy tìm dữ liệu, bạn cần sử dụng thành phần OleDbDataAdapter được định vị dưới tab Data của Toolbox. Một khi bạn kéo thành phần này vào form của bạn, nó sẽ đưa bạn qua một loạt các bước để cấu hình câu lệnh SELECT và chuẩn bị một đối tượng sẽ giúp bạn xây dựng lớp DataSet. DataAdapter chỉ thực hiện truy tìm từ CSDL. DataSet là những gì bạn sẽ dùng để liên kết với control DataGrid.

1. Lựa OleDbDataAdapter từ tab Data của Toolbox.
2. Kéo và thả nó vào form.
3. Bây giờ Data Adapter Configuration Wizard sẽ khởi động.

Màn hình đầu tiên được trình bày ở hình 2 không gì khác hơn là màn hình thông tin. Chỉ cần nhấp Next để bỏ qua màn hình này.

Hình 2: *Màn hình Welcome.*

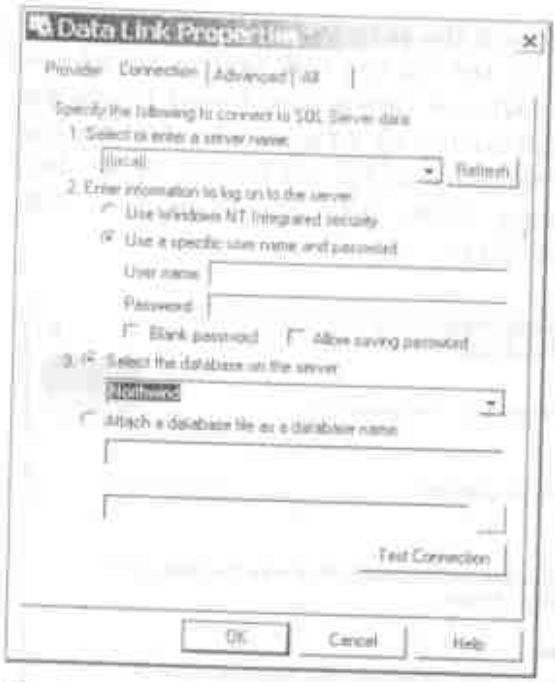
4. Nhấp Next để xem màn hình sẽ nhắc bạn về kết nối dữ liệu sử dụng. Nếu bạn chưa thiết lập kết nối dữ liệu, bạn phải nhấp nút New Connection.



Hình 3: *Chọn Data Connection.*

Nếu bạn chọn nhấp nút New Connection, bạn sẽ thấy một màn hình giống hình 4. Bạn có thể đã quen với màn hình này vì nó thường được dùng trong các ứng dụng ADO. Trên màn hình này là nơi bạn sẽ lựa nhà cung cấp dữ liệu bạn muốn dùng và sau đó điền thông tin thích hợp về cách kết nối với nhà cung cấp dữ liệu.

Ở thí dụ được trình bày ở hình 4, bạn đã chọn nhà cung cấp SQL Server và sẽ chọn server cục bộ.



Hình 4: *Hộp thoại Data Link Properties cho phép bạn chỉ định thông tin kết nối cho dữ liệu nằm dưới form liên kết.*

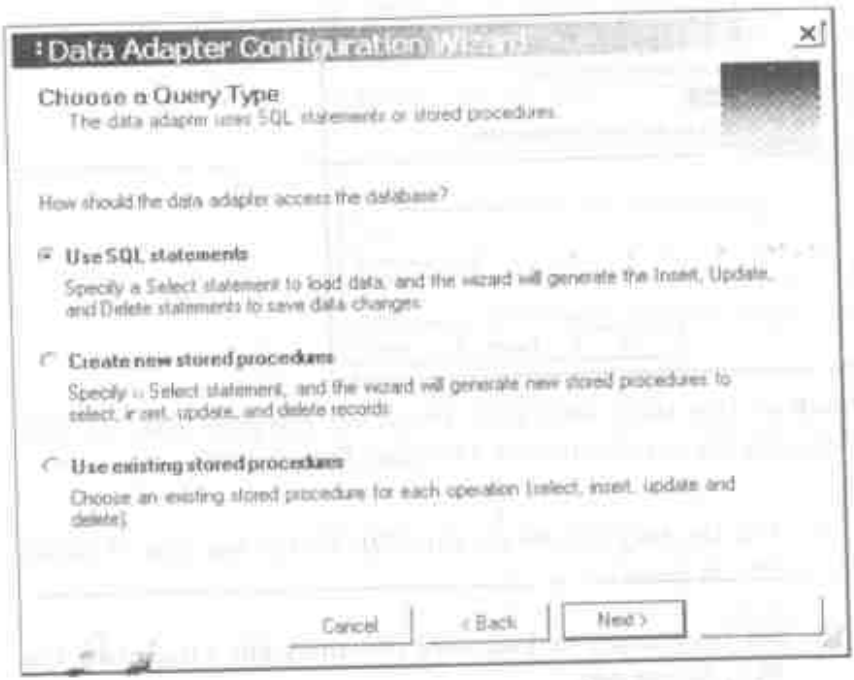
5. Đổi tên máy chủ để chỉ tên SQL Server của bạn. Ở thí dụ này là (local).

6. Đổi User name và Password cần thiết khi đăng nhập vào SQL Server này.

7. Lựa CSDL Northwind vì đây là nơi có bảng Products để truy tìm thông tin.

8. Nhấp OK để trở lại wizard và nhấp nút Next để di chuyển tới bước kế tiếp của wizard.

9. Bước kế tiếp của wizard cho phép bạn chỉ định Query Type (xem hình 5). Bạn có thể lựa Use SQL Statements để liên kết một câu lệnh SQL, lựa Create New Stored Procedures để wizard phát sinh các thủ tục dữ liệu vĩnh viễn hoặc Use Existing Stored Procedures để lựa các thủ tục đã lưu được định vị sẵn trong CSDL máy chủ. Lựa Use SQL statements và nhấp Next.



Hình *Chỉ định kiểu truy vấn form liên kết sẽ căn cứ vào.*

10. Gõ câu lệnh SQL hoặc nhấp Query Builder để wizard xây dựng câu lệnh SQL cho bạn. Trong thí dụ này, gõ câu lệnh SQL sau:

```
SELECT ProductID, ProductName, UnitPrice,
UnitsInStock
FROM Products
```

11. Nhấp Next để tiến hành tới bước cuối cùng của wizard và nhấp Finish để kết thúc tiến trình ấy. Lưu ý, đối tượng OleDbConnection và đối tượng OleDbDataAdapter có tên OleDbConnection1 và OleDbDataAdapter1 xuất hiện trong Tray của form.

Đối tượng OleDbConnection1 chứa thông tin về cách truy xuất CSDL đã lựa. Đối tượng OleDbDataAdapter1 chứa truy vấn định nghĩa các bảng và các cột trong CSDL bạn muốn truy xuất.

Lưu ý:
Ở thí dụ này, OleDbDataAdapter được lựa từ hộp công cụ, chính vì bạn có thể sẽ dùng CSDL nào đó khá hơn SQL Server. Nếu bạn truy xuất được SQL Server, bạn nên dùng các thành phần Sql* vì nó hiệu quả hơn. Các bước bạn thực hiện sẽ tương tự.

Tạo lớp Dataset

Như đã nói trước đây, tự thân DataAdapter không thể liên kết với control trên một form. Bạn cần có DataSet cho nó. Visual Studio.NET sẽ giúp bạn phát sinh lớp DataSet dựa vào truy vấn bạn đã tạo trong đối tượng DataAdapter. Để phát sinh lớp dataset mới này, thực hiện các bước sau.

1. Lựa Data | Generate Dataset từ menu VS.NET.

2. Bạn sẽ thấy hộp thoại Generate Dataset xuất hiện giống hình 6.

3. Nhấp nút tùy chọn New.

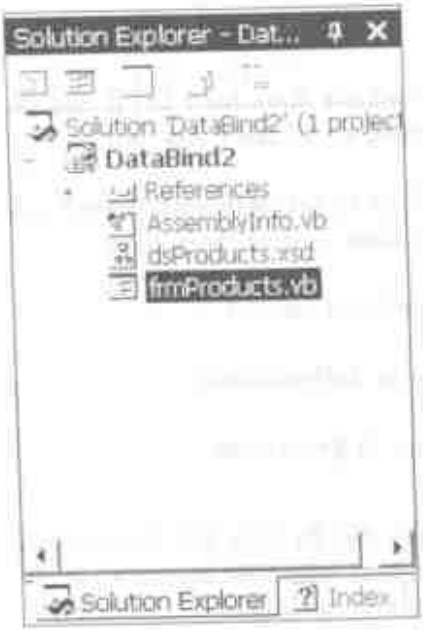
4. Gõ tên dsProducts vào.

5. Nhấp nút OK để phát sinh file định nghĩa giản đồ DataSet và lớp.

Hình 6: *Hộp thoại Generate Dataset cho phép bạn chỉ định tên và các thuộc tính khác của dataset bạn sẽ phát sinh.*

Sau khi bước này hoàn tất, bạn sẽ thấy một control mới, dsProducts1 trong Tray cho form này. Control mới này là một tham chiếu với file dsProducts.xsd cũng đã được thêm vào project của bạn. File dsProducts.xsd là một định nghĩa giản đồ XML chứa định nghĩa đầy đủ cho bảng và các cột của câu lệnh SQL bạn đã gõ vào trước đây. Có một lớp đằng sau file XSD này mà

bạn không thể thấy, trừ khi bạn lựa Project ➤ Show All Files từ menu. Sau đó bạn có thể nhấp vào dấu cộng xuất hiện đằng sau file dsProducts.xsd để xem file dsProducts.vb như được trình bày ở hình 7.



Hình 7: *File XML schema definition (XSD) có một module lớp, chứa mã nạp dataset vào bộ nhớ.*

Lớp dsProducts.vb có các đặc tính tương ứng với dataset thực và các đặc tính tương ứng với mỗi cột được chỉ định trong câu lệnh SQL của bạn. Mặc dù, bạn không cần làm gì với lớp này, nhưng biết được nó ở đó rất tốt.

Thêm DataGrid vào Form

Control DataGrid được thiết lập để sử dụng lớp DataSet để hiển thị dữ liệu. Bạn chỉ cần thêm control khung lưới dữ liệu vào form của bạn.

1. Phải chắc chắn form frmProducts được hiển thị ở chế độ thiết kế trong môi trường thiết kế VS.NET của bạn.
2. Kéo control khung lưới dữ liệu từ tab Windows Forms của Toolbox vào form. Lập kích thước control thích hợp.
3. Ấn F4 để hiển thị các đặc tính của control.
4. Xác lập đặc tính DataSource là **dsProducts1**.
5. Xác lập đặc tính DataMember là **Products**.

Các bước bạn vừa thực hiện, bây giờ đã liên kết dataset với control khung lưới dữ liệu. Chỉ còn một bước nữa sẽ hoàn tất để thấy dữ liệu xuất hiện trong control khung lưới.

Đưa dữ liệu vào control khung lưới dữ liệu

Mặc dù control khung lưới dữ liệu được liên kết với dataset, dataset ấy không tự động được đưa vào khi form được nạp. Sử dụng thủ tục sự kiện Load của form để đưa dữ liệu vào control khung lưới dữ liệu khi form được nạp.

```
Private Sub frmProducts_Load( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles MyBase.Load

    ' Load the Data Grid Control
```

```
        dsProducts1.Clear()
        OleDbDataAdapter1.Fill(dsProducts1, "Products")

    End Sub
```

Trong thủ tục sự kiện Load, đầu tiên bạn bỏ dataset và sau đó điền dataset bằng cách dùng phương thức Fill của đối tượng OleDbDataAdapter1 bạn đã tạo trước đây. Bạn cần chuyển vào tên bảng làm tham số thứ hai, vì nó được dùng bởi control khung lưới dữ liệu để truy tìm DataMember (Products) chính xác mà bạn đã chỉ định trước đây trong đặc tính DataMember.

Ở điểm này, bạn sẵn sàng xem dữ liệu từ bảng Products được hiển thị ở control khung lưới dữ liệu. Chỉ còn một bước nữa để hoàn tất trước khi bạn có thể chạy form này, bạn cần báo cho project biết form Startup của project này sẽ là gì.

1. Nhắp tên project của bạn trong cửa sổ Solution Explorer.
2. Nhắp nút chuột phải và chọn Properties từ menu.
3. Thả xuống hộp combo Startup Object và chọn frmProducts từ danh sách.
4. Nhắp OK.
5. Ấn **F5** để chạy project này và nếu bạn thực hiện mọi thứ chính xác, bây giờ bạn sẽ thấy dữ liệu sản phẩm bên trong control khung lưới dữ liệu giống màn hình được trình bày ở hình 1.

Làm việc với các hộp combo

Thí dụ bạn vừa tạo đáp ứng tốt cho các tập kết quả nhỏ, nhưng sẽ không đủ hiệu lực nếu bảng các sản phẩm có hàng

ngàn bản ghi. Vấn đề là nó phải truy tìm tất cả các sản phẩm từ bảng Products và hiển thị chúng trong control khung lưới dữ liệu. Mặc dù để minh họa được tường tận, nhưng kĩ thuật này không thực tế trong một ứng dụng về sản phẩm.

Máy chủ CSDL được tối ưu hóa để xử lý khối lượng dữ liệu lớn, nhưng bạn sẽ chỉ cho trở lại các tập kết quả nhỏ. Thí dụ được trình bày ở phần trước sẽ hiệu quả hơn nhiều nếu đầu tiên người dùng giới hạn dữ liệu bằng cách lựa một phân loại các sản phẩm đặc biệt từ hộp combo và sau đó chỉ hiển thị các sản phẩm đó cho phân loại đã lựa. Phần này trình bày cách tạo hộp combo dữ liệu liên kết. Phần kế tiếp trình bày cách sử dụng hộp combo để giới hạn dữ liệu được hiển thị trong control khung lưới dữ liệu. Thí dụ này bao gồm các bước căn bản sau:

1. Thêm adapter dữ liệu thứ hai vào form.
2. Phát sinh dataset thứ hai.
3. Thêm hộp combo vào form.
4. Liên kết hộp combo với adapter dữ liệu thứ hai.
5. Đưa dữ liệu từ adapter dữ liệu thứ hai vào hộp combo.

Thêm adapter dữ liệu thứ hai vào form

Trước khi bạn thêm một hộp combo vào form, đầu tiên hãy thêm adaper dữ liệu thứ hai vào form. Bạn sẽ dùng adapter dữ liệu này để phát sinh dataset chứa danh sách các phân loại từ bảng Categories trong CSDL Northwind. Để thêm adapter dữ liệu thứ hai, thực hiện các bước sau:

1. Lựa OleDbDataAdapter từ tab Data của Toolbox. Kéo và thả nó vào form. Data Adapter Configuration Wizard xuất hiện. Nhấp Next.

2. Lựa kết nối với CSDL Northwind bạn đã tạo ở thí dụ trước đây. Nhấp Next.

3. Lựa Use SQL Statements và nhấp Next.

4. Nhập câu lệnh SQL sau:

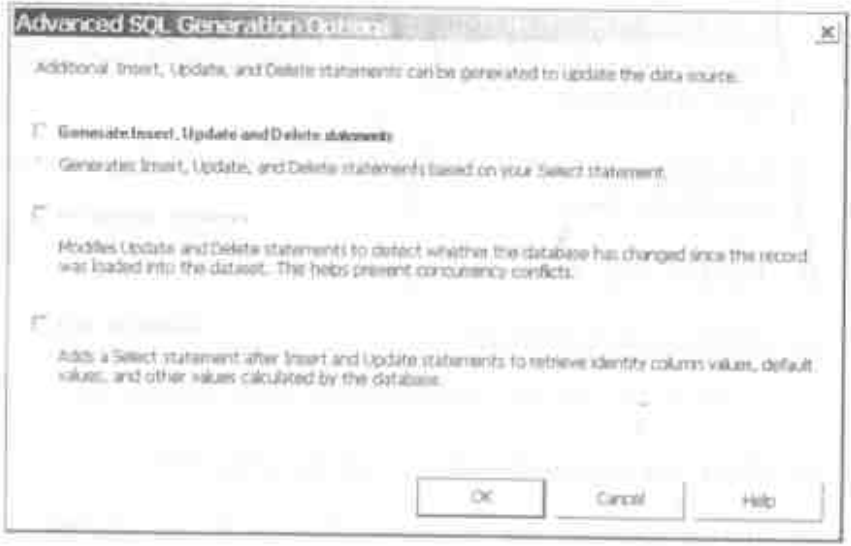
```
SELECT CategoryID, CategoryName FROM Categories
```

5. Nhấp nút Advanced Options trên hộp thoại này.

6. Bỏ các tùy chọn cho câu lệnh Generate Insert, Update và Delete như được trình bày ở hình 8. Các câu lệnh này không cần thiết đối với một dataset chỉ dùng để đưa vào một hộp combo. Nhấp OK để đóng hộp thoại này.

7. Nhấp Finish để phát sinh SQLDataAdapter.

Bây giờ sẽ có một đối tượng có tên SQLDataAdapter2 được thêm vào Tray của form.



Hình 8: *Đừng chọn câu lệnh Generate Insert, Update and Delete khi tạo một dataset được dùng để liên kết với hộp combo.*

Phát sinh Dataset thứ hai

Bước kế tiếp là phát sinh lớp dataset và file XSD cho OleDbDataAdapter mới này chứa danh sách các phân loại. Bao gồm các bước sau:

1. Lựa Data|Generate Dataset. Hộp thoại Generate Dataset xuất hiện.

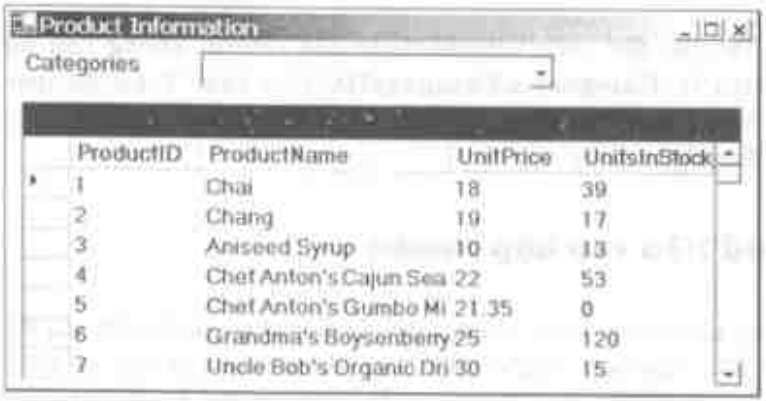
2. Phải chắc chắn Categories (OleDbDataAdapter2) được lựa trong hộp danh sách các bảng.

3. Nhấp New và nhập tên **dsCat**. Phải chắc chắn là bạn lựa tùy chọn Add This Dataset to the Designer. Nhấp OK. Điều này phát sinh dataset.

Bạn đã thêm một đối tượng mới vào Tray có tên dsCat1 và một file XSD mới vào Solution Explorer có tên dsCat.xsd. Hai mục này kết hợp tạo lớp dataset và được dùng để liên kết dữ liệu trên hộp combo bạn sắp tạo.

Thêm hộp combo vào form

Bây giờ bạn sẵn sàng thêm hộp combo vào form. Bạn phải tăng chiều cao của form và di chuyển control khung lưới dữ liệu xuống để tạo khoảng trống cho hộp combo mới này. Sau khi thực hiện điều này, bạn có thể kéo và thả hộp combo từ tab Windows Forms của Toolbox vào form. Đặt nó ở trên control khung lưới dữ liệu, như được trình bày ở hình 9.



Hình 9: *Form chứa control khung lưới dữ liệu và hộp combo.*

Liên kết hộp combo với DataAdapter

Hộp combo bạn đã thêm vào chưa liên kết với bất kỳ dữ liệu nào. Để liên kết hộp combo với dataset dsCat, thực hiện các bước sau:

1. Lựa hộp combo.

2. Đổi tên thành **cboCategories**.

3. Xác lập đặc tính DataSource là dsCat1. Điều này chỉ định đây là nguồn dữ liệu để từ đó đặt control khung lưới dữ liệu này.

4. Đổi đặc tính DropDownStyle thành **DropDownList**.

5. Xác lập đặc tính DisplayMember là **Categories.CategoryName**. Bạn sẽ phải nhấp danh sách thả xuống trong đặc tính DisplayMember và sau đó mở rộng Categories để hiển thị danh sách các cột bạn có thể sử dụng. Bước này chỉ định cột trong nguồn dữ liệu được dùng để đặt control khung lưới dữ liệu.

6. Xác lập đặc tính ValueMember của control khung lưới dữ liệu là **Categories.CategoryID**. Đặc tính ValueMember được dùng để chỉ định giá trị thực sử dụng khi một mục được lựa.

Đưa dữ liệu vào hộp combo

Giống như với control khung lưới dữ liệu, bạn phải viết mã để đưa dữ liệu vào hộp combo. Mã được thêm vào thủ tục sự kiện Load ngay dưới mã bạn đã dùng để nạp control khung lưới dữ liệu.

```
Private Sub frmProducts_Load( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles MyBase.Load

    ' Load the data grid control
    dsProducts1.Clear()
    OleDbDataAdapter1.Fill(dsProducts1, "Products")
```

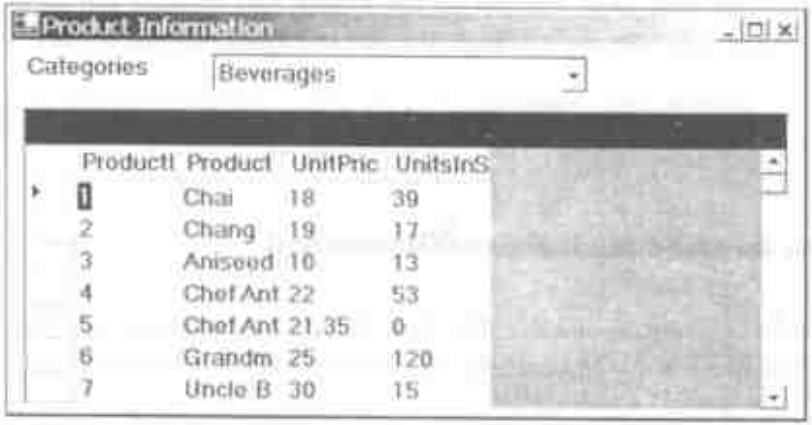
```
    ' Load the Combo Box
    dsCat1.Clear()
    OleDbDataAdapter2.Fill(dsCat1, "Categories")

End Sub
```

Mã được trình bày bằng nét đậm ở trên là mã bạn sẽ thêm để nạp hộp combo. Nó rất giống mã bạn đã dùng để nạp control khung lưới dữ liệu. Bây giờ bạn sẽ nạp control DataAdapter thứ hai bạn đã tạo.

Chúng ta hãy chạy mã này để xem các phân loại có được nạp hay không.

1. Ấn **F5** để chạy form này.

2. Thả hộp combo Categories xuống để xem các phân loại được nạp vào danh sách hay không. Nếu bạn đã thực hiện mọi thứ chính xác, chúng sẽ ở đó và màn hình của bạn có thể giống như hình 10.



Hình 10: *Việc sử dụng hộp combo thả xuống để hiển thị ít dữ liệu hơn trong khung lưới dữ liệu có thể tăng hiệu suất.*

Tạo truy vấn tham số hóa

Bây giờ bạn đã nạp hộp combo với các phân loại, giờ là lúc dùng phân loại được lựa từ hộp combo để lựa các sản phẩm nạp vào control khung lưới dữ liệu. Các bước cơ bản như sau:

1. Xóa mã cư trú control khung lưới dữ liệu khi form được nạp.
2. Sửa đổi đặc tính SelectCommand của đối tượng OleDbDataAdapter1 để nó nhận tham số cho Category ID có dữ liệu bạn muốn hiển thị.
3. Thêm mã thi hành khi người dùng lựa một mục trong hộp combo.

Xóa mã cư trú control khi form được nạp

Bây giờ bạn phải xóa mã cư trú khung lưới dữ liệu khi form được nạp. Hãy xóa mã sau khỏi thủ tục sự kiện Load của form:

```
dsProducts1.Clear()
OleDbDataAdapter1.Fill(dsProducts1, "Products")
```

Sửa đổi đặc tính SelectCommand

Bước kế tiếp là sửa đổi đặc tính SelectCommand của adapter dữ liệu để nó dựa vào truy vấn đã tham số hóa. Các bước như sau:

1. Nhấp đối tượng OleDbDataAdapter1 trong Tray của form.
2. Ấn **F4** để gọi ra cửa sổ Properties.

3. Nhấp dấu cộng bên đặc tính SelectCommand để mở rộng danh sách các phân loại phụ.

4. Nhấp đặc tính CommandText. Nhấp Build (...) để hiển thị hộp thoại Query Builder. (Xem hình 11.)

5. Thêm mệnh đề WHERE giống câu lệnh SQL sau:

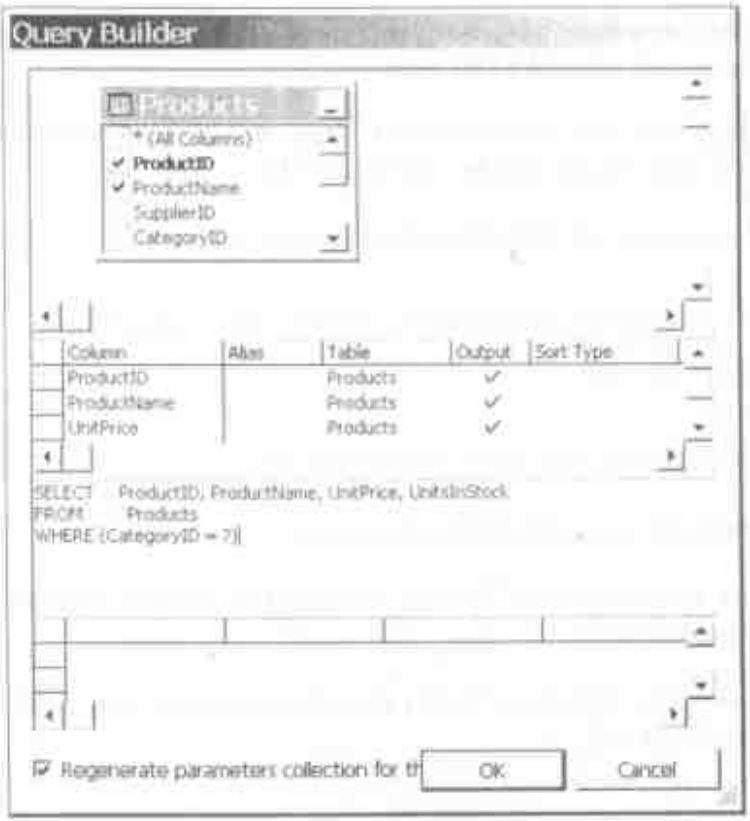
```
SELECT ProductID, ProductName, UnitPrice, UnitsInStock
FROM Products
WHERE CategoryID = ?
```

6. Nhấp OK để chấp nhận sự thay đổi này.

7. Nhấp đối tượng OleDbDataAdapter1.

8. Lựa Data | Generate Dataset và nhấp OK để phát sinh lại dataset sẵn có.

9. Phải chắc chắn bạn đã lựa dataset dsProducts sẵn có để phát sinh lại.



Hình 11: *Hộp thoại Query Builder giúp bạn xây dựng các câu lệnh SQL cho các adapter dữ liệu.*

Thêm mã thi hành khi người dùng lựa một mục trong hộp combo

Đoạn mã cần thiết cuối cùng cư trú lại control khung lưới dữ liệu khi một phân loại được lựa trong hộp combo cboCategories. Các bước cơ bản để thực hiện điều này như sau:

1. Trả lời sự kiện SelectedIndexChanged.

2. Xác lập tham số trong dataset với dữ liệu được truy tìm từ mục đã lựa trong hộp combo.

3. Điền dataset bằng cách dùng tham số này.

Thực hiện các bước sau để thêm mã và tạo cho nó hoạt động.

1. Mở form ở chế độ thiết kế.

2. Nhấp đôi hộp combo cboCategories để hiển thị thủ tục sự kiện SelectedIndexChanged. Đây là sự kiện kích hoạt khi bạn chọn một mục mới từ hộp combo.

3. Viết mã sau vào trong thủ tục này.

```
Private Sub cboCategories_SelectedIndexChanged( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles cboCategories.SelectedIndexChanged
    With OleDbDataAdapter1.SelectCommand.Parameters
        .Item(0).Value = cboCategories.SelectedValue
    End With

    ' Clear the dataset
    DsProducts1.Clear()
    ' Load the dataset using the parameter value
    OleDbDataAdapter1.Fill(DsProducts1, "Products")
End Sub
```

Bạn phải dùng đối tượng SelectCommand của adapter dữ liệu để truy tìm đối tượng Parameter đã chỉ định. Sau đó bạn có thể xác lập đặc tính Value của tham số này là đặc tính SelectedValue của hộp combo. Đặc tính SelectedValue được điền vào với

Category ID vì bạn xác lập ValueMember của hộp combo sử dụng trường CategoryID từ dataset. Một khi bạn điền giá trị này, bạn có thể điền vào dataset Products và nó sẽ tự động cư trú lại control khung lưới dữ liệu với chính các sản phẩm cho phân loại đã lựa đó.

Ở điểm này, bạn có thể chạy ứng dụng lại để xem nó hoạt động như thế nào.

1. Ấn **F5** để chạy ứng dụng.

2. Lựa phân loại từ hộp combo và bạn sẽ thấy một danh sách các sản phẩm nhỏ hơn được hiển thị trong control khung lưới dữ liệu.

3. Lựa một phân loại khác để xem tập các sản phẩm khác.

Liên kết dữ liệu thủ công

Bạn không phải dùng wizard để tạo lớp DataSet bạn dùng để nạp khung lưới dữ liệu hoặc một control nhận biết dữ liệu khác. Thay vào đó, bạn có thể viết một vài dòng mã mới để nạp dữ liệu vào hộp combo hoặc khung lưới dữ liệu. Bây giờ bạn sẽ tạo một project khác với một form frmProducts khác giống form bạn đã tạo trước đây ở chương này. Nhưng bạn sẽ chỉ tạo giao diện và tự liên kết tất cả. Bạn sẽ không dùng bất kỳ các wizard nào để nạp dữ liệu vào control combo và khung lưới dữ liệu.

1. Tạo một project mới có tên **DataBindGrid**.

2. Tạo một form có tên **frmProducts**.

3. Thêm control Label, ComboBox và DataGrid vào form.

4. Đặt tên của chúng là Label1, cboCategories và grdProducts tương ứng.

5. Nhấp đôi vào form và thêm mã sau:

```
Private Sub frmProducts_Load( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles MyBase.Load
    Call ComboLoad()
End Sub
```

Trong thủ tục ComboLoad bạn sẽ cư trú hộp combo cboCategories với dữ liệu bạn tạo sử dụng lớp DataTable. Lớp DataTable có thể được nạp từ một đối tượng DataAdapter bạn tự tạo thay vì thông qua một wizard để tạo nó. Tạo thủ tục sau bên trong form frmProducts này.

```
Private Sub ComboLoad()
    Dim da As OleDb.OleDbDataAdapter
    Dim dt As DataTable = New DataTable()
    Dim strSQL As String
    Dim strConn As String

    strConn = "Provider=sqloledb;Data
     Source=(local);Initial       Catalog=Northwind;User
ID=sa"

    strSQL = "SELECT CategoryId, CategoryName  "
    strSQL &= "FROM Categories"

    Try
        da    =    New    OleDb.OleDbDataAdapter(strSQL,
strConn)

        da.Fill(dt)

        With cboCategories
            ' Set these properties first
            .DisplayMember = "CategoryName"
            .ValueMember = "CategoryId"
```

```
            ' Then set the DataSource
            .DataSource = dt
        End With

    Catch e As Exception
        MessageBox.Show(e.Message)

    End Try
End Sub
```

Lưu ý:

Mã trên màn hình không chính xác! Mã ở trên chính xác. Bạn phải xác lập .DisplayMember và .ValueMember trước khi xác lập DataSource, thì bạn sẽ không gặp vấn đề gì trong thủ tục GridLoad.

Điểm chủ yếu của mã ở trên là sẽ tạo đối tượng OleDbDataAdapter và chuyển cho nó chuỗi SQL và chuỗi Connection. DataAdapter này sẽ đảm trách việc kết nối với CSDL và điền toàn bộ lớp DataTable bằng dữ liệu.

Một khi DataTable được điền dữ liệu, bạn phải xác lập đặc tính DisplayMember và ValueMember với các tên cột bạn đã dùng trong câu lệnh SQL. Đặc tính DisplayMember được dùng để hiển thị dữ liệu từ cột này bên trong phần thấy được của hộp combo. Đặc tính ValueMember lưu trữ một giá trị khác xuất phát từ cột CategoryId ở một danh sách ẩn trong hộp combo. Bạn sẽ dùng giá trị ấy từ đặc tính này sau đó trong mã này.

Điền DataGrid

Một khi người dùng lựa một mục từ control hộp combo, thủ tục sự kiện SelectedIndexChanged được kích hoạt. Sau là mã bạn cần viết cho thủ tục sự kiện này.

```
Private Sub cboCategories_SelectedIndexChanged( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) _
 Handles cboCategories.SelectedIndexChanged
    Call GridLoad()
End Sub
```

Thủ tục này chỉ gọi một thủ tục có tên GridLoad để kéo dữ liệu đã lựa từ hộp combo và nạp khung lưới với các sản phẩm cho phân loại được lựa. Bạn sẽ thấy khi khảo sát thủ tục GridLoad mà bạn xây dựng mệnh đề WHERE trên SELECT sử dụng đặc tính cboCategories.SelectedValue. Đây là đặc tính được điền với Category Ids vì bạn xác lập đặc tính ValueMember trên hộp combo.

Lưu ý:

Mã trên màn hình không chính xác! Nếu bạn xác lập .DisplayMember và .ValueMember trước khi xác lập DataSource, thì bạn không gặp vấn đề gì trong thủ tục GridLoad. Mã sau chính xác.

```
Private Sub GridLoad()
    Dim da As OleDb.OleDbDataAdapter
    Dim dt As DataTable = New DataTable()
    Dim strSQL As String
    Dim strConn As String

    strConn = "Provider=sqloledb;Data
     Source=(local);Initial       Catalog=Northwind;User
ID=sa"

    strSQL = "SELECT ProductID, ProductName, "
    strSQL &= "UnitPrice, UnitsInStock "
    strSQL &= "FROM Products "
    strSQL &= "WHERE CategoryID = " & _
     cboCategories.SelectedValue

    Try
```

```
        da    =    New    OleDb.OleDbDataAdapter(strSQL,
strConn)

        da.Fill(dt)

        grdProducts.PreferredColumnWidth = _
         DataGrid.AutoColumnSize
        grdProducts.DataSource = dt

    Catch e As Exception
        MessageBox.Show(e.Message)

    End Try
End Sub
```

Một khi câu lệnh được xây dựng, bạn xây dựng đối tượng DataAdapter, gọi ra phương thức Fill trên đối tượng này để điền DataTable. Sau đó bạn có thể xác lập đặc tính DataSource trên control DataGrid là DataTable mới này và nó sẽ hiển thị các cột dữ liệu từ các bản ghi đã truy tìm.

Những gì khác với VB6

Việc liên kết dữ liệu bằng ADO.NET mạnh hơn liên kết dữ liệu trong VB 6.0 rất nhiều. Với VB 6.0, bạn ít điều khiển được cách dữ liệu được liên kết hoặc những gì sẽ tiến hành bên dưới các vỏ bọc. Sử dụng VB 6.0 khi bạn dùng các form đã liên kết, bạn thêm control dữ liệu và các control nhập dữ liệu vào form. Về cơ bản bạn bị ràng buộc vào tính năng mà control dữ liệu cung cấp. Rất khó khắc phục các vấn đề hoặc sửa đổi ứng xử của control dữ liệu, vì Microsoft đã không cho bạn thấy mã nguồn bên đàng sau control.

Việc sử dụng liên kết dữ liệu với Windows Forms và ADO.NET, bạn điều khiển được cách dữ liệu được liên kết và cách ứng xử của form nhiều hơn. Liên kết dữ liệu sử dụng các lớp ADO.NET và phát sinh mã lớp mà bạn có thể xem và sửa đổi.

Điều này có nghĩa là khi những gì không làm việc hoặc không làm việc theo cách bạn muốn, thì bạn có thể can thiệp, chứ không như trong VB6. Liên kết dữ liệu bị giới hạn trong VB 6.0 không phải là sự chọn lựa tối ưu cho các ứng dụng sản xuất; liên kết dữ liệu với ADO.NET là một tùy chọn có thể tồn tại cho người phát triển .NET.

Tóm tắt

Liên kết dữ liệu có thể giúp bạn phát triển ứng dụng trong một thời gian rất ngắn. Bằng cách dùng liên kết dữ liệu, bạn không phải viết tất cả mã được đòi hỏi bởi các form chưa liên kết của VB 6.0, nhưng có được nhiều lợi ích của các form chưa liên kết. Mặc dù bạn đã tìm hiểu cách dùng các đối tượng riêng của SQL Server ở chương này, nhưng còn có các đối tượng tương đương cho phép bạn kết nối bất kỳ nguồn dữ liệu nào bằng cách dùng cùng các phương thức chính xác bạn vừa tìm hiểu. Thông thường, các điều tác này có thể được hoàn tất mà bạn chỉ phải viết rất ít mã.

Ở chương này chúng ta đã tìm hiểu:

- Các khái niệm về liên kết dữ liệu.
- Cách xây dựng các form liên kết dữ liệu.
- Cách làm việc với các hộp văn bản, control khung lưới dữ liệu và các hộp combo.
- Cách giới hạn dữ liệu hiển thị trong một form.
- Cách tạo các form nhập dữ liệu.

- Cách đi từ hàng này tới hàng khác.

Câu hỏi ôn tập

1. Simple Data Binding là gì?

2. Complex Data Binding là gì?

3. Bạn sẽ thực hiện kiểu liên kết nào khi liên kết với một hộp văn bản?

4. Bạn sẽ thực hiện kiểu liên kết nào khi liên kết với một DataGrid?

5. Nếu bạn đang dùng SQL Server, bạn sẽ dùng lớp SqlDataAdapter hay lớp OleDbDataAdapter?

6. Bạn xác lập đặc tính nào trên một Combo Box với một tên cột trước khi xác lập DataSource để hiển thị dữ liệu trong hộp combo ấy?

Bài tập

- Tạo Data Bound DataGrid để hiển thị thông tin khách hàng từ bảng Customers trong Northwind.

- Thêm hộp combo để chỉ lựa các khách hàng ở riêng một nước.

Trả lời câu hỏi ôn tập

1. Liên kết một cột dữ liệu với một đặc tính.
2. Liên kết nhiều cột dữ liệu với một control hỗ trợ nhiều cột, như một khung lưới.
3. Simple.
4. Complex.
5. SqlDataAdapter.
6. DisplayMember.

Chương 18

Đối tượng ADO.NET Connection và ADO.NET Command

- Tìm hiểu cách tạo đối tượng ADO.NET Connection.
- Tìm hiểu cách đệ trình SQL bằng đối tượng ADO.NET Command.

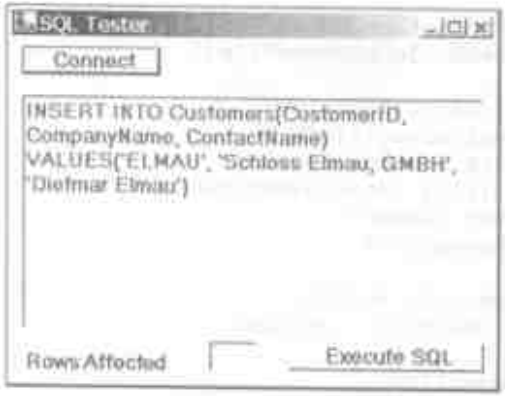
ADO.NET Connection

Bạn sử dụng đối tượng ADO.NET Connection để tạo kết nối CSDL giữa chương trình của bạn và engine CSDL. Thông thường bạn sẽ duy trì kết nối mở vừa đủ thời gian để truy tìm hoặc cập nhật dữ liệu. Bằng cách này, bạn sử dụng các tài nguyên máy chủ trong thời gian càng ít càng tốt. Điều này giúp bạn phát triển các ứng dụng nhanh và phù hợp là các tài nguyên thân thiện. Bạn càng dùng ít tài nguyên, bạn càng có thể hỗ trợ nhiều người dùng trên các ứng dụng của bạn cùng một lúc.

Nếu bạn đang tạo một ứng dụng CSDL, cuối cùng bạn phải mở một kết nối với CSDL đó. Đối tượng OleDbConnection sẽ cho phép bạn tạo kết nối đó. Bạn cũng phải truy tìm và sửa đổi dữ liệu trong CSDL ấy và đó là nơi bạn sẽ phải sử dụng đối tượng OleDbCommand.

Ở chương này, bạn sẽ tìm hiểu cách tạo và mở một kết nối với CSDL SQL Server, bằng cách dùng một lớp OleDbConnection. Ngoài ra, bạn sẽ tìm hiểu cách đệ trình câu lệnh INSERT tới cùng CSDL SQL Server bằng cách dùng đối tượng OleDbCommand.

Thực hiện các bước sau để tìm hiểu cách mở một kết nối với CSDL SQL Server. Hình 1 trình bày form mà bạn sẽ tìm thấy trong project dưới folder FirstStep cho chương này.



Hình 1: *Sử dụng màn hình này để đệ trình các câu lệnh SQL với CSDL SQL Server.*

Các bước để mở một kết nối

1. Mở project ConnectTest.sln được định vị trong folder FirstStep.

2. Mở form frmConnect.vb trong khung xem thiết kế.

3. Nhấp đôi nút Connect.

4. Thêm mã bên dưới vào thủ tục sự kiện btnConnect_Click.

```
Private Sub btnConnect_Click( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles btnConnect.Click
    Dim oConn As OleDb.OleDbConnection
    Dim strConn As String

    Try
        ' Create the Connection object
        oConn = New OleDb.OleDbConnection()

        ' Build the connection string
        strConn = "Provider=sqloledb;"
        strConn &= "Data Source=(local);"
        strConn &= "Initial Catalog=Northwind;"
        strConn &= "User ID=sa;"
        strConn &= "Password=;"

        ' Set the Connection String
        oConn.ConnectionString = strConn

        ' Open the Connection
        oConn.Open()

        MessageBox.Show("Connection Open", _
          "btnConnect_Click()")

        ' Close the Connection
        oConn.Close()

    Catch oExcept As Exception
        MessageBox.Show(oExcept.Message, _
```

```
            "btnConnect_Click()")

        End Try

    End Sub
```

Trong thủ tục sự kiện này, đầu tiên bạn tạo một thể nghiệm mới của lớp OleDbConnection. Sau đó bạn sẽ điền vào ít nhất đặc tính ConnectionString trước khi mở kết nối.

Các chuỗi nhà cung cấp OLE DB

Kế tiếp bạn tạo một chuỗi nhà cung cấp OLE DB để trỏ tới một CSDL SQL Server. Một chuỗi nhà cung cấp có một tập các thuộc tính được tách rời bằng dấu chấm phẩy. Mỗi chuỗi nhà cung cấp sẽ trông khác nhau tùy theo kiểu nhà cung cấp bạn cần sử dụng và các thuộc tính nào được xác lập cho mỗi kiểu hệ thống CSDL khác nhau. Thí dụ, nhà cung cấp bên dưới là một thí dụ về những gì bạn sẽ dùng để kết nối với SQL Server cục bộ.

```
Provider=sqloledb;Data Source=(local);Initial
Catalog=Northwind;User ID=sa;Password=;
```

Chuỗi nhà cung cấp bên dưới là một thí dụ về cách bạn sẽ kết nối với CSDL Access 2000.

```
Provider=Microsoft.Jet.OleDb.4.0;Data
Source=C:\Northwind.mdb
```

Lưu ý:

Nếu bạn đang sử dụng một engine CSDL khác, bạn sẽ phải tìm các thuộc tính thích hợp để xác lập cho engine đặc biệt của bạn. Tham khảo trợ giúp của ADO.NET về việc xác lập chuỗi kết nối cho các nhà cung cấp OLE DB khác.

Bạn sẽ tạo chuỗi kết nối này bằng biến chuỗi *strConn*, sau đó gán biến đó với đặc tính ConnectionString của đối tượng OleDbConnection.

Sau khi đặc tính ConnectionString được xác lập, bạn gọi ra phương thức Open trên đối tượng OleDbConnection. Điều này khiến cho đối tượng kết nối nạp nhà cung cấp đã chỉ định và mở kết nối với nguồn dữ liệu. Nếu được cung cấp, thuộc tính User ID và Password được dùng để đăng nhập vào nguồn dữ liệu. Thuộc tính Initial Catalog cho kết nối biết tạo CSDL mặc định mà nó được chỉ định bằng thuộc tính này.

Sau khi bạn kết thúc kết nối, bạn luôn luôn phải đóng nó. Trong mã thí dụ bạn vừa gõ vào chỉ để mở một kết nối và sau đó đóng nó lại ngay. Thường bạn sẽ thực hiện một thao tác nào đó trên kết nối ấy trước khi đóng nó. Điều này sẽ được trình bày ở phần kế tiếp.

1. Ấn **F5** để chạy project.

2. Nhấp nút Connect để chạy mã bạn vừa gõ vào.

3. Nếu mọi thứ được xác lập chính xác, bạn sẽ thấy một hộp thông báo cho bạn biết kết nối ấy đã được mở.

Bây giờ bạn đã biết cách mở một kết nối, bạn nên làm điều gì đó với nó. Không giống như các phiên bản ADO trước đây, ADO.NET không cho phép bất kỳ sự thi hành câu lệnh hoạt động SQL nào (INSERT, UPDATE và DELETE) trên một đối tượng kết nối. Để đệ trình các câu lệnh hoạt động SQL, bạn sẽ dùng đối tượng OleDbCommand.

Đối tượng ADO.NET Command

Đối tượng OleDbCommand rất giống đối tượng ADO command cũ. Nó được dùng để lưu trữ các câu lệnh SQL cần được thi hành dựa vào nguồn dữ liệu. Đối tượng OleDbCommand có thể thi hành các câu lệnh SELECT hoặc INSERT, UPDATE hay các câu lệnh DELETE. Ở hình 1, bạn thấy một thí dụ về câu lệnh INSERT mà ban có thể thi hành dựa vào CSDL Northwind để thêm một dòng vào bảng Customers. Bây giờ bạn sẽ tìm hiểu cách viết mã sẽ thi hành câu lệnh INSERT.

Các bước thi hành câu lệnh INSERT

1. Mở form frmConnect.vb.
2. Nhấp đôi nút Execute SQL.
3. Viết mã bên dưới vào thủ tục sự kiện btnExecute_Click.

```
Private Sub btnExecute_Click(ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles btnExecute.Click
    Dim oCmd As OleDb.OleDbCommand
    Dim strConn As String

    Try
        ' Build the connection string
        strConn = "Provider=sqloledb;"
        strConn &= "Data Source=(local);"
        strConn &= "Initial Catalog=Northwind;"
        strConn &= "User ID=sa;"
        strConn &= "Password=;"

        ' Create the Command Object
        oCmd = New OleDb.OleDbCommand()
        ' Assign Connection to Command Object
        oCmd.Connection = _
         New OleDb.OleDbConnection(strConn)
        ' Open the Connection
```

```
            oCmd.Connection.Open()
            ' Assign the SQL to the Command Object
            oCmd.CommandText = txtSQL.Text
            ' Execute the SQL,
            ' Return Number of Records Affected
            txtRows.Text = _
             oCmd.ExecuteNonQuery().ToString()

            MessageBox.Show("SQL statement succeeded", _
             "btnExecute_Click()")

            ' Close the Connection
            oCmd.Connection.Close()

        Catch oExcept As Exception
            txtRows.Text = 0.ToString()
            MessageBox.Show("Error executing SQL: " & _
             oExcept.Message, "btnExecute_Click()")

        End Try
    End Sub
```

Trong thủ tục sự kiện btnExecute_Click bạn cần khai báo biến có tên *oCmd*, nó sẽ tham chiếu đối tượng OleDbCommand. Sau đó bạn tạo chuỗi kết nối OLE DB sẽ được dùng để tạo kết nối với nguồn dữ liệu. Kế tiếp bạn lập thể nghiệm đối tượng OleDbCommand và xác lập đặc tính Connection của nó là đối tượng New OleDbConnection. Một khi bạn đã tạo đối tượng OleDbConnection này, bạn có thể mở kết nối đó.

Sau khi kết nối mở, bạn có thể điền vào đặc tính CommandText với câu lệnh SQL bạn muốn đệ trình với CSDL. Trong trường hợp này, câu lệnh SQL đến từ Text Box trên form này.

Bạn gọi ra phương thức ExecuteNonQuery của đối tượng command để đệ trình SQL với CSDL chương trình phụ. Phương thức ExecuteNonQuery giả định bạn đã điền vào một truy vấn hành động bằng đặc tính CommandText. Truy vấn hành động là

một truy vấn không cho trở lại bất kỳ các dòng nào như tập kết quả.

Phương thức này sẽ cho trở lại số dòng được dùng bởi câu lệnh SQL đã đệ trình. Bạn chuyển đổi giá trị này thành một chuỗi để nó có thể được đặt vào đặc tính Text của hộp văn bản *txtRows*. Cuối cùng bạn đóng kết nối bằng cách dùng phương thức Close trên đặc tính Connection.

Lưu ý:

Bạn không phải đóng kết nối dứt khoát như bạn đã thực hiện trong thủ tục này. Khi đối tượng command ra khỏi phạm vi, đối tượng kết nối được chứa trong đặc tính Connection cũng sẽ ra khỏi phạm vi ấy. Một khi đối tượng OleDbConnection bị hủy bởi bộ thu thập dữ liệu không thích hợp (garbage), phương thức Close tự động được gọi. Vấn đề để nó tự động đóng sẽ làm cho bạn không biết chính xác khi nào đối tượng OleDbConnection sẽ bị hủy vì bạn phải chờ garbage collector (bộ thu thập rác) thực hiện công việc của nó. Việc đóng kết nối này dứt khoát sẽ giải phóng kết nối về phạm vi cho thủ tục nào đó sử dụng lại ngay khi có thể.

Bây giờ bạn có thể thử nút mới này bằng cách viết câu lệnh INSERT hoặc UPDATE hay DELETE dựa vào một trong các bảng ở CSDL Northwind và đệ trình với CSDL ấy.

1. Chạy chương trình bằng cách ấn **F5**.

2. Gõ vào câu lệnh INSERT, UPDATE hoặc DELETE hợp lệ. Bạn chỉ có thể sử dụng câu lệnh INSERT đã được đặt vào hộp văn bản như một thí dụ.

3. Nhắp nút Execute SQL.

Nếu mọi thứ hoạt động tốt, thì bạn sẽ thấy số dòng được dùng thể hiện trong hộp văn bản bạn đã tạo ở bên trái nút này.

Ở các phiên bản ADO trước đây, bạn được phép xác lập đặc tính ConnectionTimeout là một giá trị từ 0 tới *n*. Nó biểu thị thời gian chờ một kết nối xảy ra trước khi một ngoại lệ được đưa ra. Trong ADO.NET, đặc tính này là chỉ đọc. Để xác lập ConnectionTimeout bạn phải chuyển "Connect Timeout=n" vào chuỗi của nhà cung cấp. Ngoài ra, bạn không còn có thể xác lập các đặc tính riêng, chẳng hạn như Provider, DataSource, Database, vân vân như các đặc tính trên đối tượng OleDbConnection. Các giá trị này bây giờ chỉ là chỉ đọc và phản ảnh các giá trị được phân tách từ chuỗi nhà cung cấp.

Tóm tắt

Ở chương này bạn đã tìm hiểu cách tạo một kết nối với CSDL SQL Server bằng cách dùng đối tượng kết nối ADO.NET. Bạn cũng đã tìm hiểu cách đệ trình các truy vấn hành động bằng đối tượng ADO.NET OleDbCommand. Còn có nhiều thay đổi về các đặc tính và các phương thức trong các đối tượng này.

Câu hỏi ôn tập

1. Bạn sử dụng đối tượng ADO.NET Connection để làm gì?

2. Đúng hay sai: Bạn có thể đệ trình câu lệnh SQL UPDATE bằng đối tượng kết nối?

3. Bạn đặt SQL vào đặc tính nào trên đối tượng Command?

4. Bạn dùng phương thức nào để đệ trình truy vấn hành động SQL với CSDL chương trình phụ?

5. Đúng hay sai: Bạn nên đóng một kết nối dứt khoát ngay sau khi bạn đã thực hiện xong?

Bài tập

- Thực hiện các bài tập ở chương này.

Trả lời câu hỏi ôn tập

1. Mở kết nối với một CSDL.

2. Sai.

3. CommandText.

4. ExecuteNonQuery.

5. Đúng.

Chương 19

Đối tượng ADO.NET DataReader

- Tìm hiểu cách dùng lớp ADO.NET DataReader
- Tìm hiểu các cơ chế con trỏ khác nhau bạn có thể sử dụng
- Tìm hiểu cách nạp các đối tượng vào hộp danh sách

Sử dụng ADO.NET DataReader

Trong ADO.NET, bạn không còn có sẵn đối tượng Recordset để làm việc. Thay vào đó, bạn có các đối tượng mới, chẳng hạn như DataSets, DataTables và DataReaders sẽ được dùng để truy tìm các bản ghi từ các nguồn dữ liệu. Ở chương này bạn sẽ tìm hiểu về đối tượng DataReader.

Đối tượng DataReader là con trỏ kiểu chỉ tiếp tới. Bạn có thể chỉ định kiểu DataReader bạn muốn, chẳng hạn như SequentialAccess, KeyInfo, SchemaOnly, SingleResult và

SingleRow. Mỗi con trỏ này chỉ cho phép bạn di đi chuyển tới qua tập kết quả. Trong trường hợp của SequentialAccess và SingleRow, thậm chí bạn phải truy tìm các cột theo thứ tự bạn đã chỉ định chúng trong câu lệnh SELECT.

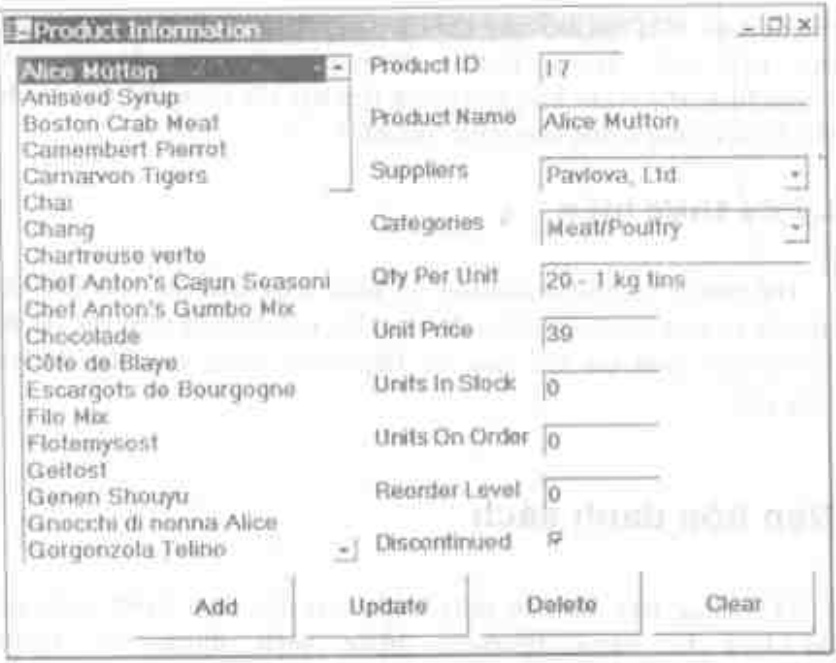
Lý do thực hiện

Đối tượng OleDbDataReader là cách truy tìm các bản ghi rất nhanh từ một nguồn dữ liệu. Nó là kiểu con trỏ chỉ tiếp tới, do đó nó có hiệu suất tốt khi nạp các hộp danh sách, các hộp combo, vân vân.

Nạp hộp danh sách

Ở chương này, bạn sẽ thấy cách nạp một hộp danh sách với dữ liệu từ bảng Products bằng cách dùng đối tượng OleDbDataReader. Ngoài ra, bạn sẽ thấy cách truy tìm một bản ghi đơn giản sau khi nhấp vào hộp danh sách.

Hình 1 trình bày một màn hình nhập dữ liệu đơn giản mà bạn có thể dùng để hiển thị, bổ sung, soạn thảo và xóa thông tin sản phẩm.



Hình 1: *Màn hình nhập dữ liệu Client/Server tiêu biểu.*

Các bước nạp hộp danh sách

Để xây dựng màn hình này, chúng ta hãy tiến hành từng bước. Đầu tiên bạn sẽ thực hiện các bước bên dưới để nạp hộp danh sách với các tên của tất cả các sản phẩm ở bảng Products trong CSDL Northwind.

1. Nạp DataReaderSample.sln được định vị ở folder \FirstStep.

2. Nhấp đôi file frmProducts.vb trong trình khảo sát solution. Nó sẽ hiển thị form Products.

3. Nhấp đôi vào nơi bất kỳ trên form (chắc chắn bạn không nhấp vào control).

4. Bây giờ bạn sẽ hiển thị thủ tục sự kiện frmProduct_Load cho form này.

5. Đưa ra lời gọi thủ tục ListLoad bạn sẽ viết ở bước kế tiếp. Thủ tục của bạn giống mã được trình bày bên dưới.

```
Private Sub frmProduct_Load( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles MyBase.Load
        ListLoad()
End Sub
```

6. Bây giờ tạo thủ tục ListLoad ngay dưới End Sub của thủ tục sự kiện frmProduct_Load.

7. Gõ mã sau vào thủ tục ListLoad này.

```
Private Sub ListLoad()
    Dim oCmd As OleDb.OleDbCommand
    Dim oDR As OleDb.OleDbDataReader
    Dim strSQL As String
    Dim strConn As String

    strConn = ConnectStringBuild()

    strSQL = "SELECT ProductName "
    strSQL &= "FROM Products"

    Try
        oCmd = New OleDb.OleDbCommand()
        With oCmd
            .Connection = _
             New OleDb.OleDbConnection(strConn)
            .Connection.Open()
            .CommandText = strSQL
            oDR = _
              .ExecuteReader( _
              CommandBehavior.SequentialAccess)
```

```
            End With

            lstProducts.Items.Clear()
            Do While oDR.Read()

    lstProducts.Items.Add(oDR.Item("ProductName"))
            Loop

        Catch oExcept As Exception
            MessageBox.Show(oExcept.Message)

        End Try
    End Sub
```

Mã ở trên khai báo hai đối tượng, một cho OleDbCommand và một cho OleDbDataReader. Đối tượng OleDbCommand bạn sẽ dùng để lưu giữ và thi hành câu lệnh SELECT gửi tới nguồn dữ liệu. OleDbDataReader là đối tượng sẽ truy tìm dữ liệu từ tập kết quả trở lại từ câu lệnh SELECT.

Vào đối tượng OleDbCommand bạn xác lập đặc tính Connection là một đối tượng OleDbCommand mới. Bạn chuyển chuỗi kết nối tới đối tượng OleDbCommand này. Bạn sẽ xây dựng hàm ConnectStringBuild sau ở phần này. Sau khi chuỗi kết nối thích hợp được xác lập, bạn có thể mở kết nối trên đối tượng lệnh.

Bây giờ bạn đặt câu lệnh SELECT vào đặc tính CommandText. Khi bạn gọi ra phương thức ExecuteReader, đối tượng lệnh sẽ đệ trình câu lệnh SELECT với nguồn dữ liệu chương trình phụ. Kết quả được cho trở lại và đối tượng DataReader là một con trỏ chỉ định tới tập kết quả này. Con trỏ được xác lập ngay trước bản ghi đầu tiên trong tập kết quả này.

Bây giờ bạn sẽ lặp qua từng dòng trong DataReader bằng cách gọi ra phương thức Read. Phương thức Read sẽ di chuyển con trỏ từ dòng này tới dòng khác. Sau khi phương thức Read thi hành xong bạn có thể chuyển tên cột bạn muốn để truy tìm đặc tính

Item trên DataReader. Nó sẽ cho trở lại dữ liệu thực từ cột đó. Vì bạn đã chỉ định SequentialAccess trong phương thức ExecuteReader, bạn phải đọc các cột theo cùng thứ tự đã chỉ định trong câu lệnh SELECT.

Dữ liệu được trả lại từ đặc tính Item sẽ cho trở lại như kiểu dữ liệu Object. Vì các mục bạn thêm vào hộp danh sách là kiểu Object không cần chuyển đổi khi sử dụng phương thức Add của tập hợp Items trên hộp danh sách.

Bằng cách này bạn tiếp tục vòng lặp cho tới khi phương thức Read cho trở lại False. Điều này có nghĩa là bạn tới cuối các dòng dữ liệu được cho trở lại từ câu lệnh SELECT.

Trước khi bạn có thể thử thủ tục này để xem dữ liệu có được nạp chính xác vào hộp danh sách hay không, bạn phải xây dựng chuỗi kết nối.

Ngay sau End Sub của thủ tục ListLoad, thêm thủ tục mới có tên ConnectStringBuild.

1. Gõ mã bên dưới vào để tạo chuỗi kết nối này. Bạn nên thay đổi chuỗi kết nối nếu đang sử dụng một tên máy chủ khác hoặc một engine CSDL khác.

```
Private Function ConnectStringBuild() As String
    Dim strConn As String

    strConn = "Provider=sqloledb;"
    strConn &= "Data Source=(local);"
    strConn &= "Initial Catalog=Northwind;"
    strConn &= "User ID=sa"

    Return strConn
End Function
```

2. Ấn **F5** để chạy ứng dụng.

Nếu bạn đã gõ mọi thứ chính xác, bạn sẽ thấy một danh sách các sản phẩm trong hộp danh sách.

Vấn đề với mã trên

Một trong các vấn đề với mã ở trên là bạn có thể sao chép các tên sản phẩm trong bảng của bạn. Nếu ở trường hợp này, không có cơ chế nào cho bạn lưu trữ primary key (khóa chính) vào danh sách này để duy nhất bạn có thể nhận dạng bản ghi nào là bản ghi trong danh sách. Không giống VB6, ListBox trong .NET không có đặc tính ItemData. ItemData được dùng để lưu trữ kiểu dữ liệu long integer được liên kết với văn bản trong hộp văn bản. .NET đã loại bỏ đặc tính ItemData này. Tuy nhiên, còn có phương thức xử lý kiểu tình huống này tốt hơn nhiều.

Lớp Generic ListItem

Điểm chủ yếu để thực hiện công việc này là tạo một lớp có đủ các đặc tính để lưu giữ dữ liệu bạn muốn đưa vào hộp danh sách. Đương nhiên, một đặc tính của lớp này sẽ được dùng để hiển thị dữ liệu trong phần văn bản của hộp danh sách. Đặc tính (hoặc các đặc tính) khác sẽ được dùng để lưu giữ thông tin primary key. Đối với bảng Product, bạn chỉ cần một đặc tính cho cột ProductName và một đặc tính cho cột ProductID. Thực ra, đối với nhiều bảng bạn sẽ chỉ cần hai đặc tính như ở đây. Do đó, bạn sẽ xây dựng một lớp generic với hai đặc tính được gọi là Value và ID mà bạn có thể dùng lại trong bất kỳ tình huống nào như lớp này.

Các bước tạo lớp Generic ListItem

Bây giờ bạn sẽ thêm một file mới vào project của bạn. File mới này sẽ chứa định nghĩa lớp cho lớp mục danh sách generic này.

- Lựa Project | Add Class... để thêm một lớp mới vào project.
- Cho nó một tên, chẳng hạn như clsListItems.vb.
- Tạo lớp như được trình bày ở mã sau.

```
Public Class PDSAListItemNumeric
    Private mstrValue As String
    Private mintID As Integer

    Public Sub New()

    End Sub

    Public Sub New(ByVal strValue As String, _
     ByVal intID As Integer)
        mstrValue = strValue
        mintID = intID
    End Sub

    Property Value() As String
        Get
            Return mstrValue
        End Get
        Set(ByVal Value As String)
            mstrValue = Value
        End Set
    End Property

    Property ID() As Integer
        Get
            Return mintID
        End Get
        Set(ByVal Value As Integer)
            mintID = Value
        End Set
```

```
    End Property

    Public Overrides Function ToString() As String
        Return mstrValue
    End Function
End Class
```

Lớp trên có hai đặc tính, Value và ID, sẽ được dùng để lưu giữ văn bản và primary key cho bất kỳ bảng nào bạn sẽ đặt vào hộp danh sách. Đối với một lớp bất kỳ bạn muốn đặt vào tập hợp Items trên control danh sách, bạn phải ghi đè phương thức ToString. ToString là phương thức theo kiểu dữ liệu Object mặc định mà tất cả các lớp tự động thừa kế nó. Khi hộp danh sách hiển thị các mục trong tập hợp Items của nó, nó sẽ luôn luôn gọi phương thức ToString để truy tìm dữ liệu. Trong phương thức ToString của lớp này bạn sẽ cho trở lại đặc tính Value vì nó là giá trị bạn muốn hiển thị trong hộp danh sách.

Sử dụng lớp mới này trong ListLoad

Bây giờ bạn đã tạo lớp mới này, chúng ta hãy dùng nó trong thủ tục ListLoad. Thay vì chỉ thêm ProductName trực tiếp từ OleDbDataReader, bạn phải thêm ProductName và ProductID vào một thể nghiệm mới của lớp PDSAListItemNumeric và sau đó thêm đối tượng này vào hộp danh sách.

Thay đổi mã trong thủ tục ListLoad bạn đã tạo trước đây bằng mã như ở bên dưới.

```
Private Sub ListLoad()
    Dim oCmd As OleDb.OleDbCommand
    Dim oDR As OleDb.OleDbDataReader
    Dim oItem As PDSAListItemNumeric
    Dim strSQL As String
    Dim strConn As String

    strConn = ConnectStringBuild()
```

```
    strSQL = "SELECT ProductID, ProductName "
    strSQL &= "FROM Products"

    Try
        oCmd = New OleDb.OleDbCommand()
        With oCmd
            .Connection = _
             New OleDb.OleDbConnection(strConn)
            .Connection.Open()
            .CommandText = strSQL
            oDR = _
             .ExecuteReader( _
             CommandBehavior.SequentialAccess)
        End With

        lstProducts.Items.Clear()
        Do While oDR.Read()
            oItem = New PDSAListItemNumeric()
            With oDR
                oItem.ID = CInt(.Item("ProductID"))
                oItem.Value = _
                 .Item("ProductName").ToString()
            End With
            lstProducts.Items.Add(oItem)
        Loop
        If lstProducts.Items.Count > 0 Then
            lstProducts.SetSelected(0, True)
        End If

    Catch oExcept As Exception
        MessageBox.Show(oExcept.Message)
    End Try
End Sub
```

Sự thay đổi chính của bạn đối với thủ tục này là bạn đã thêm cột ProductID vào câu lệnh SELECT, sau đó thêm thuyết minh của đối tượng oItem mới bên trong vòng lặp Read. Mỗi lần bạn đọc một bản ghi mới, bạn tạo một đối tượng PDSAListItemNumeric mới, lưu trữ ProductID vào đặc tính ID và ProductName vào đặc tính Value. Sau đó bạn thêm đối tượng mới này vào hộp danh sách.

Bây giờ bạn đã thay đổi mã này để sử dụng lớp PDSAListItemNumeric, bạn nên chạy project để chắc chắn bạn đã gõ mọi thứ chính xác.

- Ấn **F5** để xem bạn còn nạp Products vào hộp danh sách hay không.

Hiển thị thông tin sản phẩm

Một khi bạn đã đặt các đối tượng này vào hộp danh sách, khi bạn nhấp một mục trong hộp danh sách, bạn sẽ phải truy tìm đối tượng đó để có thể nhận đặc tính ID và Value. Khi bạn nhấp một mục trong hộp danh sách, thủ tục sự kiện SelectedIndexChanged sẽ kích hoạt. Bên trong sự kiện này, bạn có thể viết mã để gọi một thủ tục con khác gọi là FormShow.

- Đưa form Products vào chế độ xem thiết kế.
- Nhấp đôi hộp danh sách để hiển thị thủ tục sự kiện SelectedIndexChanged.
- Thêm một lời gọi thủ tục có tên FormShow.

```
Private Sub lstProducts_SelectedIndexChanged( _
 ByVal sender As Object, ByVal e As System.EventArgs) _
 Handles lstProducts.SelectedIndexChanged
    FormShow()
End Sub
```

Bây giờ tạo thủ tục FormShow này sau câu lệnh End Sub của thủ tục sự kiện trên. FormShow sẽ truy tìm đối tượng PDSAListItemNumeric từ hộp danh sách và xây dựng câu lệnh SELECT để truy tìm tất cả các cột từ bảng Products cho mục đặc biệt được lựa. Sau đó nó sẽ xây dựng đối tượng DataReader của một dòng, đọc dữ liệu từ nguồn dữ liệu, rồi đặt tất cả thông tin

chi tiết cho sản phẩm đó vào các hộp văn bản thích hợp trên form này.

- Tạo thủ tục FormShow bên trong form này.
- Gõ mã bên dưới vào.

```
Private Sub FormShow()
    Dim oCmd As OleDb.OleDbCommand
    Dim oDR As OleDb.OleDbDataReader
    Dim oItem As PDSAListItemNumeric
    Dim strSQL As String
    Dim strConn As String

    strConn = ConnectStringBuild()

    ' Get Primary Key From List Box
    oItem = CType(lstProducts.SelectedItem, _
     PDSAListItemNumeric)

    strSQL = "SELECT ProductID, ProductName, "
    strSQL &= " QuantityPerUnit, UnitPrice, "
    strSQL &= " UnitsInStock, UnitsOnOrder, "
    strSQL &= " ReorderLevel, Discontinued "
    strSQL &= " FROM Products "
    strSQL &= " WHERE ProductID = " & oItem.ID

    Try
        oCmd = New OleDb.OleDbCommand()
        With oCmd
            .Connection = _
             New OleDb.OleDbConnection(strConn)
            .Connection.Open()
            .CommandText = strSQL
            oDR = .ExecuteReader( _
             CommandBehavior.SequentialAccess)
        End With

        If oDR.Read() Then
            With oDR
                txtID.Text
.Item("ProductID").ToString()
                txtName.Text = _
```

```
                .Item("ProductName").ToString()
               txtQty.Text = _
                .Item("QuantityPerUnit").ToString()
               txtPrice.Text = _
                .Item("UnitPrice").ToString()
               txtInStock.Text = _
                .Item("UnitsInStock").ToString()
               txtOnOrder.Text = _
                .Item("UnitsOnOrder").ToString()
               txtReorder.Text = _
                .Item("ReorderLevel").ToString()
               chkDisc.Checked = _
                CType(.Item("Discontinued"), Boolean)
           End With
       End If
       oDR.Close()
       oCmd.Connection.Close()

   Catch oException As Exception
       MessageBox.Show(oException.Message)

   End Try
End Sub
```

Không có gì quá mới mẻ trong thủ tục FormShow. Đầu tiên bạn sẽ truy tìm đối tượng PDSAListItemNumeric từ đặc tính SelectedItem của hộp danh sách và đặt nó vào biến oItem.

```
oItem = CType(lstProducts.SelectedItem, _
 PDSAListItemNumeric)
```

Hàm CType chuyển đổi từ kiểu dữ liệu này sang một kiểu dữ liệu khác. Trong trường hợp này, bạn sẽ chuyển đổi kiểu Object thành kiểu PDSAListItemNumeric. Tất cả các mục được đặt vào tập hợp Items của hộp danh sách được lưu trữ như các kiểu Object generic.

Phần thủ tục còn lại chỉ tạo đối tượng OleDbCommand, mở kết nối và tạo đối tượng OleDbDataReader để đọc một bản ghi từ nguồn dữ liệu. Sau đó nó sử dụng từng cột và đặt dữ liệu vào các hộp văn bản thích hợp.

Bây giờ bạn đã tạo thủ tục FormShow, chạy thử để chắc chắn nó hoạt động.

1. Khởi động ứng dụng bằng cách ấn **F5**.

2. Nhấp một mục nhập trong hộp danh sách để chắc chắn thông tin chi tiết cho sản phẩm đặc biệt đó thể hiện trong các hộp văn bản trên form.

3. Đừng lo ngại về các hộp combo cho Supplier và Category, bạn sẽ tìm hiểu cách làm việc với các control đó ở phần kế tiếp.

Nạp các hộp combo

Trên form Product, có hai hộp combo cần được nạp, Categories và Suppliers. Cả hai đều có thể được nạp giống như hộp danh sách. Bạn sẽ sử dụng lớp PDSAListItemNumeric để nạp cả primary key lẫn văn bản cho hộp combo. Đối với bảng Categories bạn lựa cột CategoryID và CategoryName. Đối với bảng Suppliers bạn lựa cột SupplierID và CompanyName. Trong bảng Products, cột CategoryID và SupplierID là các foreign key đưa vào bảng Categories và Suppliers tương ứng.

- Tạo thủ tục CategoryLoad bên trong form này.

- Gõ mã bên dưới vào nơi nào đó trong cửa sổ mã của form.

```
Private Sub CategoryLoad()
    Dim oCmd As OleDb.OleDbCommand
    Dim oDR As OleDb.OleDbDataReader
    Dim strSQL As String
    Dim strConn As String
    Dim oItem As PDSAListItemNumeric
```

```
        strConn = ConnectStringBuild()

        strSQL = "SELECT CategoryID, CategoryName "
        strSQL &= "FROM Categories"

        Try
            oCmd = New OleDb.OleDbCommand()
            With oCmd
                .Connection = _
                 New OleDb.OleDbConnection(strConn)
                .Connection.Open()
                .CommandText = strSQL
                ' Closes connection when
                ' closing DataReader object
               oDR = .ExecuteReader( _
                CommandBehavior.CloseConnection)
            End With

            Do While oDR.Read()
                oItem = New PDSAListItemNumeric()
                With oDR
                    oItem.ID = CInt(.Item("CategoryID"))
                    oItem.Value = _
                      .Item("CategoryName").ToString()
                End With

                cboCategory.Items.Add(oItem)
            Loop
            oDR.Close()
            ' No need to close this because of the
            '.CloseConnection on the ExecuteReader
            'oCmd.Connection.Close()

        Catch oExcept As Exception
            MessageBox.Show(oExcept.Message)

        End Try
    End Sub
```

Có một sự khác biệt giữa thủ tục này và thủ tục ListLoad bạn đã tạo trước đây. Trên phương thức ExecuteReader bạn đã chuyển vào một hằng mới, CloseConnection. Tham số này sẽ cho lớp OleDbDataReader biết để đóng kết nối khi nó cần được đóng.

Điều này tránh cho bạn phải đưa ra một lời gọi phương thức Close cụ thể trên đặc tính Connection của đối tượng lệnh.

Tạo thủ tục SupplierLoad

Bây giờ bạn tạo thủ tục SupplierLoad giống như thủ tục trên. Thực ra, bạn có thể chép và dán thủ tục trên trở lại form và chỉ cần thay đổi tên.

- Chép thủ tục CategoryLoad và dán nó ngay dưới cùng thủ tục này.
- Đổi tên của thủ tục thành SupplierLoad.
- Đổi tên các cột và bảng thích hợp trong câu lệnh SELECT. Các cột sử dụng trong bảng Suppliers là SupplierID và CompanyName.
- Thay đổi tất cả các tham chiếu với hộp combo cboCategory trong thủ tục này để sử dụng hộp combo cboSupplier.

Gọi các thủ tục này khi nạp form

Bây giờ bạn phải thêm một lời gọi các thủ tục này từ thủ tục sự kiện frmProducts_Load.

- Thay đổi thủ tục sự kiện frmProduct_Load như sau.

```
Private Sub frmProduct_Load( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles MyBase.Load
    ' Load Suppliers
    SupplierLoad()
    ' Load Categories
    CategoryLoad()
```

```
        ' Load List Box of Products
        ListLoad()
    End Sub
```

Bây giờ chạy chương trình để chắc chắn các thủ tục này thực sự nạp dữ liệu vào các hộp combo thích hợp trên màn hình.

- Ấn **F5** để xem hộp combo Category và Supplier có được nạp hay không.

Tìm giá trị trong hộp combo

Bây giờ bạn đã nạp hộp combo Category và Supplier, bạn phải có thể định vị các hộp combo là các giá trị thích hợp bất kỳ khi nào bạn nhấp một sản phẩm mới trong hộp danh sách. Thí dụ, khi giá trị CategoryID trong bảng Products được đọc vào bằng đối tượng OleDbDataReader trong thủ tục FormShow, bạn phải tìm thấy CategoryID đó trong các đối tượng được nạp vào hộp combo Category. Còn một vài bước bạn cần thực hiện để hoàn tất công việc này.

1. Thay đổi thủ tục FormShow và thêm câu lệnh Dim vào đầu thủ tục với biến có tên strID.

```
Dim strID As String
```

2. Thay đổi thủ tục FormShow và thêm cột CategoryID và SupplierID vào câu lệnh SELECT.

3. Tìm đoạn mã được trình bày bên dưới và thêm dòng tô sáng bằng nét đậm.

```
strSql = "SELECT ProductID, ProductName, "
```

```
strSQL &= " SupplierID, CategoryID, "
strSQL &= " QuantityPerUnit, UnitPrice, "
strSQL &= " UnitsInStock, UnitsOnOrder, "
strSQL &= " ReorderLevel, Discontinued "
strSQL &= " FROM Products "
strSQL &= " WHERE ProductID = " & oItem.ID
```

4. Bây giờ tìm đoạn mã trong FormShow nạp dữ liệu từ OleDbDataReader vào các hộp văn bản và thêm các dòng sau bằng nét đậm.

```
txtID.Text = .Item("ProductID").ToString()
txtName.Text = .Item("ProductName").ToString()
strID = .Item("SupplierID").ToString()
Call FindItem(cboSupplier, strID)
strID = .Item("CategoryID").ToString()
Call FindItem(cboCategory, strID)
txtQty.Text = .Item("QuantityPerUnit").ToString()
txtPrice.Text = .Item("UnitPrice").ToString()
txtInStock.Text = .Item("UnitsInStock").ToString()
txtOnOrder.Text = .Item("UnitsOnOrder").ToString()
txtReorder.Text = .Item("ReorderLevel").ToString()
```

5. Kế tiếp bạn sẽ tạo hàm FindItem để chấp nhận tham chiếu hộp combo và biến chuỗi. Hàm này sẽ tìm biến chuỗi đó bên trong đặc tính ID của các đối tượng trong hộp combo và xác lập hộp combo để hiển thị dữ liệu được định vị trong vị trí đó.

```
Private Sub FindItem(ByVal cboCombo As ComboBox, _
 ByVal strID As String)
    Dim intLoop As Integer
    Dim boolFound As Boolean
    Dim oItem As PDSAListItemNumeric

    oItem = New PDSAListItemNumeric()
    For intLoop = 0 To cboCombo.Items.Count - 1
        oItem = CType(cboCombo.Items(intLoop), _
         PDSAListItemNumeric)
        If oItem.ID = CInt(strID) Then
```

```
                cboCombo.SelectedIndex = intLoop
                boolFound = True
                Exit For
            End If
        Next
        If Not boolFound Then
            cboCombo.SelectedIndex = -1
        End If
    End Sub
```

Hàm FindItem này lặp qua tất cả các giá trị trong hộp combo, mỗi lần xóa một mục khỏi hộp combo và chuyển đổi nó thành đối tượng PDSAListItemNumeric. Sau đó bạn có thể so sánh đặc tính ID của đối tượng này để xem nó có bằng giá trị được chuyển vào làm tham số thứ hai của hàm này. Nếu nó tìm thấy giá trị này, nó sẽ xác lập đặc tính SelectedIndex là vị trí này để buộc hộp combo tự định vị thành giá trị đó.

Bạn cũng có thể phải tạo các thủ tục cho kiểu ứng dụng client/server để thêm, soạn thảo và xóa dữ liệu. Bạn chỉ cần dùng đối tượng OleDbCommand để đệ trình câu lệnh INSERT, UPDATE và DELETE bằng phương thức ExecuteNonQuery của đối tượng này. Mẫu hoàn chỉnh đi cùng với chương này có các thủ tục được viết sẵn để bạn có thể khảo sát chúng.

Bạn có thể tăng thêm hiệu suất cho các tình huống nào đó bằng cách chuyển một hằng CommandBehavior khác vào phương thức ExecuteReader trên đối tượng OleDbCommand. Bạn được phép chỉ định *KeyInfo* để đọc vào những thông tin primary key. Bạn có thể chỉ định *SchemaOnly* để đọc vào thông tin giản đồ cột không có dữ liệu. Bạn có thể chỉ định *SingleColumn* nếu sẽ cho trở lại một giá trị kết hợp. Hoặc nếu bạn biết chỉ một dòng sẽ được cho trở lại, chỉ định *SingleRow* để có được hiệu suất tốt hơn trong tình huống này.

Tóm tắt

Lớp OleDbDataReader truy tìm dữ liệu nhanh và hiệu quả. Vì nó là con trỏ chỉ tiếp tới và bạn phải truy tìm các cột theo một thứ tự nào đó, có thể cần phải viết mã nhiều hơn một chút. Nhưng bạn sẽ thấy các lợi ích về hiệu suất đáng phải thực hiện điều đó.

Câu hỏi ôn tập

1. Đúng hay sai: Đối tượng DataReader có thể di chuyển tới lui trong dữ liệu?

2. Bạn gọi ra phương thức nào để cho trở lại đối tượng DataReader?

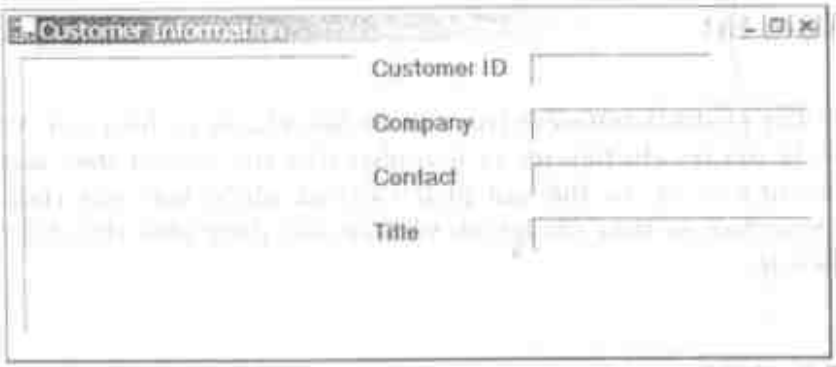
3. Phương thức nào được dùng để di chuyển con trỏ tới một dòng mới?

4. Bạn dùng đặc tính nào trên DataReader để truy tìm dữ liệu của các cột riêng biệt?

5. Nếu bạn sẽ cho trở lại chỉ một dòng dữ liệu, bạn sẽ chỉ định kiểu con trỏ nào trong phương thức ExecuteReader?

Bài tập

- Mở solution \Exercise\DataReaderExercise.sln cho chương này. Bạn sẽ thấy một form như sau.



- Tạo các thủ tục cần thiết để nạp hộp danh sách.
- Tạo thủ tục cần thiết để hiển thị dữ liệu sau khi nhấp một mục trong hộp danh sách.

Trả lời câu hỏi ôn tập

1. Sai.
2. ExecuteReader.
3. Read.
4. Item.
5. SingleRow.

Chương 20

DataSet và DataTable của ADO.NET

- Tìm hiểu cách dùng các đối tượng DataTable.
- Tìm hiểu cách dùng các đối tượng DataSet.
- Tìm hiểu cách sửa đổi dữ liệu bằng cách dùng các đối tượng DataSet.

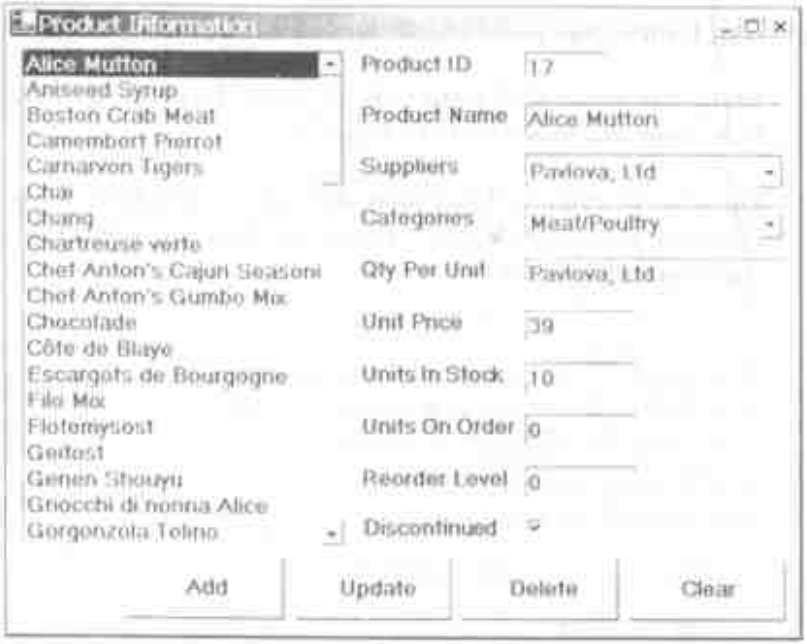
Sử dụng DataSet và DataTable của ADO.NET

Đối tượng ADO.NET DataSet giống như một CSDL trong bộ nhớ. Đối tượng này lưu giữ các đối tượng DataTable, trong đó mỗi đối tượng là một sự biểu thị dữ liệu đã được truy tìm bằng câu lệnh SELECT. Dữ liệu trong DataSet này được lưu trữ như XML. Các DataSet cũng có thể lưu trữ thông tin giản đồ, các sự ràng buộc và các mối quan hệ giữa nhiều đối tượng DataTable. Thông qua DataSet bạn có thể thêm, soạn thảo và xóa dữ liệu.

Lý do thực hiện điều này

Các DataSet được dùng khi bạn cần truy tìm dữ liệu từ một nguồn dữ liệu và sẽ sử dụng dữ liệu đó trong các control trên một form. Các DataSet có thể được liên kết trực tiếp với một số control hoặc bạn có thể thao tác chúng thủ công đối với một control khá sắc xảo. Còn có nhiều khả năng mềm dẻo khi bạn lập trình các đối tượng này trực tiếp và bạn có thể mất phần nào khả năng này nếu bạn dùng các phương thức liên kết. Tuy nhiên, bạn sẽ không phải viết mã.

Ở chương này bạn sẽ tìm hiểu cách tạo form nhập dữ liệu bằng cách dùng đối tượng DataTable và DataSet. Hình 1 trình bày màn hình mẫu bạn sẽ xây dựng. Bạn sẽ sử dụng bảng Products trong CSDL Northwind. CSDL mẫu này trở nên như một CSDL Access hoặc được cài đặt như một phần của SQL Server.



Hình 1: Màn hình Add/Edit/Delete sử dụng các DataSet.

Nạp ComboBox bằng cách dùng đối tượng DataTable

Có hai hộp combo trên form Product Information. Một cho các phân loại sản phẩm, một cho các nhà cung cấp sản phẩm. Bây giờ bạn tìm hiểu cách nạp dữ liệu vào các hộp combo này bằng cách dùng đối tượng DataTable.

Các bước sử dụng DataTable để nạp Combo Box

Thực hiện các bước sau để nạp hộp combo Supplier.

1. Nạp DataSetSample.sln từ folder \FirstStep.

2. Nhấp vào form frmProduct.vb và nhấp biểu tượng View Code hoặc lựa View | Code từ menu.

3. Tạo thủ tục SupplierLoad ngay dưới dòng "Windows Form Designer Generated Code". Thêm mã được trình bày bên dưới vào thủ tục mới này.

```
Private Sub SupplierLoad()
    Dim oAdapter As OleDb.OleDbDataAdapter
    Dim oTable As DataTable = New DataTable()
    Dim oItem As MSDN.ListItemNumeric
    Dim strSQL As String
    Dim strConn As String
    Dim intLoop As Integer

    strConn = ConnectionStringBuilder()

    strSQL = "SELECT SupplierID, CompanyName "
    strSQL &= "FROM Suppliers"

    Try
        oAdapter =
         New OleDb.OleDbDataAdapter(strSQL, strConn)
        oAdapter.Fill(oTable)

        For intLoop = 0 To oTable.Rows.Count() - 1
            oItem = New MSDN.ListItemNumeric()
            With oTable.Rows(intLoop)
                oItem.Value =
                 .Item("CompanyName").ToString()
                oItem.ID = CInt(.Item("SupplierID"))
            End With

            cboSupplier.Items.Add(oItem)
        Next

    Catch oExcept As Exception
        MessageBox.Show(oExcept.Message)

    End Try
End Sub
```

Bạn sẽ cần hai đối tượng để nạp hộp combo này. Bạn cần có OleDbDataAdapter là đối tượng được dùng để điền dữ liệu vào toàn bộ DataTable hoặc DataSet. Bạn cần có đối tượng DataTable để lưu giữ tất cả dữ liệu được truy tìm bằng câu lệnh SELECT được đệ trình với nguồn dữ liệu. Bạn cũng cần có lớp PDSAListItemNumeric mà bạn sẽ đặt giá trị vào để hiển thị trong hộp combo cùng với giá trị khóa chính từ bảng Suppliers. Lớp PDSAListItemNumeric được xây dựng sẵn cho bạn và được chứa trong solution bạn đã nạp ở đầu chương này.

Xây dựng chuỗi kết nối

Trước khi bạn có thể đệ trình một câu lệnh SQL với nguồn dữ liệu bằng ADO.NET, bạn cần cho ADO.NET các hướng mà nguồn dữ liệu này cư trú và nhà cung cấp nào sử dụng để tới dữ liệu này. Điều này được thực hiện bằng chuỗi của nhà cung cấp OLE DB. Trong thủ tục SupplierLoad bạn đã xây dựng, bạn đã gọi một hàm có tên ConnectStringBuild để cho trở lại chuỗi của nhà cung cấp này. Bây giờ bạn sẽ tạo hàm này.

Thêm hàm sau vào ngay dưới End Sub của thủ tục SupplierLoad bạn vừa tạo

```
Private Function ConnectStringBuild() As String
    Dim strConn As String

    strConn = "Provider=sqloledb;"
    strConn &= "Data Source=(local);"
    strConn &= "Initial Catalog=Northwind;"
    strConn &= "User ID=sa"

    Return strConn
End Function
```

Có thể bạn cần thay đổi chuỗi kết nối này nếu bạn đang dùng một SQL Server ở một nơi khác trên mạng của mình khác với

trên máy cục bộ của bạn. Nếu bạn không sử dụng SQL Server, bạn có thể sử dụng file NorthWind.mdb thường đi cùng với Microsoft Access như một CSDL mẫu. Nếu bạn đang sử dụng Access, bạn phải thay đổi chuỗi kết nối như sau:

```
Provider=Microsoft.Jet.OleDb.4.0;Data
Source=C:\Access\Northwind.mdb
```

Bạn sẽ thay đổi đường dẫn trong thuộc tính Data Source thành đường dẫn nơi Northwind.mdb được định vị.

Sử dụng Adapter dữ liệu để điền DataTable

Sau khi bạn xây dựng chuỗi kết nối và chuỗi SQL, bạn tạo một thể nghiệm mới của đối tượng OleDbDataAdapter và chuyển tới nó chuỗi kết nối và chuỗi SQL đó. Gọi ra phương thức Fill trên DataAdapter và chuyển tới nó đối tượng DataTable. Phương thức Fill sẽ mở kết nối với CSDL bằng cách dùng chuỗi kết nối đã cung cấp, điền dữ liệu vào DataTable, sau đó đóng kết nối.

Một khi bạn đã nạp dữ liệu vào DataTable, bạn có thể **lặp qua mỗi đối tượng DataRow để tạo thành DataTable. Mỗi lần qua vòng lặp bạn sẽ tạo một đối tượng PDSAListItemNumeric mới, thêm khóa chính (SupplierID) vào đặc tính ID và cột CompanyName vào đặc tính Value. Sau đó bạn thêm đối tượng PDSAListItemNumeric này vào ListBox.

Vì không có đặc tính ItemData trong VB.NET như trong VB6, bạn sẽ phải tạo một lớp như PDSAListItemNumeric để lưu giữ dữ liệu khóa chính cùng như giá trị văn bản để hiển thị. Bạn nên khảo sát lớp này để xem nó được kết hợp như thế nào.

Nạp các phân loại vào hộp Combo

Để nạp các phân loại vào hộp combo trên form Products này, bạn có thể chép và dán thủ tục SupplierLoad, đổi tên nó và đổi tên các cột và bảng trong câu lệnh SELECT.

Các bước nạp các phân loại vào hộp combo

1. Chép toàn bộ thủ tục SupplierLoad vào bộ nhớ bằng cách tô sáng mã và ấn Ctrl-C.

2. Dán mã ngay dưới End Sub của thủ tục SupplierLoad.

3. Đổi tên của thủ tục mới này thành CategoryLoad.

4. Thay đổi câu lệnh SELECT để sử dụng cột CategoryID và CategoryName.

5. Thay đổi câu lệnh SELECT để sử dụng tên bảng Categories.

6. Đổi các cột nạp lớp PDSAListItemNumeric là CategoryID và CategoryName.

Bây giờ bạn đã gõ vào toàn bộ mã để sử dụng đối tượng DataTable nạp Suppliers và Categories, chúng ta hãy xem chúng có hoạt động hay không.

1. Mở form để xem nó ở chế độ thiết kế.

2. Nhấp đôi vào nơi bất kỳ trên form (không phải trên control) và viết mã vào trong thủ tục sự kiện frmProduct_Load để gọi từng thủ tục bạn vừa viết.

```
Private Sub frmProducts_Load( _
 ByVal sender As System.Object, _
 ByVal e As System.EventArgs) Handles MyBase.Load
    ' Load Suppliers
    SupplierLoad()
    ' Load Categories
    CategoryLoad()
End Sub
```

3. Bây giờ chạy chương trình bằng cách ấn **F5**.

Nếu mọi thứ đều hoạt động tốt, bạn sẽ thấy một form sản phẩm còn trống, nhưng hộp combo Supplier và Category phải có dữ liệu ở trong.

Tạo đối tượng DataSet

Đối tượng DataTable lưu giữ dữ liệu cần thiết của một bảng và hữu dụng để nạp các hộp combo và các hộp danh sách. Đối tượng DataSet như một wrapper (trình bao bọc) chung quanh một hoặc nhiều đối tượng DataTable. Bây giờ bạn sẽ tìm hiểu cách dùng đối tượng DataSet.

Khi bạn đang kết hợp màn hình nhập dữ liệu, tạo một DataSet bạn khai báo như một đặc tính của lớp Form rất hữu dụng. Sau đó đặc tính này có thể được dùng khắp form để nạp dữ liệu vào hộp danh sách, bổ sung, soạn thảo và xóa dữ liệu trên form.

Các bước tạo DataSet

1. Mở file frmProducts.vb trong project mẫu.

2. Mở cửa sổ mã và đi tới ngay đầu mã form.

3. Ngay sau câu lệnh Inherits, thêm biến Private có tên moDS được khai báo như một lớp DataSet.

```
Public Class frmProducts
    Inherits System.Windows.Forms.Form

    Private moDS As DataSet
```

Nó sẽ tạo một biến riêng có thể được dùng khắp form. Kế tiếp bạn sẽ cần tạo DataSet đó và nạp dữ liệu vào nó.

4. Tạo một thủ tục mới có tên DataSetCreate và gõ mã được trình bày bên dưới vào thủ tục mới này.

```
Private Sub DataSetCreate()
    Dim oAdapter As OleDb.OleDbDataAdapter
    Dim strSQL As String
    Dim strConn As String

    ' Get Connection String
    strConn = ConnectStringBuild()

    ' Build SQL String
    strSQL = "SELECT * "
    strSQL &= "FROM Products"

    moDS = New DataSet()
    Try
        ' Create New Data Adapter
        oAdapter =
         New OleDb.OleDbDataAdapter(strSQL, strConn)
        ' Fill DataSet From Adapter and give it a name
        oAdapter.Fill(moDS , "Products")
        ' Create a Primary Key
        With moDS.Tables("Products")
            .PrimaryKey = New DataColumn() _
              {.Columns("ProductID")}
        End With
```

```
        Catch oExcept As Exception
            MessageBox.Show(oExcept.Message)

        End Try
    End Sub
```

Hầu hết mã trên bây giờ nhìn sẽ quen thuộc. Bạn sẽ xây dựng một chuỗi kết nối và câu lệnh SQL SELECT. Bạn cũng sẽ tạo OleDbDataAdapter để nạp DataSet giống như bạn đã làm với DataTable. Từ đó mọi thứ nhìn hơi khác. Bạn vẫn sẽ dùng phương thức Fill, nhưng bạn sẽ chuyển cho nó đối tượng DataSet và tên bạn muốn kết hợp với DataTable mới được tạo trong DataSet này. Ở trường hợp trên, bạn sẽ cho DataTable mới này tên "Products". Sau đó bạn dùng tên này để tham chiếu bảng trong tập hợp Tables của DataSet bạn muốn dùng.

Bạn có thể tham chiếu DataTable riêng bằng cách dùng tập hợp Tables. Thí dụ, moDS.Tables("Products") sẽ cho trở lại đối tượng DataTable được tạo khi bạn đã thực hiện phương thức Fill trên OleDbDataAdapter. Nhớ là bạn đã chuyển "Products" như tham số thứ hai trên phương thức Fill, vì vậy nó đã gán tên này vào bảng DataTable. Bây giờ bạn sẽ dùng kĩ thuật này để cho DataTable biết cột nào là khóa chính của nó.

Nếu bạn định thực hiện một tìm kiếm nào đó ở DataTable này trong DataSet, bạn sẽ phải cho DataTable biết cột PrimaryKey như thế nào. Bạn thực hiện điều này bằng cách xác lập đặc tính PrimaryKey là mảng New DataColumn. Vì các bảng có thể có một hoặc nhiều cột như khóa chính của chúng mà bạn cần tạo một mảng các đối tượng DataColumn để chuyển tới đặc tính PrimaryKey này. Bạn đã làm gì ở mã trên để truy tìm cột .Columns("ProductID") và đặt vào trong hai dấu ngoặc ôm {} ngay sau khi khai báo mảng DataColumn mới. Các dấu ngoặc ôm là cách bạn khai báo các phần tử bạn muốn đi vào mảng mới bạn đang tạo. Nếu bạn có nhiều cột, bạn sẽ tách mỗi cột bằng một dấu phẩy bên trong các dấu ngoặc ôm.

Tải hộp danh sách từ DataSet

Bạn đã tạo DataSet của dữ liệu sản phẩm và đã lưu trữ nó trong một biến có tên *moDS*. Bạn có thể dùng biến đó bất kỳ nơi đâu bên trong form này. Bây giờ bạn xây dựng một thủ tục để tải dữ liệu sản phẩm từ DataSet vào hộp danh sách trên form này.

Các bước nạp List Box

1. Tạo một thủ tục mới có tên ListLoad ở một nơi nào đó bên trong form.

2. Gõ mã sau vào thủ tục này.

```
Private Sub ListLoad()
    Dim oItem As PDSAListItemNumeric
    Dim oRow As DataRow

    lstProducts.Items.Clear()
    ' Loop through each row and get a DataRow
    For Each oRow In moDS.Tables("Products").Rows
        ' Create New Item to hold PK and Description
        oItem = New PDSAListItemNumeric()
        With oItem
            .ID = CInt(oRow.Item("ProductID"))
            .Value
oRow.Item("ProductName").ToString()
        End With

        ' Add Item to list box
        lstProducts.Items.Add(oItem)
    Next

    lstProducts.SetSelected(0, True)
End Sub
```

Mã trong thủ tục ListLoad rất đơn giản. Mỗi DataTable trong DataSet được tạo thành bởi các đối tượng DataRow. Bạn có thể

vòng lặp qua mỗi DataRow trong DataTable và mỗi lần qua vòng lặp bạn có thể xây dựng một đối tượng FOSAListItemNumeric để đưa ProductID và ProductName vào. Sau đó bạn thêm đối tượng mới này vào List Box.

Bây giờ bạn có thể chạy mã bạn đã gõ vào cho tới điểm này và để nó nạp hộp danh sách đầy đủ các tên sản phẩm.

1. Thay đổi thủ tục sự kiện frmProduct_Load để gọi hai thủ tục mới bạn đã tạo.

2. Thêm các dòng mã được trình bày bên dưới bằng nét đậm.

```
Private Sub frmProduct_Load(
 ByVal sender As System.Object,
 ByVal e As System.EventArgs) Handles MyBase.Load
    ' Load Suppliers
    SupplierLoad()
    ' Load Categories
    CategoryLoad()

    ' Initialize the Dataset
    DataSetCreate()
    ' Load List Box of Products
    ListLoad()
End Sub
```

3. Bây giờ ấn **F5** để chạy chương trình này.

4. Khi form xuất hiện bạn sẽ có một danh sách đầy đủ các tên sản phẩm.

Tìm một dòng riêng biệt trong DataSet

Khi bạn nhấp vào một sản phẩm trong hộp danh sách bạn sẽ hiển thị dữ liệu chi tiết cho mỗi sản phẩm trong các control thích

hợp ở bên phải hộp danh sách. Điều đầu tiên bạn cần làm là viết mã để đáp ứng sự kiện nhấp chuột của người dùng trên hộp danh sách.

1. Đưa form **Products** vào chế độ thiết kế.

2. Nhấp đôi vào hộp danh sách để hiển thị thủ tục sự kiện SelectedIndexChanged.

3. Thêm mã để gọi thủ tục có tên FormShow như được trình bày bên dưới.

```
Private Sub lstProducts_SelectedIndexChanged( _
 ByVal sender As Object, ByVal e As System.EventArgs) _
 Handles lstProducts.SelectedIndexChanged
    FormShow()
End Sub
```

4. Bây giờ bạn viết thủ tục FormShow để tìm một dòng riêng biệt bên trong DataSet, sau đó truy tìm thông tin từ dòng đó và điền vào tất cả các control thích hợp trên form sản phẩm.

5. Thêm một thủ tục mới vào form sản phẩm và đặt tên nó là FormShow.

6. Viết mã bên dưới.

```
Private Sub FormShow()
    Dim cDR As DataRow
    Dim oItem As PDSAListItemNumeric
    Dim strID As String

    ' Get Primary Key From List Box
    oItem = CType(lstProducts.SelectedItem, _
     PDSAListItemNumeric)
```

```
    ' Find row in DataSet
    With moDS.Tables("Products").Rows
        oDR = .Find(oItem.ID)
    End With

    ' Display the Data
    txtID.Text = oDR("ProductID").ToString()
    txtName.Text = oDR("ProductName").ToString()
    strID = oDR("SupplierID").ToString()
    Call FindItem(cboSupplier, strID)
    strID = oDR("CategoryID").ToString()
    Call FindItem(cboCategory, strID)
    txtQty.Text = oDR("QuantityPerUnit").ToString()
    txtPrice.Text = oDR("UnitPrice").ToString()
    txtInStock.Text = oDR("UnitsInStock").ToString()
    txtOnOrder.Text = oDR("UnitsOnOrder").ToString()
    txtReorder.Text = oDR("ReorderLevel").ToString()
    chkDisc.Checked     =     CType(oDR("Discontinued"),
Boolean)
End Sub
```

Điều đầu tiên thủ tục FormShow thực hiện là cho trở lại SelectedItem từ hộp danh sách. Mỗi mục trong hộp danh sách là một đối tượng PDSAListItemNumeric. Vì hộp danh sách chỉ lưu giữ các kiểu dữ liệu Object, bạn phải dùng hàm CType để chuyển đổi Object thành kiểu dữ liệu PDSAListItemNumeric. Sau đó bạn có thể truy tìm đặc tính ID từ đối tượng này để nhận khóa chính cho dòng bạn vừa nhấp.

Phương thức Find trên Rows sẽ cho trở lại đối tượng DataRow. Trong thủ tục FormShow bạn đã khai báo một biến có tên *oDR* như đối tượng DataRow. Bạn chuyển đặc tính ID tới phương thức Find để nó cho trở lại tham chiếu với DataRow đã tìm thấy. Một khi bạn có đối tượng DataRow này, bạn có thể tới dữ liệu trong từng cột. Bạn sẽ chỉ có thể gọi phương thức Find nếu bạn đã xác lập đặc tính PrimaryKey trên DataTable.

Thêm các dòng vào DataSet

Ở điểm nào đó, bạn sẽ muốn cho phép người dùng thêm các dòng vào các bảng của bạn. Bạn có thể thực hiện điều này bằng cách đệ trình câu lệnh INSERT thông qua đối tượng OleDbCommand hoặc bạn có thể dùng đối tượng DataSet bạn đã tạo rồi. Có một vài bước bạn cần thực hiện để thêm một dòng mới vào DataSet và CSDL. Vì DataSet được ngưng kết nối với CSDL, đầu tiên bạn phải thêm dữ liệu vào DataSet. Kế tiếp bạn phải xây dựng một kết nối với CSDL ấy và xây dựng câu lệnh INSERT. Microsoft đã cung cấp một đối tượng để xây dựng câu lệnh INSERT này cho bạn. Nó được gọi là đối tượng OleDbCommandBuilder.

Các bước thêm một dòng mới

1. Thêm thủ tục mới vào form sản phẩm của bạn.
2. Đặt tên thủ tục này là DataAdd.
3. Viết mã được trình bày bên dưới.

```
Private Sub DataAdd()
    Dim oAdapter As OleDb.OleDbDataAdapter
    Dim oBuild As OleDb.OleDbCommandBuilder
    Dim oDR As DataRow
    Dim strSQL As String
    Dim strConn As String

    ' Create New DataRow Object From DataSet
    oDR = moDS.Tables("Products").NewRow()
    oDR.BeginEdit()

    ' Load new data into row
    oDR("ProductName") = txtName.Text
    oDR("SupplierID") = CType(cboSupplier.SelectedItem,
_
```

```
        PDEAlreadyNumeric().ID
        oDR("CategoryID") = CType(cboCategory.SelectedItem,
        PDEAlreadyNumeric).ID
        oDR("QuantityPerUnit") = txtQty.Text
        oDR("UnitPrice") = CDec(txtPrice.Text)
        oDR("UnitsInStock") = CShort(txtInStock.Text)
        oDR("UnitsOnOrder") = CShort(txtOnOrder.Text)
        oDR("ReorderLevel") = CShort(txtReorder.Text)
        oDR("Discontinued") = CBool(chkDisc.Checked)

        ' Tell DataRow you are done adding data
        oDR.EndEdit()
        ' Add DataRow to DataSet
        moDS.Tables("Products").Rows.Add(oDR)

        Try
            ' Get Connection String
            strConn = ConnectStringBuild()
            ' Build SQL String
            strSQL = "SELECT * FROM Products"
            ' Create New DataAdapter
            oAdapter =
             New OleDb.OleDbDataAdapter(strSQL, strConn)
            ' Create CommandBuild from Adapter
            ' This will build INSERT, UPDATE and DELETE SQL
            oBuild                                    New
OleDb.OleDbCommandBuilder(oAdapter)

            ' Get Insert Command Object
            oAdapter.InsertCommand                    =
oBuild.GetInsertCommand()

            ' Submit INSERT statement through Adapter
            oAdapter.Update(moDS, "Products")
            ' Tell DataSet changes to data source are
complete
            moDS.AcceptChanges()
            ' Close the Connection
            oAdapter.InsertCommand.Connection.Close()

            ' Recreate the DataSet
            DataSetCreate()
            ' Reload the list box
            ListLoad()

        Catch oException As Exception
```

```
MessageBox.Show(ex.Exception.Message)
    End Try
End Sub
```

Để thêm một bản ghi mới vào bảng trong CSDL của bạn, đầu tiên bạn phải thêm một dòng mới vào DataTable trong DataSet của bạn. Bạn có thể thực hiện điều này bằng cách trước hết sử dụng phương thức NewRow tạo DataRow mới. Bạn gọi ra phương thức BeginEdit trên DataRow mới này để bạn có thể đặt dữ liệu vào các cột thích hợp. Khi bạn đã cập nhật tất cả các cột, bạn gọi phương thức EndEdit. Bạn thêm DataRow mới này vào DataTable trong DataSet bằng cách chuyển DataRow tới phương thức Add của tập hợp Rows trong DataTable.

Sử dụng đối tượng Command Builder để tạo SQL

Bây giờ dữ liệu đã ở trong DataSet, bạn có thể xây dựng OleDbDataAdapter để đệ trình dữ liệu mới này với CSDL. Bạn tạo DataAdapter bằng cách chuyển vào cùng câu lệnh SQL bạn đã dùng để nạp DataSet và chuỗi kết nối. Sau đó bạn chuyển đối tượng DataAdapter tới cấu tử của lớp OleDbCommandBuilder và nó sẽ tạo một đối tượng Builder mới cho bạn. Sau đó phương thức GetInsertCommand có thể được gọi trên đối tượng CommandBuilder này để truy tìm đối tượng lệnh chứa câu lệnh INSERT cho bảng này. Câu lệnh INSERT có các trình giữ chỗ cho mỗi đoạn dữ liệu trong DataSet. Câu lệnh INSERT này thể hiện điều gì đó như sau:

```
INSERT INTO "Products"( "ProductName" , "SupplierID" ,
"CategoryID" , "QuantityPerUnit" , "UnitPrice" ,
"UnitsInStock" , "UnitsOnOrder" , "ReorderLevel" ,
"Discontinued" ) VALUES ( ? , ? , ? , ? , ? , ? , ? , ?
, ? )
```

Bất kỳ nơi nào có một dấu chấm hỏi trong câu lệnh là nơi dữ liệu của DataSet sẽ được thay thế khi bạn đệ trình câu lệnh

INSERT này thông qua DataAdapter. Bạn cho DataSet biết đệ trình câu lệnh INSERT này bằng cách chuyển vào đối tượng DataSet và tên của bảng vào DataSet để được cập nhật thành phương thức Update của DataAdapter. Sau khi phương thức Update này hoàn tất, bạn gọi ra phương thức AcceptChanges trên DataSet. Nó thông báo cho DataRow mới là đã được cập nhật trong CSDL.

Bây giờ bạn đã tạo thủ tục này để thêm một dòng mới, bạn nên thử nó.

1. Đưa form vào chế độ thiết kế.

2. Nhấp đôi nút Add.

3. Trong thủ tục sự kiện btnAdd_Click, gọi thủ tục DataAdd.

```
Private Sub btnAdd_Click(ByVal sender As Object,
 ByVal e As System.EventArgs) Handles btnAdd.Click
    DataAdd()
End Sub
```

4. Ấn **F5** để chạy ứng dụng.

5. Gõ "A New Product" vào hộp văn bản Product Name.

6. Nhấp nút Add.

7. Bạn sẽ thấy sản phẩm mới trong hộp danh sách.

Cập nhật các dòng trong DataSet

Việc cập nhật các dòng trong DataSet hầu như giống việc thêm các dòng. Thay vì thêm một dòng mới, bạn chỉ cần tìm dòng sẵn có bạn muốn cập nhật. Sau đó bạn cập nhật dữ liệu vào các cột thích hợp trong DataRow, xây dựng DataAdapter và đối tượng CommandBuilder, truy tìm đối tượng lệnh UPDATE. Sau đó bạn có thể đệ trình câu lệnh UPDATE bằng phương thức Update của DataAdapter, chấp nhận các thay đổi bạn thực hiện xong.

- Gõ mã bên dưới vào như một thủ tục mới trong form này.

```
Private Sub DataUpdate()
    Dim oAdapter As OleDb.OleDbDataAdapter
    Dim oBuild As OleDb.OleDbCommandBuilder
    Dim oDR As DataRow
    Dim strSQL As String
    Dim intID As Integer
    Dim strConn As String

    ' Get Primary Key From List Box
    intID = CType(lstProducts.SelectedItem, _
     PDSAListItemNumeric).ID

    ' Find Row To Update
    oDR = moDS.Tables("Products").Rows.Find(intID)

    ' Begin the editing process
    oDR.BeginEdit()

    ' Load new data into row
    oDR("ProductName") = txtName.Text
    oDR("SupplierID") = CType(cboSupplier.SelectedItem,
_
     PDSAListItemNumeric).ID
    oDR("CategoryID") = CType(cboCategory.SelectedItem,
_
     PDSAListItemNumeric).ID
    oDR("QuantityPerUnit") = txtQty.Text
```

```
        oDR("UnitPrice") = CDec(txtPrice.Text)
        oDR("UnitsInStock") = CShort(txtInStock.Text)
        oDR("UnitsOnOrder") = CShort(txtOnOrder.Text)
        oDR("ReorderLevel") = CShort(txtReorder.Text)
        oDR("Discontinued") = CBool(chkDisc.Checked)

        ' End the editing process
        oDR.EndEdit()

        Try
            ' Get Connection String
            strConn = ConnectStringBuild()
            ' Build SQL String
            strSQL = "SELECT * FROM Products "
            ' Create New DataAdapter
            oAdapter = New _
             OleDb.OleDbDataAdapter(strSQL, strConn)
            ' Create CommandBuild from Adapter
            ' This will build INSERT, UPDATE and DELETE SQL
            oBuild                    =                    New
OleDb.OleDbCommandBuilder(oAdapter)

            ' Get Update Command Object
            oAdapter.UpdateCommand                         =
oBuild.GetUpdateCommand()

            ' Submit UPDATE through Adapter
            oAdapter.Update(moDS, "Products")
            '  Tell  DataSet  changes  to  data  source  are
complete
            moDS.AcceptChanges()
            ' Close the Connection
            oAdapter.UpdateCommand.Connection.Close()

            ' Reload the list box
            ListLoad()

        Catch oException As Exception
            MessageBox.Show(oException.Message)

        End Try
    End Sub
```

Như bạn thấy, mã này hầu như giống thủ tục DataAdd bạn đã viết trước đây. Sự khác nhau lớn nhất là thay vì gọi phương thức

GetInsertCommand, bạn gọi phương thức GetUpdateCommand. Bạn đặt đối tượng lệnh được truy tìm từ GetUpdateCommand vào đặc tính UpdateCommand trên DataAdapter.

Bây giờ bạn đã tạo thủ tục này để thêm dòng mới, bạn nên thử nó.

1. Đưa form vào chế độ thiết kế.

2. Nhấp đôi nút Update.

3. Trong thủ tục sự kiện btnUpdate_Click, gọi thủ tục DataUpdate.

```
Private Sub btnUpdate_Click(ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles btnUpdate.Click
    DataUpdate()
End Sub
```

4. Ấn **F5** để chạy ứng dụng.

5. Đổi một trong các tên sản phẩm và có thể một vài trường khác trên một trong các bản ghi.

6. Nhấp nút Update.

7. Nhấp một sản phẩm khác, sau đó nhấp lại sản phẩm bạn đã cập nhật.

8. Bạn sẽ thấy dữ liệu đã cập nhật xuất hiện trong các control trên form sản phẩm.

Xóa các dòng trong DataSet

Vào lúc này bạn sẽ thấy một mẫu để cập nhật dữ liệu thông qua đối tượng DataSet. Thực ra, để xóa dữ liệu khỏi DataSet, bạn sẽ viết hầu như mã giống nhau, nhưng thay vì cập nhật dữ liệu thích hợp trong DataRow, bạn sẽ dùng phương thức Delete. Sau đó bạn sẽ đệ trình thay đổi này bằng DataAdapter giống như bạn đã làm trong thủ tục DataAdd và DataUpdate bạn đã viết.

1. Thêm một thủ tục mới có tên DataDelete vào form.

2. Viết mã sau vào thủ tục mới này.

```
Private Sub DataDelete()
    Dim oAdapter As OleDb.OleDbDataAdapter
    Dim oBuild As OleDb.OleDbCommandBuilder
    Dim oDR As DataRow
    Dim strSQL As String
    Dim strConn As String
    Dim intID As Integer

    ' Get Connection String
    strConn = ConnectStringBuild()

    ' Get Primary Key From List Box
    intID = CType(lstProducts.SelectedItem, _
     PDSAListItemNumeric).ID

    ' Find DataRow To Delete
    oDR = moDS.Tables("Products").Rows.Find(intID)
    ' Mark DataRow for deletion
    oDR.Delete()

    Try
        ' Build SQL String
        strSQL = "SELECT * FROM Products "
        ' Create New DataAdapter
        oAdapter = New _
         OleDb.OleDbDataAdapter(strSQL, strConn)
        ' Create CommandBuild from Adapter
```

```
        ' This will build INSERT, UPDATE and DELETE SQL
        oBuild                                          New
OleDb.OleDbCommandBuilder(oAdapter)

        ' Get Delete Command Object
        oAdapter.DeleteCommand                          =
oBuild.GetDeleteCommand()

        ' Submit DELETE through Adapter
        oAdapter.Update(moDS, "Products")
        '  Tell  DataSet  changes  to  data  source  are
complete
        moDS.AcceptChanges()
        ' Close the Connection
        oAdapter.DeleteCommand.Connection.Close()

        ' Reload the list box
        ListLoad()

    Catch oException As Exception
        MessageBox.Show(oException.Message)

    End Try
End Sub
```

Ở mã trên bạn tìm thấy dòng bạn muốn xóa, sau đó áp dụng phương thức Delete vào đối tượng DataRow đó. Điều này đánh dấu dòng này để xóa trong DataSet. Một lần nữa bạn sử dụng đối tượng CommandBuilder để nhận đối tượng DeleteCommand. Sau đó bạn gọi ra phương thức Update trên DataAdapter để đệ trình câu lệnh DELETE này với CSDL.

Bây giờ bạn đã tạo thủ tục này để xóa dòng, bạn nên thử nó.

1. Đưa form vào chế độ thiết kế.

2. Nhấp đôi nút Delete.

3. Trong thủ tục sự kiện btnDelete_Click, gọi thủ tục DataDelete.

```
Private Sub btnDelete_Click(ByVal sender As Object, _
 ByVal e As System.EventArgs) Handles btnUpdate.Click
    DataDelete()
End Sub
```

4. Ấn **F5** để chạy ứng dụng.

5. Định vị một trong các sản phẩm.

6. Nhấp nút Delete.

7. Bây giờ bạn sẽ thấy sản phẩm này biến mất khỏi hộp** danh sách.

Tóm tắt

Ở chương này bạn đã tìm hiểu cách dùng DataTable và DataSet để nạp và sửa đổi dữ liệu trong CSDL của bạn. DataSet chỉ được tạo thành bằng một hoặc nhiều đối tượng DataTable. Việc cập nhật dữ liệu có thể được thực hiện mà không phải viết bất kỳ câu lệnh SQL nào. Bạn có thể để các đối tượng CommandBuilder viết mã cho bạn. Các đối tượng này làm cho việc tạo màn hình nhập dữ liệu chuẩn rất dễ tạo.

Câu hỏi ôn tập

1. Dữ liệu bên trong DataSet được lưu trữ như thế nào?

2. Đúng hay sai: Bạn không thể xác lập các mối quan hệ giữa các bảng trong DataSet không?

3. Bạn sử dụng đối tượng nào để điền dữ liệu vào DataSet hoặc DataTable?

4. Bạn có phải đóng dứt khoát kết nối sau khi điền một DataSet?

5. Để sử dụng phương thức Find bạn phải xác lập đặc tính nào trước?

6. Bạn phải sử dụng phương thức nào trước khi cập nhật dữ liệu ở một dòng trong DataTable?

7. Bạn sử dụng đối tượng nào để xây dựng đối tượng lệnh để chèn, cập nhật hoặc xóa dữ liệu?

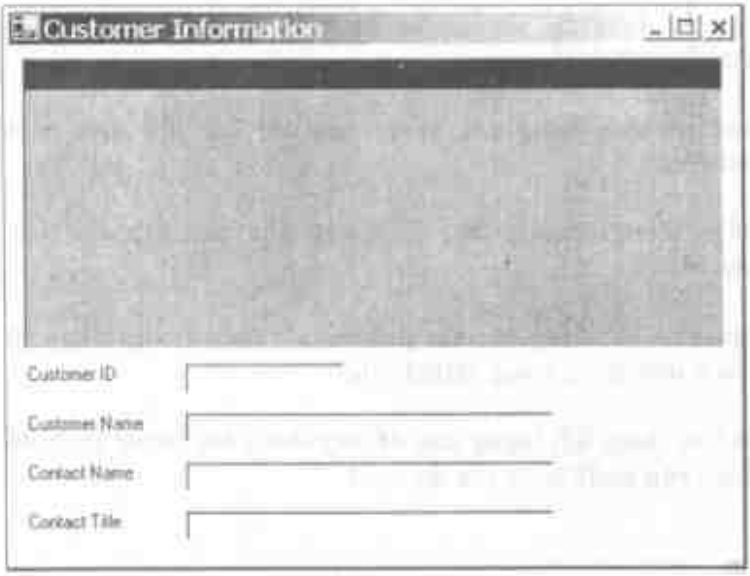
Bài tập

1. Nạp chương trình mẫu từ folder \Exercises dưới các mẫu của chương này.

2. Bạn sẽ tìm thấy một form có tên frmCust giống như sau.

3. Nạp khung lưới với CustomerID, CompanyName, ContactName và ContactTitle bằng cách dùng DataSet.

4. Khi bạn nhấp một mục trong khung lưới, hãy thực hiện tìm kiếm trong DataSet và hiển thị các trường trong các hộp văn bản bên dưới khung lưới.



Hình 2: Màn hình khách hàng thiết lập.

Trả lời câu hỏi ôn tập

1. XML.
2. Sai.
3. DataAdapter.
4. Không.
5. PrimaryKey.
6. BeginEdit
7. OleDbCommandBuilder.