Chương 2: Windows application

Tổng quan:

Trong lúc những ứng dụng kinh doanh phát triển ngày nay được thiết kế bởi World Wide Web, những client cổ điển vẫn tồn tại và sẽ luôn luôn được yêu cầu. Nó là một ứng dụng Intranet sử dụng bên trong một tổ chức hay là mẫu phần mềm cài đặt trên máy để bàn. Những chức năng mạnh và kinh nghiệm của người sử dụng như một môi trường cung cấp sẽ luôn luôn được yêu cầu cho những kiểu ứng dụng. Web form thì tuyệt vời nhưng chúng không thể so sánh kinh nghiệm của người sử dụng thông qua một client tốt.

May mắn, .NET cung cấp một khả năng để tạo những client mạnh thực thi bên trong Common Language Runtime. Ứng dụng này gọi là Window form. Bất kỳ ngôn ngữ .NET nào cũng có thể sử dụng Window Form để xây dựng Windows Applications. Những ứng dụng này được truy cập đến .NET FrameWork của các namespace và đối tượng.

Trong chương này ta bàn luận về cách để xây dựng các ứng dụng Windows trong .NET. Chúng ta sẽ bàn luận về một số chủ đề sau:

- Cách xây dựng ứng Window form sử dụng .NET FrameWork.
- Cách sử dụng Visual studio.NET để xây dựng ứng dụng Window form nhanh chóng.
- Thêm những menu hỗ trợ vào một ứng dụng bao gồm dynamic và context-sensitive menu.
- Các tài nguyên Utilizing custom và common dialog trong một đề án
- Cách sử dụng Visual inheritance để xây dựng ứng dụng Window Form.
- Cách sử dụng Window Form để điều khiển một ứng dụng
- Cách tạo và mở rộng những điều khiển cho những chức năng đặc biệt.

- Các sự kiện từ custom control

2.1 Windows Applications in .NET

Thật là quan trọng để hiểu sự khác nhau giữa rich client và thin client, bởi vì nó là điểm cốt yếu để hiểu tại sao Windows applications được xem như rich clients. Một thin client không yêu cầu quá nhiều cài đặt và hình thể trước khi sử dụng những ứng dụng của nó. Rich clients đôi lúc gọi là fat clients, yêu cầu một vài mẫu cài đặt và hình thể trên máy client. Các rich client khi chạy trên client, thì chúng có thể đạt được thuận lợi đầy đủ của môi trường và năng lượng xử lý của máy client. Vấn đề này cho phép các nhà phát triển tạo một sự tác động qua lại và thân thiện với người dùng hơn.

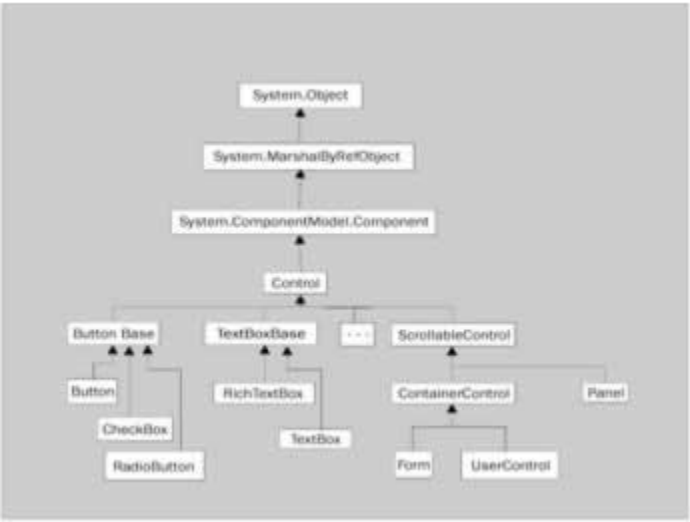
Trước .NET, các nhà phát triển có vài chọn lựa trong việc xây dựng một ứng dụng Window. Họ có thể có nhiều cách đi xuống lớp API và xây dựng một ứng dụng C hay C++ sử dụng Win32 API. Việc này là công việc rất khó và chi phối thời gian và không có nhiều cơ sở nào có thể bỏ ra nhiều thời gian để phát triển ứng dụng tại tầng lớp này. Bởi vì thế, Các môi trường mở ra để cung cấp một giao diện dễ dàng hơn cho Win32 API và cho phép các nhà phát triển có nhiều chọn lựa hơn. Microsoft Foundation Classes (MFC) là một thư viện lớp sử dụng C++ được gói gọn trong Win32 API. Visual Basic là một công cụ phát triển ứng dụng nhanh chóng, sử dụng một nguồn gốc của ngôn ngữ lập trình Basic cho phép mọi nhà phát triển tạo các ứng ụng Window tương đối nhanh chóng. Tuy nhiên, các giải pháp này đều không sử dụng .NET Framework hay CLR.

.NET Framework chứa một tầng lớp năng xuất mới gói gọn trong Win32 API, và giống như MFC và Visual Basic, nó cho phép phát triển hiệu suất cao hơn và dễ dàng hơn của các ứng dụng Window. Môi trường này được gọi là Windows Forms; nó cho phép các nhà phát triển tạo một tương tác cấp cao và các ứng dụng Window lớn sử dụng bất kỳ ngôn ngữ .NET nào. Chúng ta sẽ được xem xét môi trường Windows Forms trong chương này.

Bằng cách sử dụng .NET để tạo các ứng dụng Window, các nhà phát triển biết được nhiều tính năng mới và hay được cung cấp bởi .NET. Tất cả namespaces và classes

trong .NET Framework có thể sử dụng bên trong một .NET rich client. Thêm vào đó, ứng dụng có thể được phát triển trong bất kỳ ngôn ngữ .NET nào; các thành phần khác nhau của ứng dụng có thể sử dụng các ngôn ngữ khác nhau.

Hầu như tất cả chức năng chúng ta đang xem xét trong chương này không được giải thích thông qua các lớp chứa bên trong *System.Windows.Forms* namespace. Đây là một namespaces lớn chứa nhiều lớp và các namespace phụ vào, tất cả chúng làm cho ta dễ dàng tạo các ứng dụng Window. Biểu đồ bên dưới hiển thị vài lớp thừa hưởng từ *System.Windows.Forms.Control*. Đối tượng này hành động như một lớp cơ sở cho đa số lớp trong namespace này, và chứa nhiều chức năng cơ bản của giao diện hiển thị và tương tác với người dùng.



Trong biểu đồ trên, nếu một lớp không được thêm vào đầu một namespace nó được chứa trong *System.Windows.Forms* namespace. Không hiển thị số lượng lớn lớp thừa hưởng từ Control, tất cả chúng cung cấp chức năng chuyên dụng. Chúng cung cấp truy cập đến thư viện bao quát của các control Windows Form. Biểu đồ này cung cấp một cái nhìn tổng quát về cách thừa kế và cách các lớp Form và Control ăn khớp trong sơ đồ lớn.

- *System.Windows.Forms.Control* - hành động này như lớp cơ bản cho phần lớn các lớp trong namespace. Nó chứa chức năng cơ bản của thao tác xử lý bàn phím và nhập từ chuột và xử lý tin nhắn window.

- *System.Windows.Forms.ButtonBase* - Lớp này hỗ trợ chức năng cơ bản của một nút mà mọi lớp thừa hưởng sử dụng trong các cách khác nhau.
- *System.Windows.Forms.TextBoxBase* - một lần nữa, lớp này là một lớp cơ sở được sử dụng để cung cấp chức năng và thuộc tính thông thường cho các lớp thừa hưởng. Cả hai lớp TextBox và RichTextBox sử dụng chức năng cung cấp bởi TextBoxBase.
- *System.Windows.Forms.ScrollableControl* - đây là một lớp cơ bản khác cung cấp hỗ trợ cho các lớp thừa hưởng. Lớp này quản lý sự phát sinh và hiển thị của các thanh cuộn đến người dùng để truy cập đến gốc của một hiển thị.
- *System.Windows.Forms.ContainerControl* - Lớp này quản lý chức năng yêu cầu cho một control để hành động như một sự chứa đựng những control khác.
- *System.Windows.Forms.Panel* - đây là control khác có thể chứa các control thêm vào, nhưng khác với lớp *ContainerControl*, nó phân loại các control một cách đơn giản.
- *System.Windows.Forms.Form* - Đây là lớp mà phân phát với việc tạo ra và hiển thị các cửa sổ. Lớp này có thể được dùng để tạo bất kỳ loại cửa sổ nào: standard, toolbox, borderless, even modal dialog boxes và multi-document interfaces.
- *System.Windows.Forms.UserControl* - Đây là lớp có thể được dùng để thừa hưởng từ việc tạo một custom control đến việc được dùng trong một nơi phức tạp trong một ứng dụng hay tổ chức.

2.2 Windows Forms

Hầu hết mọi ứng dụng Windows Form mở rộng chức năng của *System.Windows.Forms*. Chức năng cơ bản của lớp Form không thể tạo một cửa sổ có thể sống và tương tác trong môi trường Windows một cách đúng đắn. Đây là một thuận lợi như một điểm khởi đầu và bằng việc mở rộng lớp Form và thêm các control tuỳ biến và

các bộ điều khiển sự kiện tuỳ biến, một ứng dụng rất hữu ích được tạo để có thể tương tác với người dùng và dữ liệu hiện tại thông qua một giao diện người dùng tinh vi.

Chúng ta đang xem xét cách tiến trình này làm việc theo hai cách. Để hiểu tốt hơn cách mà Windows Forms hoạt động và cách nó tương tác với .NET Framework, chúng ta sẽ xây dựng một ứng dụng Window hoàn toàn mà không sử dụng Visual studio.NET. Nó sẽ cung cấp cho bạn một sự đánh giá mạnh mẽ về Visual studio.NET khi chúng ta chuyển đến xây dựng một ứng dụng Window Form sử dụng nó. VS.NET cho phép các nhà phát triển tạo ứng dụng Window Form nhanh hơn và hiệu quả hơn.

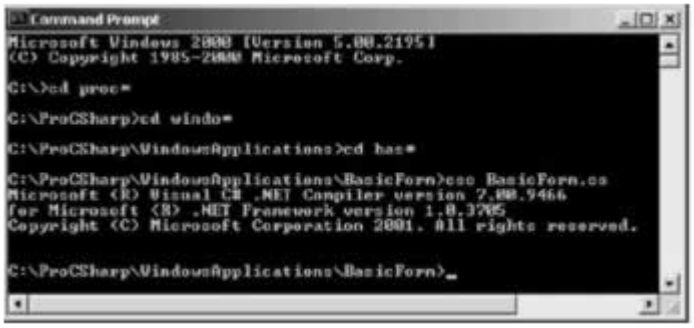
2.2.1 Windows Forms không sử dụng Visual Studio .NET

Hầu hết mọi ứng dụng Window Form sẽ mở rộng lớp *System.Windows.Form* để tuỳ chỉnh và thêm nguyên lý kinh doanh. Vì thế, ứng dụng Windows Form đơn giản nhất sẽ trình bày bên dưới:

```
using System;
using System.Windows.Forms;
namespace WindowsFormsApp
{
  class MyForm : Form
  {
    static void Main(string[] args)
    {
      MyForm aForm = new MyForm();
      Application.Run(aForm);
    }
  }
```

```
}
```

Để xem vấn đề này trong hành động, bạn hãy lưu đoạn mã trên với tên BasicForm.cs, sau đó biên dịch và chạy nó. Khi đó bạn sẽ thấy kết quả như sau:



Khi ứng dụng trên được chạy, một cửa sổ cơ bản sẽ được mở ra. Chú ý rằng cửa sổ hành động giống như một cửa sổ chuẩn và có thể được thu nhỏ, mở to, kéo đi, hay đóng lại. Nó là một ứng dụng Window đầy đủ chức năng trong 13 dòng mã. Hãy xem đoạn mã của nó để hiểu những gì đang xảy ra trước khi ta thấy những điều thú vị hơn sau đây.

```
class MyForm : Form
```

Dòng này chỉ rằng lớp của chúng ta đang thừa hưởng từ lớp *System.Windows.Forms.Form*, có nghĩa là chúng giành được truy cập đến tất cả chức năng của lớp Form cơ bản. Tiếp đến, chú ý rằng trong phương thức *Main()* chúng ta tạo một thể hiện của đối tượng MyForm và chuyển nó đến phương thức *Application.Run()*:

```
static void Main(string[] args)
{
  MyForm aForm = new MyForm();
  Application.Run(aForm);
}
```

Application là một lớp static trong *System.Windows.Forms* namespace, nó chứa các phương thức để bắt đầu và dừng các ứng dụng và các luồng. Phương thức *Run()* có

thể chấp nhận vài tham số; bằng việc truyền vào một đối tượng Form chúng ta đang báo hiệu với .NET Framework bắt đầu xử lý các tin nhắn Window cho form này, và để thoát khỏi ứng dụng khi form này đóng.

2.2.1.1 Các Control

Hãy thêm một control đơn giản *Button* vào form. Chúng ta sẽ thấy các sự kiện bao quát hơn, bây giờ chúng ta chỉ xem xét những gì nó làm để thêm một control vào một ứng dụng Window Form không dùng Visual studio.NET.

Về cơ bản, mọi control trên form là một thành phần dữ liệu của lớp custom *Form*. Vì thế, để thêm một *Button* vào *form*, chúng ta sẽ thêm một thành phần dữ liệu *Button* mới vào lớp *MyForm*. Thêm dòng sau vào tập tin BasicForm.cs:

```
class MyForm : Form
{
  //Data member to hold Button control
  private Button BigButton;
```

Trước khi thành phần dữ liệu này làm bất cứ điều gì hoặc hiển thị một nút trên form nó phải được khởi tạo và các thuộc tính khác nhau của *Button* phải được định hình. Nó nên được thực hiện trong constructor cho đối tượng *MyForm*. Tại thời điểm đó chúng ta sẽ cài các thuộc tính cho chính đối tượng *Form*, như là size và name. Chú ý rằng có nhiều thuộc tính có thể được cài và thực hiện. Vì thế trong constructor là thời điểm tốt nhất để thực hiện khởi tạo giá trị. Thêm khối mã sau vào constructor của *MyForm*:

```
  public MyForm()
  {
    //Set the properties for the Button
    BigButton = new Button();
    BigButton.Location = new System.Drawing.Point(50, 50);
```

```
    BigButton.Name = "BigButton";
    BigButton.Size = new System.Drawing.Size(100, 100);
    BigButton.Text = "Click Me!";

    //Set properties of the Form itself
    ClientSize = new System.Drawing.Size(200, 200);
    Controls.Add(BigButton);
    Text = "My Windows Form!";
  }
```

Đoạn mã này đầu tiên khởi tạo một đối tượng *Button* mới và ấn định nó vào thành phần dữ liệu riêng *BigButton*. Nó sau đó cài các thuộc tính Location, Name, Size, và Text để với các giá trị thích hợp. Bất kỳ thuộc tính nào không cài ở đây sẽ lấy giá trị mặc định.

Những dòng tiếp theo cài kích cở của *form*, và sau đó phương thức *this.Controls.Add()* được gọi để thêm control *Button* vào tập hợp *Controls* của form. Việc này được yêu cầu trước khi nút sẽ được hiển thị trên form. Tập hợp *Controls* sẽ chứa tất cả các control trên một form và có thể cập nhật và sửa đổi tự động trong thời gian chạy để thêm và xoá các control nếu cần. Chúng ta sẽ xem xét cách chúng thực hiện ở phần sau của chương.

Nếu bạn chạy ứng dụng tại điểm này, bạn sẽ thấy một cửa sổ như sau:

Tuy nhiên, không có gì xảy ra khi nút được click. Để thay đổi chúng ta sẽ cần thêm một sự kiện vào đoạn mã.

2.2.1.2 Các sự kiện (event):

Mỗi đối tượng trong một ứng dụng Windows Form có một tập sự kiện. Nếu bạn muốn có một đoạn mã thực hiện một điều gì đó khi các sự kiện xảy ra, bạn nên thêm một bộ điều khiển sự kiện(event handler) vào lớp và kết hợp nó với đối tượng.

Để Windows Forms được sử dụng đoạn mã của bạn, bạn phải truyền cho nó vị trí của phương thức bộ điều khiển sự kiện trong đoạn mã của bạn. Bạn thực hiện bằng cách tạo một thể hiện delegate thích hợp kết hợp với một phương thức trong lớp custom *Form*.

Để thêm vài chức năng cho nút đó, ta cần thêm vài dòng mã vào lớp chúng ta. Thêm phương thức sau vào lớp *Form* của chúng ta. Nó sẽ hành động như bộ điều khiển sự kiện cho sự kiện *Click* của nút. Chú ý rằng bộ điều khiển sự kiện có thể được gọi bất kỳ đối tượng nào. Sự kiện của control tự định nghĩa tham số cho phù hợp với bộ điều khiển.

```
static void Main(string[] args)
{
```

```
    MyForm aForm = new MyForm();
    Application.Run(aForm);
  }

  private void ClickHandler(object sender, System.EventArgs e)
  {
    MessageBox.Show("Clicked!","My Windows Form",MessageBoxButtons.OK);
  }
```

Hầu hết các bộ điều khiển sự kiện Windows Forms có dạng phương thức này. Thông số đầu tiên chứa đối tượng khởi sự kiện. Trong trường hợp này nó sẽ là đối tượng *Button* từ lớp *MyForm*. Thông số tiếp theo chứa dữ liệu về sự kiện trong một thông số *System.EventArgs* hay lớp thừa hưởng. Lớp *System.EventArgs* không chứa dữ liệu- Nó chỉ hành động như một lớp cơ sở. Nếu một sự kiện phải truyền dữ liệu đến client thì nó phải sử dụng một lớp thừa hưởng. Sự kiện *Button.Click* không cần truyền bất kỳ thông tin thêm vào, vì thế nó sử dụng lớp *System.EventArgs* cơ sở.

Cuối cùng, thêm đoạn mã sau vào constructor *MyForm* để sự kiện gắn bộ điều khiển sự kiện của chúng ta vào sự kiện trong lớp *MyForm.*

```
public MyForm()
{
  //Set the properties for the Button
  BigButton = new Button();
  BigButton.Location = new System.Drawing.Point(50, 50);
  BigButton.Name = "BigButton";
  BigButton.Size = new System.Drawing.Size(100, 100);
```

```
  BigButton.Text = "Click Me!";
  BigButton.Click += new EventHandler(ClickHandler);
  //Set properties for the Form itself
  ClientSize = new System.Drawing.Size(200, 200);
  Controls.Add(BigButton);
  Text = "My Windows Form!";
}
```

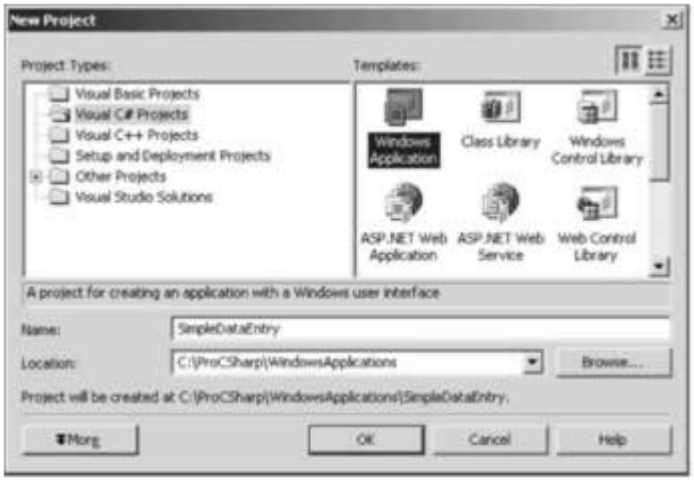
Ví dụ này trình bày cách Windows Form sử dụng delegates để wrap một phương thức của đối tượng trước khi ấn định nó vào sự kiện chúng ta muốn vận dụng. *System.EventHandler* delegate được sử dụng để tham khảo phương thức *ClickHandler()* và nó được liên kết với sự kiện *Click* của nút bằng cách thêm nó vào bộ điều khiển sự kiện *Click*. Chú ý cú pháp sử dụng - có nghĩa là các bộ điều khiển sự kiện thêm vào có thể được liên kết với một sự kiện đơn. Chúng sẽ được xử lý để chúng được thêm vào bộ điều khiển sự kiện.

Biên dịch ứng dụng lại, và chạy nó. Lúc này khi click nút bạn sẽ thấy một hộp tin nhắn nhỏ.

2.2.2 Windows Form sử dụng Visual Studio .NET

Giống như trong .NET, sử dụng Visual studio.NET tạo các ứng dụng Windows Form đơn giản hơn nhiều. Visual studio.NET giảm số lượng mã rắc rối mà các nhà phát triển phải viết, cho phép các nhà phát triển tập trung vào giải quyết các vấn đề kinh doanh.

Hãy xem cách tạo một ứng dụng Window Forms đơn giản sử dụng Visual studio.NET. Chúng ta sẽ tạo một màn hình thực thể dữ liệu đơn giản cho một hệ thống quản lý thông tin cá nhân hư cấu. Loại màn hình này sẽ được gắn vào một số form của cơ sở dữ liệu sử dụng để lưu trữ dữ liệu cá nhân. Chúng ta xem xét cách để tạo một tầng giao diện người dùng trong chương này. Tạo một dự án Windows Application C# mới trong Visual studio.NET với tựa đề là *SimpleDataEntry*.



Sau khi dự án được tạo, bạn sẽ thấy một form đơn giản trong Visual Studio.NET trong màn hình thiết kế. Màn hình thiết kế được dùng để thêm control vào form. Click phải trên tập tin *Form1.cs* trong Solution Explorer và chọn *View Code*. Nó sẽ hiển thị mã phát ra bởi form được hiển thị trong màn hình thiết kế. Nhìn qua đoạn mã này. Với việc thêm vào của một vài tiêu chuẩn, mà Visual studio.NET biên dịch như là phương thức *InitalizeComponent()*, đoạn mã nhìn rất giống với ứng dụng Windows Forms ban đầu. Chú ý cách dùng của *Application.Run* trong phương thức *Main*, và sự thật là lớp *Form* này thừa hưởng từ *System.Windows.Forms.Form*.

Phương thức I*nitializeComponent()* được dùng bởi Visual studio.NET để xây dựng *Form* tại thời gian chạy. Tất cả control và thuộc tính mà một nhà phát triển cài suốt thời gian thiết kế được cài tại thời gian chạy trong phương thức này. Khi có những sự thay đổi được tạo ra cho *Form* trong thời gian thiết kế, Visual Studio.NET sẽ cập nhật phương thức này.

Quay lại màn hình thiết kế để thêm vài control vào *form* này để làm cho nó hữu dụng và thú vị hơn. Chú ý rằng khi bạn chọn *form*, cửa sổ *properties* cài đặt các thuộc tính khác nhau của các control trong ứng dụng Windows Forms của chúng ta. Nó là một bộ phận quan trọng của Visual studio.NET IDE, khi sử dụng nó thì dễ tìm kiếm tên của mọi thuộc tính, của mỗi control trong tài liệu .NET hơn. Có một số nút ở tại đầu của cửa sổ này. Hai thay đổi đầu tiên là cách mà các thuộc tính được hiển thị. Nhóm đầu tiên hiển thị các mục chọn trong phạm trù luận lý, như là tất cả thuộc tính với hình thức, cách cư xử, thiết kế và vân vân. Nút thứ hai sắp xếp tất cả thuộc tính theo thứ tự alphabe. Hai nút kế tiếp chốt vào giữa sự hiển thị thuộc tính hoặc các sự kiện. Chúng ta sẽ bàn luận những sự kiện và cách thêm chúng vào các control tiếp đó. Nút cuối cùng mở trang thuộc tính của dự án này:

Cài các thuộc tính sau của *form* bằng cách sửa đổi chúng trực tiếp trong cửa sổ *Properties*:

Property	**Value**
Text	Data Entry Form
Size	300, 220
(Name)	frmMain
StartPosition	CenterScreen

Các cài đặt này sẽ tạo một cửa sổ 300 tới 220 pixel ở giữa màn hình. Thuộc tính Name là một thuộc tính quan trọng trên tất cả các controls. Giá trị này được dùng như tên đối tượng của các biến thành viên của lớp, và được dùng để tham khảo đế control trong đoạn mã.

Bây giờ thêm hai control Button vào form. Cài các thuộc tính của hai control Button như sau:

Property	**button1 Value**	**button2 Value**
(Name)	btnSave	btnCancel
Location	125, 157	210, 157
Size	78, 25	78, 25
Text	Save	Cancel

Ở đây chúng ta đang thay đổi các tên mặc định của *Button* đến một giản đồ đặt tên chuẩn hơn, và định vị chúng vào vị trí chúng ta muốn chúng trên *Form1*.

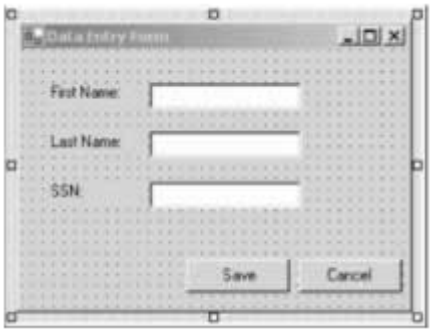
Quay về màn hình mã để xem xét những gì Visual studio.NET đã làm suốt thời gian này. Bạn sẽ thấy phần thêm của hai biến thành viên mới trong lớp *Form*. Nếu bạn

mở rộng vùng tiêu đề "Windows Form Designer generated code" bạn sẽ thấy phương thức *InitializeComponent(),* nơi mà tất cả control trên form được khởi tạo và định hình chính xác. Phương thức này được gọi trong constructor của form.

Tiếp đó thêm ba control *TextBox* và ba Control *Label Next* vào *Form*. Gán các thuộc tính như sau:

Property	**TextBox1**	**TextBox2**	**TextBox3**	**Label1**	**Label2**	**Label3**
(Name)	txtFName	txtLName	txtSSN	label1	label2	label3
Location	97, 25	97, 61	97, 99	20, 25	20, 62	20, 99
Size	115, 20	115, 20	115, 20	70, 18	70, 18	70, 18
Text	*(Blank)*	*(Blank)*	*(Blank)*	First Name:	Last Name:	SSN:

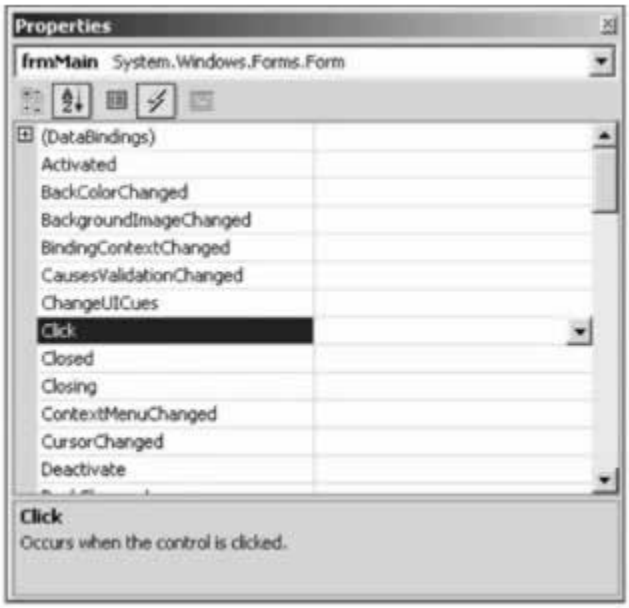
Bạn nên có một *Form* giống một màn hình thực thể dữ liệu về thông tin người dùng. Một end-user có thể sử dụng màn hình này để nhập tên đầu và cuối của chúng như Social Security Number. Tại lúc này *Form1* sẽ giống như sau:



2.2.2.1 Các sự kiện (event):

Các ứng dụng Windows là event-driven và không thêm mã, nó đáp ứng các sự kiện. Visual studio.NET tạo ra nó rất đơn giản bằng cách thêm mã đáp ứng các sự kiện phát sinh bởi ngươi dùng và hệ thống.

Cửa sổ *Properties* được cập nhật để phản ánh toàn bộ danh sách sự kiện có thể được điều khiển từ đối tượng này. Để thấy danh sách này, chọn nút thứ tư từ bên trái. Nó sẽ hiển thị một danh sách sự kiện cho đối tượng đang chọn, và chọn bất kỳ đối tượng nào nó sẽ có liên kết mã với chúng. Màn hình bên dưới chỉ dãy sự kiện khi đối *Form* được chọn.



Việc thêm một sự kiện có thể được vận dụng một trong hai cách. Để thêm một sự kiện mặc định cho một control bằng cách click đôi lên nó trong màn hình thiết kế.

Cách khác để thêm các bộ điều khiển sự kiện vào đoạn mã của bạn và các tuỳ chọn nếu bạn không thêm các sự kiện mặc định, bằng cách sử dụng cửa sổ *Properties*. Một lần nữa, đoạn mã pluming đúng sẽ được chèn vào lớp Form và bạn sẽ lấy các bộ điều khiển sự kiện cho sự kiện được chọn.

Chú ý cửa sổ *Properties*, bạn phải làm như vậy nếu bạn có nhiều nút mà tất cả chúng cùng làm những việc giống nhau và yêu cầu cùng một sự xử lý.

Hãy thêm vài đoạn mã trong bộ điều khiển sự kiện của hai control *Button* của chúng ta. Thêm bộ điều khiển sự kiện *Click* vào hai *Buttons* đang tồn tại. Thêm đoạn mã sau vào tập tin:

```
private void btnSave_Click(object sender, System.EventArgs e)
{
  SaveFile();
}
private void btnCancel_Click(object sender, System.EventArgs e)
{
  Clear();
}
private void SaveFile()
{
  //Save the values to an XML file
  //Could save to data source, Message Queue, etc.
  System.Xml.XmlDocument aDOM = new System.Xml.XmlDocument();
  System.Xml.XmlAttribute aAttribute;

  aDOM.LoadXml("<UserData/>");

  //Add the First Name attribute to XML
  aAttribute = aDOM.CreateAttribute("FirstName");
  aAttribute.Value = txtFName.Text;
  aDOM.DocumentElement.Attributes.Append(aAttribute);

  //Add the Last Name attribute to XML
```

```
    aAttribute = aDOM.CreateAttribute("LastName");
    aAttribute.Value = txtLName.Text;
    aDOM.DocumentElement.Attributes.Append(aAttribute);

    //Add the SSN attribute to XML
    aAttribute = aDOM.CreateAttribute("SSN");
    aAttribute.Value = txtSSN.Text;
    aDOM.DocumentElement.Attributes.Append(aAttribute);

    //Save file to the file system
    aDOM.Save("UserData.xml");
  }

  private void Clear()
  {
    //Erase all the text
    txtFName.Text = "";
    txtLName.Text = "";
    txtSSN.Text = "";
  }
```

Ví dụ đơn giản này lưu dữ liệu được nhập bởi một tập tin XML trên hệ thống tập tin. Mọi ứng dụng sẽ sử dụng ADO.NET để lưu thông tin vào một nguồn dữ liệu back-end. Tuy nhiên, trong ví dụ này chúng ta sẽ xuất khẩu một tập tin XML nhỏ.

Chúng ta sử dụng các phương thức *private* để thể hiện chức năng thực sự, vì thế chúng ta có thể sử dụng cùng chức năng từ các tuỳ chọn menu sau này. Mọi đoạn mã trong phương thức *SaveFile()* bao gồm việc viết ra tập tin XML chứa dữ liệu user-supplied. Sự kiện *Click* của nút *Cancel* gọi phương thức *Clear()* để xoá tất cả các control *textbox*. Chú ý rằng trong một ứng dụng hoàn chỉnh, nó có thể đóng cửa sổ này và trả về người dùng một màn hình chính.

Chú ý rằng có một lỗi trong Visual studio.NET thỉnh thoảng yêu cầu một nhà phát triển thay đổi tên của lớp Form sử dụng trong phương thức *Main()* bằng tay. Nếu bạn có một lỗi khi biên dịch phương thức *Main()* và bảo đảm nó giống như đoạn mã sau. Phải bảo đảm rằng đoạn mã tạo đối tượng sử dụng tên lớp *frmMain*. Khi một tên lớp *Form* bị thay đổi thì dòng này không luôn luôn cập nhật.

```
static void Main()
{
  Application.Run(new frmMain());
}
```

Nếu bạn chạy ứng dụng này tại lúc này bạn sẽ có một cửa sổ thực thể dữ liệu nhỏ có đầy đủ chức năng. Bạn có thể nhập dữ liệu, lưu nó vào một tập tin XML, và xoá tất cả giá trị. Việc đó thì đơn giản nhưng nó biểu lộ cách tạo các ứng dụng sử dụng Visual studio.NET.

2.2.2.2 Resizing Windows

Một vấn đề với cửa sổ thực thể dữ liệu của chúng ta là khi nó được thay đổi kích thước thì các control bị khoá lại trong một vùng. Điều đó có vẽ buồn cười và không chuyên nghiệp với một ứng dụng cao cấp,do đó nên hỗ trợ khả năng thay đổi kích thước lại và định vị một cửa sổ trong bất kỳ hình dạng nào người dùng mong muốn. Bất kỳ nhà phát triển nào viết mã để điều khiển việc thay đổi kích thước và thay thế của các control sẽ đánh giá sự dễ dàng khi sử dụng .NET Framework và Window Forms để làm việc này.

Với một thuộc tính đơn thì tất cả công việc này có thể được điều khiển bởi .NET Framework.

Thuộc tính *Anchor* thể hiện năng lực kỳ diệu này, và nó là một thành viên của hầu hết tất cả các lớp trong *System.Windows.Forms* namespace bởi vì nó là một thuộc tính của lớp *System.Windows.Forms.Control*. Nhắc lại là, mọi control đều thừa hưởng từ lớp này.

Thuộc tính *Anchor* được cài một liên kết cuả một hay nhiều cạnh của cha mẹ nó. Cài một trong những cạnh này trong thuộc tính sẽ dẫn đến control duy trì mối quan hệ vị trí giữa cạnh của nó và cạnh của cha mẹ nó khi *form* được thay đổi kích thước và di chuyển. Thuộc tính này rất quan trọng để thiết kế giao diện người dùng thân thiện, và nên được thí nghiệm để hiểu cách nó làm việc.

Visual Studio .NET bao gồm một cửa sổ pop-up để cài thuộc tính này vào đúng mối liên kết. Cửa sổ pop-up này cho phép một nhà phát triển chọn cạnh để neo control. Cửa sổ pop-up này có thể được tìm thấy như một phần của cửa sổ *Properties*.

Chúng ta sẽ dùng thuộc tính *Anchor* để tạo một giao diện người dùng hiệu quả hơn cho màn hình thực thể dữ liệu của chúng ta.

Chọn các control *TextBox* trong môi trường thiết kế Visual studio.NET. Thay đổi thuộc tính *Anchor* vào Top, Left, Right sử dụng cửa sổ pop-up. Nó sẽ duy trì khoảng cách giữa top, left, và right của các cạnh của cha mẹ, bằng cách đó thay đổi kích thước control chính xác.

Chọn các control *Button* thứ hai và thay đổi thuộc tính *Anchor* của nó ở Bottom, Right. Nó sẽ duy trì vị trí đóng của chúng ở bottom-right của form. Chạy ứng dụng và thay đổi kích thước cửa sổ để thấy cách các control điều chỉnh chính bản thân chúng.

Một lần nữa, thuộc tính này được là quyết định hoàn toàn trong thiết kế giao diện người dùng chuyên nghiệp trong .NET, và sử dụng nó giảm số lượng của công việc yêu cầu bởi các nhà phát triển. Các nhà phát triển tự do này tập trung giải quyết vấn đề kinh doanh thực tế để thay cho việc thay đổi kích thước cấp thấp.

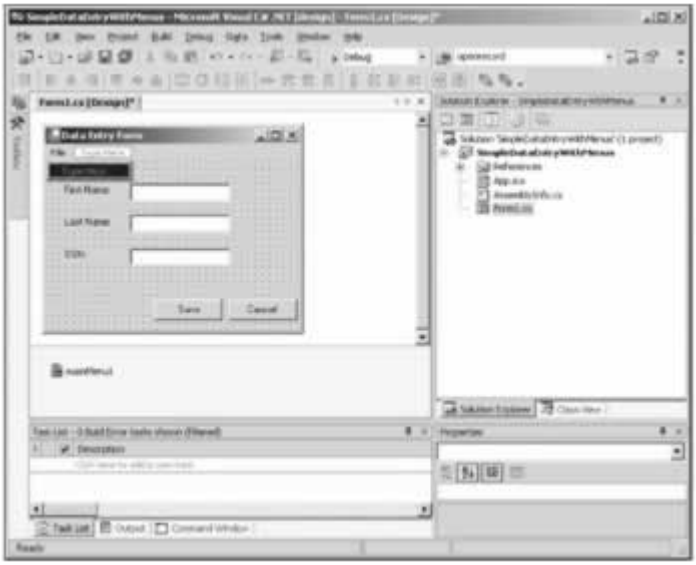
2.2.3 Menus

Các menu được dùng trong hầu hết mọi ứng dụng Window, và chúng cung cấp một cách tuyệt vời để giao tiếp người dùng với các tuỳ chọn để họ làm việc theo các chức năng có sẵn. Có hai kiểu menu khác nhau. Thông thường nhất là một menu chính(main menu), ở đầu của một cửa sổ và thường bao gồm các mục như File, Edit, và Help. Vài ứng dụng chứa các menu theo ngữ cảnh để cho phép người dùng truy cập đến thông tin về các chủ đề hay mục đặc biệt. Menu theo ngữ cảnh được ẩn cho đến khi người dùng nhấn chuột phải - sau đó menu được hiển thị tại vị trí con trỏ.

Windows Forms cung cấp hỗ trợ đầy đủ cho việc thêm hai kiểu menu vào một ứng dụng. Lớp *System.Windows.Forms.Menu* cung cấp lớp cơ sở cho tất cả lớp menu trong hệ thống. Lớp *MainMenu* tượng trưng cho menu chính, và có thể liên kết với một form. Menu này chứa một tập hợp đối tượng *MenuItem* tượng trưng cho một tuỳ chọn menu riêng rẽ.

Lớp *ContextMenu* thì có thể thêm các menu theo ngữ cảnh cho một ứng dụng. Lớp này cũng chứa một tập hợp đối tượng *MenuItem*, nhưng ContextMenu có thể xuất hiện trong bất kỳ vị trí nào trong một form, nó không chỉ tại đầu của một cửa sổ như lớp *MainMenu*.

Chúng ta sẽ thêm một menu vào ứng dụng thực thể dữ liệu của chúng ta. Thêm một menu vào một ứng dụng Window Form thì dễ như thêm bất kỳ control chuẩn nào như là một *Button* hay *TextBox*. Chọn control *MainMenu* từ thanh công cụ và vẽ một hộp trên bề mặt thiết kế. Nó sẽ thêm một menu tại đầu của form. Chọn menu và gõ File để thêm mục menu đầu tiên. Bây giờ khi bạn click trên File một menu mới sẽ hiển thị bên dưới, nó có thể thêm vào như chúng ta thêm mục menu File. Bạn có thể tiếp tục gõ vào MenuItem, bằng cách đó tạo ra cấu trúc thực sự của hệ thống menu trong IDE



Sử dụng hệ thống menu để tạo menu sau. Chú ý rằng: bằng cách nhập một ký tự gạch(–) đơn lẽ thì một dòng riêng lẽ được tạo. Nó rất hữu ích cho việc phân chia các nhóm chọn lựa trong một menu. Một phần quan trọng khác để nhớ là bằng cách mở đầu một ký tự với ký hiệu là (&) thì ký tự đó trở thành phím tắt cho mục menu này. Vì thế một người dùng có thể chọn menu bằng cách chỉ sử dụng bàn phím.

Top Level Menu Item	**Contained Menu Items**
Text – &File Name – mnuFile	Text – &Save Name – mnuSave
	Text – &Cancel Name – mnuCancel
	Text – "-" (Single Dash) Text – E&xit Name – mnuExit

Top Level Menu Item	Contained Menu Items
Text – &Color Name – mnuColor	Text – &Gray Name – mnuGray RadioCheck – true Checked – true
	Text – G&reen Name – mnuGreen RadioCheck – true
	Text – &Blue Name – mnuBlue RadioCheck – true
	Text – &Red Name – mnuRed RadioCheck – true
	Text – &Purple Name – mnuPurple RadioCheck – true

Chạy ứng dụng và thấy rằng bạn có một cửa sổ với một menu đang làm việc trên đó. Tuy nhiên không có gì xảy ra khi một menu được chọn. Để thay đổi, bộ diều khiển sự kiện này phải được thêm bên dưới những mục menu riêng lẽ. Chúng ta sẽ tiếp tục với cùng ví dụ này và thêm sự kiện điều khiển để người dùng có thể sử dụng menu.

Các *MenuItems* riêng rẽ là mọi control giống như các cái khác, và chúng có thể được chọn trong bề mặt thiết kế. Làm các việc này bằng cách chỉ ra các thuộc tính và sự kiện của chúng trong cửa sổ *Properties*. Sử dụng danh sách sự kiện để thêm bộ điều khiển sự kiện *Click* cho các mục chọn Save, Cancel, và Exit. Thêm đoạn mã sau trong bộ điều khiển sự kiện mới:

```
private void mnuSave_Click(object sender, System.EventArgs e)
{
  SaveFile();
}
private void mnuCancel_Click(object sender, System.EventArgs e)
{
  Clear();
}
private void mnuExit_Click(object sender, System.EventArgs e)
{
  Close();
}
```

2.2.3.1 Dynamic Menus

Các menu thường được dùng để phản ánh trạng thái của ứng dụng. Khi người dùng tạo các chọn lựa và thay đổi trong ứng dụng, menu phải phản ánh các sự thay đổi này. Các mục menu có thể được thêm, xoá và chỉnh sửa để phản ánh tình trạng ứng dụng hiện tại. Một lần nữa, các *MenuItem* hành động như các thành phần khác và có thể được vận dụng.

MenuItems có thể có một nút kiểm kế bên để minh hoạ tuỳ chọn hiện tại. Nó rất hữu dụng cho người dùng để họ có thể đánh giá chính tình trạng của ứng dụng của họ. Thuộc tính *Checked* như một biến cờ, nó có thể cài để hiện hay dấu một điểm kiểm tra kế bên mục menu. Nếu thuộc tính *RadioCheck* được cài bằng true thì nút kiểm sẽ xuất hiện như một chấm đơn giản. Vì thế chỉ một mục menu đơn giản được chọn tại một thời điểm với thuộc tính *RadioCheck*.

Chúng ta đang thêm một số mã bên dưới các mục menu *color* để thay đổi màu nền của form. Chúng ta sẽ thực hiện bằng cách sử dụng một bộ điều khiển sự kiện cho mọi đối tượng *MenuItem*.

Trong ứng dụng của chúng ta, thêm phương thức sau vào:

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


}
```

Chúng ta thêm một bộ điều khiển sự kiện ở đây để thay cho việc cho phép Visual Studio.NET IDE làm giùm chúng ta. Chúng ta cần làm như vậy để chúng ta có thể liên kết phương thức đơn này với mọi bộ điều khiển sự kiện *Click* của các mục menu. Nó sẽ cho phép chúng ta điều khiển tình trạng của menu và ứng dụng từ phương thức đơn này.

Trở lại với màn hình thiết kế của IDE, click trên mục menu *Gray*. Trong cửa sổ *Properties* chuyển tới màn hình sự kiện và chọn sự kiện *Click*. Click trên mũi tên thả xuống để hiển thị một danh sách tên phương thức có thể liên kết với sự kiện này. Đây là cách để gắn các phương thức vào các sự kiện. Chọn phương thức *mnuItems_Click()* từ danh sách. Lập lại thủ tục này với mọi mục trong menu *Color*. Sự kiện *Click* của mọi mục nên được liên kết với cùng phương thức.

Bây giờ, mọi đối tượng được liên kết với cùng phương thức bộ điều khiển sự kiện, thêm đoạn mã sau để cập nhật *BackColor* của form và tình trạng menu.

```
private void mnuItems_Click(object sender, System.EventArgs e)
{
  MenuItem aObj;
  //Set the BackColor of the form based on the selected object
  if(sender == mnuGray)
    this.BackColor = System.Drawing.SystemColors.Control;
  else if(sender == mnuGreen)
    this.BackColor = Color.Green;
  else if(sender == mnuBlue)
    this.BackColor = Color.Blue;
  else if(sender == mnuRed)
    this.BackColor = Color.Red;
  else if(sender == mnuPurple)
    this.BackColor = Color.Purple;

  //Set all checkboxes to false
  mnuGray.Checked = false;
  mnuGreen.Checked = false;
  mnuBlue.Checked = false;
  mnuRed.Checked = false;
  mnuPurple.Checked = false;
```

```
    //Change the selected item to checked
    aObj = (MenuItem)sender;
    aObj.Checked = true;
}
```

Đoạn mã này sử dụng sự kiện là tham số *sender* trong một bộ điều khiển sự kiện là đối tượng để khởi sự kiện. Điều này được yêu cầu bởi vì bộ điều khiển sự kiện này được dùng bởi tất cả đối tượng *MenuItem*. Vì thế, bước đầu là xác định mục menu được chọn bởi người dùng. Và sau đó thay đổi *BackColor* của form theo màu sắc tương ứng.

Bước kế tiếp là cài một nút kiểm kế bên mục menu thích hợp. Chúng ta có thể thực hiện việc này bằng cách cài đặt đơn giản thuộc tính *Checked* bằng true, sau đó đặt nó vào đối tượng *MenuItem*. Tuy nhiên, trước khi chúng ta làm việc này, chúng ta cần cài tất cả đối tượng *MenuItem* là unchecked.

Chạy ứng dụng và chọn các tuỳ chọn màu sắc khác nhau. Bạn sẽ thấy màu nền của của sổ thay đổi, và hộp kiểm trong menu cập nhật để phản ánh màu sắc hiện tại

2.2.3.2 Menus ngữ cảnh

Mọi ứng dụng Window cho phép người dùng click phải và hiện lên một menu theo ngữ cảnh. Nó có nghĩa là các sự chọn lựa menu được dựa vào đối tượng, hay ngữ cảnh , người dùng đã chọn. Các menu ngữ cảnh cho phép ứng dụng biểu thị thông tin thêm vào hay các chọn lựa người dùng.

Các menu theo ngữ cảnh có thể được thêm vào các ứng dụng Windows Forms rất dễ dàng. Chúng ta sẽ thêm một menu theo ngữ cảnh vào cửa sổ thực thể dữ liệu của chúng ta, các chọn lựa *Save* và *Cancel* khi người dùng click phải ở bất kỳ đâu trên Form.

Để thêm một menu ngữ cảnh vào một form, đơn giản thêm control *ContextMenu* từ thanh công cụ vào *Form1*. Khi đối tượng *ContextMenu* được thêm vào form nó sẽ xuất hiện trong vùng footer bên dưới bề mặt thiết kế form. Khi biểu tượng này được chọn, menu chính, nếu nó tồn tại sẽ không xuất hiện trong form và được thay thế với chính menu ngữ cảnh đó. Nó có thể được chỉnh sửa trong bề mặt thiết kế bằng cách gõ các mục menu khác nhau, như là sửa các menu chính. Mặc dù nó xuất hiện, các menu sẽ được hiển thị ở đỉnh của form như menu chính, nó sẽ được ẩn cho đến khi chúng ta gán nó vào form.

Sau khi thêm *ContextMenu* vào *Form1*, thêm các mục menu dưới đay bằng cách gõ vào các giá trị sau:

Menu Item Name	**Text Property Value**
mnuSaveContext	Save
mnuCancelContext	Cancel

Một lần nữa, mọi mục menu là các đối tượng *MenuItem* riêng lẽ và có các thuộc tính có thể cài trong cửa sổ *Properties* và chọn sự kiện *Click*. Trong dãy thả xuống chọn *mnuSave_Click* cho *mnuSaveContext MenuItem* và *mnuCancel_Click* cho *mnuCancelContext MenuItem*. Nó sẽ nối những sự kiện này với cùng bộ điều khiển sự kiện được gọi khi các mục menu chính được click.

Bây giờ chúng ta có một menu ngữ cảnh, nó xuất hiện khi chúng ta click phải trên *Form*. Để thêm một menu ngữ cảnh vào một *Form*, thuộc tính *ContextMenu* của đối tượng *Form* phải được cài vào đối tượng *ContextMenu* của chúng ta. Khi nó được cài, form sẽ tự động hiển thị *ContextMenu* khi ngươi dùng click phải. Nó sẽ hiển thị *ContextMenu* tại vị trí người dùng click phải. Chú ý rằng thuộc tính này có thể được cập

nhật trong thời gian chạy. Thật là quan trọng để ghi chú rằng thuộc tính này là một thành viên của lớp Control, có nghĩa là tất cả control Windows Forms đều có thuộc tính này.

Cài thuộc tính *ContextMenu* của *Form1* vào *contextMenu1* sử dụng cửa sổ *properties*. Một combo box sẽ hiển thị các đối tượng *ContextMenu* hiện tại để chọn từ trên form. Multiple *ContextMenus* có thể được thêm vào một form, mặc dù chỉ một được gán vào form tại một thời điểm.

Chạy ứng dụng và click phải bất kỳ đâu trên trên form để thấy menu ngữ cảnh hiển thị hai tuỳ chọn *Save* và *Cancel*.

2.2.4 Dialogs

Dialogs là một kiểu đặc biệt của Form để lấy thông tin người dùng và tương tác với ngươi dùng trong các ứng dụng Window. Có một tập các hộp dialog định nghĩa trước để lấy thông tin như vị trí tập tin, màu sắc, và cài đặt máy in. Một ứng dụng tuỳ biến thường sử dụng hộp thoại dialog để thuận tiện chọn dữ liệu từ endusers.

Tạo một hộp dialog thì rất giống với tạo một Form chuẩn. Trên thực tế, cùng tiến trình xử lý được dùng để thêm vào một dự án Visual studio.NET. Sự khác chính là bản liệt kê *FormBorderStyle*, nó phải được cài là *Fixel Dialog*. Nó tạo cửa sổ không lớn và là nguyên nhân nó giống với hộp dialog Window. Nó cũng là một thực hành Window chuẩn

để huỷ *ControlBox*, *MinimizeBox*, và *MaximizeBox* từ một hộp dialog, vì thế các thuộc tính này nên được cài là false trong cửa sổ *properties*.

Bất kỳ control Windows Forms chuẩn nào cũng có thể tồn tại trên một hộp dialog. Bề mặt thiết kế trong Visual studio.NET được dùng để thiết kế các *Form* chuẩn, và các tuỳ chọn giống nhau có thể dùng cho các nhà phát triển.

2.2.4.1 Modal vs. Modeless

Khi chúng ta muốn hiển thị hộp dialog, có hai chọn lựa: modal hay modeless. Hai khái niệm này chỉ cách dialog tương tác với ứng dụng. Một modal dialog ngăn chận các luồng hiện tại và yêu cầu người dùng trả lời vào hộp dialog trứoc khi tiếp tục với ứng dụng. Một Modeless dialog thì giống một cửa sổ chuẩn hơn.

2.2.4.2 Dialog Box Results

Thường rất quan trọng để hiểu cách người dùng đóng một hộp dialog. Một ví dụ điển hình đó là một dialog *File Open*. Nếu người dùng chọn một tập tin thì hành động tiếp theo cho ứng dụng là load tập tin đó, tuy nhiên nếu người dùng click nút *Cancel* hay đóng hộp dialog thì ứng dụng sẽ không load bất kỳ tập tin nào.

Bí quyết để hiểu cách người dùng tương tác với một hộp dialog là bảng liệt kê *DialogResult*. Các giá trị cho bảng liệt kê này như sau:

Value	**Description**
Abort	Giá trị này thì được trả về khi một người dùng chọn một nút có nhãn Abort. Trong trường hợp này, người dùng muốn huỹ thao tác hiện tại và không lưu sự thay đổi.
Cancel	Giá trị này thường được trả về khi một người dùng chọn một nút có nhãnlà Cancel, đóng hộp thoại bằng cách nhấn nút "x", hay nấn phím Esc. Người

Value	Description
	dùng muốn huỹ các thay đổi và trả về trạng thái trước khi mở hộp thoại.
Ignore	Giá trị này được trả về khi một người dùng chọn nút có nhãn Ignore. Giá trị này có thể được sử dụng khi ứng dụng cảnh báo người dùng về các điều kiện xảy ra lỗi, nhưng người dùng chọn lệnh Ignore.
No	Giá trị này thường được trả về khi một người dùng chọn nút có nhãn No. Nó thì bình thường khi hộp thoại được dùng để hỏi người dùng một câu hỏi es/no.
Yes	Giá trị này thường được trả về khi một người dùng chọn nút có nhãn Yes. Nó là con trỏ đếm đến kết quả trả về Nó và được dùng trong cùng tình huống.
None	Không có gì được trả về từ hộp thoại.
OK	Giá trị này đươc trả về khi một người dùng chọn nút có nhãn OK. Nõ thì bình thường trong các hộp thoại và các tin cảnh báo nơi nào quan trong cho người dùng thừa nhận thông tin.
Retry	Giá trị này được trả về khi một người dùng chọn nút có nhãn Retry. Nó có ích khi một thao tác không thành công sẽ thành công nếu được thử lại.

Để truy cập vào giá trị này bạn phải sử dụng thuộc tính *DialogResult* của Form. Thuộc tính này là public và có thể truy cập ngay khi người dùng đã đóng hộp dialog.

2.2.4.3 Mở một Dialog

Có hai cách để mở một hộp dialog, một là hiển thị modal và một là hiển thị modeless. Để hiện một dialog như một modal dialog, thì sử dụng phương thức sau:

```
DialogResult Form.ShowDialog()
```

Phương thức này là một bộ phận của lớp Form. Phương thức này có thể chấp nhận không có tham số hoặc một đối tượng Form như một tham số. Đối tượng Form này đại diện cho ower của hộp dialog, hộp này có ích bởi vì tất cả đối tượng Form có một con trỏ quay về cha mẹ của chúng, cho phép một hộp dialog có thể lấy hoặc cài dữ liệu vào cha mẹ của nó. Còn nếu không tham số thì truyền các mặc định cửa sổ hiện tại vào cha mẹ nó.

Chú ý phương thức này trả về một giá trị bảng liệt kê *DialogResult*. Phương thức này ngăn chặn sự thực thi, và không có mã nào sau khi nó thực thi cho đến khi người dùng đóng hộp dialog. Khi sự việc này xảy ra thì *DialogResult* mã được trả về và ứng dụng có thể tiếp tục xử lý. Đoạn mã sẽ như sau:

```
if (aDialogObject.ShowDialog() == DialogResult.Yes)
{
  //User selected Yes
  //Use the properties of the aDialogObject to perform actions
}
else
{
  //User selected No – do not perform action
}
```

Đoạn mã trên hiển thị hộp dialog để hỏi người dùng nếu họ muốn lưu tập tin hiện tại. Nếu người dùng chọn Yes thì đoạn mã sẽ thả vào khối *if* nếu chọn No thì khối *else* sẽ được thực thi.

2.2.4.4 Common Dialogs

.NET Framework cung cấp truy cập đến những common dialog này thông qua các lớp sau. Mỗi lớp này tượng trưng một common dialog và có thể được hiển thị như một hộp dialog. Tất cả lớp này tồn tại trong *System.Windows.Forms* namespace:

Class	**Description**
ColorDialog	Nó cho phép một ngươi dùng chọn một màu từ bảng màu.
FontDialog	Hộp dialog này hiển thị tất cả font hiện có trên hệ thống và cho phép người dùng chọn một cái để dùng trong ứng dụng.
OpenFileDialog	Cho phép một người dùng mở một tập tin sử dụng hộp thoại mở tập tin chuẩn.
SaveFileDialog	Nó cho phép người dùng chọn một tập tin, thư mục hay địa chỉ mạng để lưu dữ liệu của ứng dụng.
PageSetupDialog	Nó cho phép người dùng cài kích cỡ trang, canh lề, và các đặc tính in ấn khác.
PrintDialog	Nó cho phép người dùng cài định dạng trang hiện hành và các đặc tính in ấn qua hộp dialog thuộc tính in ấn chuẩn.
PrintPreviewDialog	Nó hiển thị một tài liệu như nó xuất hiệnk trên máy in đang chọn với các cài đặt trang hiện hành.

Tất cả lớp này thừa kế từ lớp *System.Windows.Forms.CommonDialog*, ngoại trừ lớp *PrintPreviewDialog*. Lớp *System.Windows.Forms.CommonDialog* cung cấp các chức năng cơ bản yêu cầu hiện một hộp combox dialog. Mọi lớp common dialog được hiển thị

sử dụng phương thức *ShowDialog()*, nhưng chúng chứa các thuộc tính tuỳ biến sử dụng để định hình và hỏi chức năng tuỳ biến của chúng.

ColorDialog

Dialog này hiển thị hộp dialog màu sắc chung. Nó hữu ích khi một ngươi dùng được cho phép định dạng nền của một Form hay control, và bạn muốn cung cấp chúng như một cách để chọn màu ưu tiên.

Thuộc tính chính được dùng với lớp này là thuộc tính *Color*. Nó chứa màu sắc chọn lựa khi hộp dialog trả điều klhiển cho ứng dụng. Thuộc tính này là một cấu trúc màu, chúng có lợi bởi vì .NET framework cần cấu trúc này trong các phương thức và thuộc tính khác.

Một tiện lợi khác là nó cho phép người dùng định nghĩa và sử dụng một tập màu sắc custom-defined. Đặc trưng này có thể hiện bằng mặc định, nhưng có thể bị ẩn bằng cách cài thuộc tính *AllowFullOpen* bằng giá trị false.

Như các hộp dialog khác, giá trị trả về từ *ShowDialog()* phải được xem xét để hiểu cách người dùng thoát khỏi dialog. Đây là một đoạn mã nhỏ sử dụng lớp này:

```
ColorDialog aClrDialog = new ColorDialog();
aClrDialog.AllowFullOpen = false;
aClrDialog.Color = this.BackColor;
if (aClrDialog.ShowDialog() == DialogResult.OK)
{
   this.BackColor = aClrDialog.Color;
}
aClrDialog.Dispose();
```

Chú ý rằng bạn có thể sử dụng bất kỳ hộp dialog nào như ví dụ này, hay bạn có thể thêm thành phần vào một Form trong Visual studio.NEY. Đoạn mã sẽ rất giống, nhưng lớp dialog sẽ có giá trị cho mọi phương thức trong *Form*, và tất cả thuộc tính này có thể được cài tại thời gian thiết kế trong cửa sổ *Properties*.

FontDialog

Lớp này cho phép một người dùng chọn một kiểu font, size, và color. Nó rất hữu ích trong các ứng dụng, nhưng nó cũng có thể được dùng để cho phép người dùng định dạng bằng cách chọn kiểu font để hiển thị trong nhập liệu và màn hình báo cáo.

Lớp này chứa một số lượng lớn các thuộc tính mà có thể được cài để định hình các chức năng của hộp dialog :

Property	**Description**
Color	Thuộc tính này lấy hay cài màu sắc font được chọn. Chú ý thuộc tính ShowColor phải là true để nó hợp lệ.
Font	Đây là thuộc tính quan trọng nhất củ hộp dialog này. Nó trả về cấu trúc Font mô tả việc chọn font của người dùng. Nó có thể sau đó được áp dụng vào một đối tượng Control hay Form để thay đổi font.
MaxSize	Lấy và cài kích cỡ điểm lớn nhất mà mọt ngươi dùng có thể chọn trong hộp dialog.
MinSize	Lấy hay cài kích cỡ điểm nhỏ nhất mà một người dùng có thể chọn trong hộp dialog.
ShowApply	Thuộc tính Boolean có thể được cài là true để hiển thị một nút Apply. Nếu nó được dùng, một bộ điều khiển sự kiện được viết để bắt được sựu

Property	Description
	kiện Apply khi nó xảy ra. Bởi vì khi người dùng chọ Apply, control vẫn không trả về cho ứng dụng, nhưng bộ điềukhiển sự kiện có thể sau đó xử lý font thay đổi trong ứng dụng.
ShowColor	Thuộc tính Boolean có thể được cài là true để hiển thị một danh sách màu. Nó cho phép người dùng chọn màu của font cũng như kiểu hay kích cỡ.
ShowEffects	Thuộc tính Boolean có thể được cài là true để cho phép người dùng chỉ định các tuỳ chọn strikethrough, underline, và text color.

Một bộ điều khiển sự kiện có thể được viết để trả lời sự kiện Apply khi nó được kích bởi người dùng ấn vào nút Apply.

```
FontDialog aFontDialog = new FontDialog();
aFontDialog.ShowColor = true;
aFontDialog.ShowEffects = true;
aFontDialog.MinSize = 1;
aFontDialog.MaxSize = 35;
aFontDialog.Font = SomeControl.Font;
if (aFontDialog.ShowDialog() == DialogResult.OK)
{
   SomeControl.Font = aFontDialog.Font;
}
aFontDialog.Dispose ();
```

Đầu tiên chúng ta khởi tạo đối tượng FontDialog() với một vài cài đặt, trong đó có thuộc tính *font* của *SomeControl* để mô tả đối tượng ta đang cập nhật. Việc khởi tạo thuộc tính font như trên là rất tốt bởi vì người dùng sẽ thấy font được chọn trong hộp dialog, và họ không lúng túng. Nếu người dùng chọn OK trong FontDialog thì thuộc tính font của SomeControl được cập nhật để phản ánh font mà người dùng chọn.

OpenFileDialog

Lớp này rất hữu ích, như nhiều ứng dụng yêu cầu người dùng điều hướng hệ thống tập tin để mở và dùng các tập tin dữ liệu. Dialog này là dialog Window chuẩn cho việc mở tập tin, và người dùng nên làm quen với nó.

Lớp này chứa nhiều thuộc tính dùng để cài hình thức và cách cư xử của chính hộp dialog đó. Cả hai lớp này và lớp *SaveFileDialog* thừa kế từ lớp cơ sở *FileDialog*. Vì nguyên nhân này nhiều thuộc tính được chia sẽ. Thuộc tính chính được trình bày ở trong bảng sau:

Property	**Description**
CheckFileExists	Cài thuộc tính này là true để làm cho hộp dialog hiển thị một cảnh báo nếu người dùng chỉ định một tên tập tin không tồn tại. Cách này làm cho đoạn mã của bạn không phải kiểm tra một đường dẫn. Mặc định là true.
FileName	Đây là thuộc tính quan trọng được dùng để cài và khôi phục tên tập tin được chọn trong hộp dialog tập tin.
FileNames	Nếu Multiselect là enabled, thuộc tính này sẽ trả về một mảng tên tập tin mà người dùng chọn.
Filter	Nó cài chuỗi lọc tên tập tin, để xác định các chọnlựa xuất hiện

Property	Description
	trong hộp "Files of type" trong hộp dialog.
FilterIndex	Chỉ mục của bộ lọc chọn trong hộp dialog tập tin.
InitialDirectory	Thư mục khởi tạo hiển thị bởi hộp dialog tập tin.
Multiselect	Thuộc tính Boolean được cài để cho biết hộp dialog có cho phép các tập tin bội có được chọn hay không. Mặc định là false.
ReadOnlyChecked	Thuộc tính Boolean cho biết nếu check box chỉ đọc được chọn. Mạc định là false.
RestoreDirectory	Thuộc tính Boolean cho biết hộp dialog có trả lại thư mục hiện tai trước khi đóng hay không. Mặc định là false.
ShowHelp	Thuộc tính Boolean cho biết nút Help có được hiển thị trong hộp dialog tập tin hay không.
ShowReadOnly	Thuộc tính Boolean cho biết dialog có chứa một check box chỉ đọc hay không.Mặc định là false.
Title	Tiều đề của dialog tập tin được hiển thị.

Lớp này được dùng để lấy một tên tập tin hay nhiều tên tập tin từ người dùng. Khi nó được thực hiện thì ứng dụng có thể xử lý một tập tin hay nhiều tập tin được cho biết bởi người dùng. Thuộc tính *Filter* là một khoá để cung cấp một giao diện hữu ích cho người dùng. Bằng cách thu hẹp các tập tin hiển thị thích hợp vào ứng dụng hiện hành, người dùng chắc chắn hơn để tìm tập tin chính xác.

Thuộc tính *Filter* là một chuỗi có thể chứa các tuỳ chọn lọc phức tạp. Mọi filter chứa một diện mạo tóm tắc, theo sau bởi một thanh dọc (|) và mẫu filter là một chuỗi tìm kiếm DOS. Các chuỗi cho các tuỳ chọn lọc khác nhau sẽ phân biệt bởi một thanh dọc.

Bạn có thể thêm các mẫu đa filter vào một filter đơn bằng cách phân các kiểu tập tin với dấu chấm phẩy. Ví dụ "Image Files(*.BMP;*.JPG;*.GIF)|*.BMP;*.JPG;*.GIF|All files (*.*)|*.*"

Đoạn mã sau tạo một đối tượng *OpenFileDialog*, định hình một số thuộc tính trên nó, và hiển thị nó với người dùng để cho phép họ chọn một tập tin. Ứng dụng có thể sau đó sử dụng thuộc tính *FileName* hay *FileNames* để xử lý tham khảo đến tập tin hay các tập tin.

```
OpenFileDialog aOpenFileDialog = new OpenFileDialog();
aOpenFileDialog.Filter = "Text Files (*.txt)|*.txt|Word Documents" +
                         "(*.doc)|*.doc|All Files (*.*)|*.*";
aOpenFileDialog.ShowReadOnly = true;
aOpenFileDialog.Multiselect = true;
aOpenFileDialog.Title = "Open files for custom application";
if (aOpenFileDialog.ShowDialog() == DialogResult.OK)
{
  //Do something useful with aOpenFileDialog.FileName
  //or aOpenFileDialog.FileNames
}
aOpenFileDialog.Dispose();
```

Hộp dialog này có ba tuỳ chọn trong combo box "Files of Type". Một tuỳ chọn là Text Files, một cái khác là Word Documents, và thứ ba là All Files.

SaveFileDialog

Hộp dialog này rất giống *OpenFileDialog*, và trong thực tế chúng thừa hưởng từ một lớp cơ sở. Các chức năng cơ bản của hộp dialog này là cho phép một người dùng chọn một nơi để lưu dữ liệu.

Nhiều thuộc tính giống như lớp *OpenFileDialog*; tuy nhiên các thuộc tính sau là các thành viên của *OpenFileDialog* và không tồn tại trong lớp *SaveFileDialog*:

- CheckFileExists
- Multiselect
- ReadOnlyChecked
- ShowReadOnly

Nhưng, hai thuộc tính sau chỉ hợp lệ khi là thành viên của lớp *SaveFileDialog*:

- CreatePrompt
- OverwritePrompt

Chú ý rằng các thuộc tính khác tạo cùng một kiểu, bao gồm các thuộc tính Filter, Title, và FileName. Đoạn mã sau trình bày cách sử dụng lớp này:

```
SaveFileDialog aSaveFileDialog = new SaveFileDialog();
aSaveFileDialog.Filter = "Text Files (*.txt)|*.txt|Word Documents" +
                         "(*.doc)|*.doc|All Files (*.*)|*.*";
aSaveFileDialog.CreatePrompt = true;
aSaveFileDialog.OverwritePrompt = true;
aSaveFileDialog.Title = "Save file for custom application";
if (aSaveFileDialog.ShowDialog() == DialogResult.OK)
{
  //Do something useful with aSaveFileDialog.FileName;
```

```
}
aSaveFileDialog.Dispose();
```

PageSetupDialog

Hộp dialog này được dùng để cài định hướng và lề của trang.

Thuộc tính chính trong lớp này là *Document*. Nó được yêu cầu trước khi phương thức *ShowDialog()* có thể được gọi, và một ngoại lệ được ném nếu nó không được gán một giá trị. Thuộc tính *Document* chấp nhận một đối tượng *PrintDocument*, đối tượng này là thành viên của *System.Drawing.Printing* namespace. Đối tượng này là cốt yếu cho tiến trình in ấn trong .NET, và tượng trưng các trang mà ạôt ứng dụng sẽ in. Bằng cách cài các thuộc tính của đối tượng này và sử dụng GDI+ để vẽ bề mặt của nó, ứng dụng có thể in bởi máy in.

Ví dụ bên dưới mô tả cách các hộp Dialog được gọi và sử dụng:

```
PageSetupDialog aPageSetup = new PageSetupDialog();
System.Drawing.Printing.PrintDocument aDoc = new
System.Drawing.Printing.PrintDocument();
aPageSetup.Document = aDoc;
if (aPageSetup.ShowDialog() == DialogResult.OK)
{
   //Do something useful with aPageSetup.Document;
}

aPageSetup.Dispose();
```

Đoạn mã này tạo một đối tượng *PageSetupDialog* mới, liên kết với một đối tượng *PrintDocument* mới với nó và hiển thị dialog.

PrintDialog

Dialog này được dùng để chọn máy in, số lượng bản sao, và các trang để in trong một tài liệu. Như dialog trước, đối tượng này yêu cầu một đối tượng *PrintDocument* có hiệu lực để được liên kết với thuộc tính *Document* trước khi nó được hiển thị.

Đối tượng cũng chứa các thuộc tính khoá sau:

Property	**Description**
AllowPrintToFile	thuộc tính Boolean có thể được cài là true để hiển thị checkbox "Print to file" trog dialog. mặc định là true.
AllowSelection	Thuộc tính Boolean có thể được cài là true để cho phép in ấn chỉ những phần hiện hành. Mặc định là false.
AllowSomePages	thuộc tính Boolean có thể được cài là true để cho biết các tuỳ chọn From Page và To Page là enabled. Mặc định là false.
Document	Thuộc tính PrintDocument mô tả bề mặt in ấn hiện hành.
PrintToFile	thuộc tính Boolean có thể được cài là true để cho biết checkbox "Print to file" được checked. khi diaog trả về nó có thẻ được check để thấy nếu ngươi dùng muốn ứng dụng in tài liệu vào một tập tin. Mặc định là false.
ShowHelp	Thuộc tính Boolean có thể được cài là true để cho biết nut Help nên được hiển thị. Mặc định là false.

Giống như các lớp dialog thông thường, các thuộc tính này được định hình trước khi gọi phương thức *ShowDialog()* để hiển thị hộp dialog chính xác cho người dùng. Đoạn mã dùng lớp này rất giống với đoạn mã trước:

```
PrintDialog aPrintDialog = new PrintDialog();
System.Drawing.Printing.PrintDocument aDoc = new
                                System.Drawing.Printing.PrintDocument();
aPrintDialog.Document = aDoc;
aPrintDialog.AllowSomePages = true;
aPrintDialog.AllowSelection = true;
if (aPrintDialog.ShowDialog() == DialogResult.OK)
{
  //Do something useful with aPrintDialog.Document;
}
aPrintDialog.Dispose();
```

PrintPreviewDialog

Lớp này cung cấp một cách nhanh chóng để giới thiệu các khả năng duyệt trước khi in vào một ứng dụng. Lớp này chấp nhận đối tượng *PrintDocument* trong thuộc tính *Document* của nó, và cùng đoạn mã mà điều khiển việc in ấn của mộtmáy in sẽ trả lại tài liệu cho hộp dialog này.

Hộp dialog này hỗ trợ co dãn, thu nhỏ, và đánh số trang và một tập các tuỳ chọn khác .

Visual Inheritance

.NET Framework lấy khái niệm thừa kế và cho phép một nhà phát triển sử dụng nó để phát triển các ứng dụng Windows Forms. Một đối tượng Form có thể thừa kế từ một đối tượng Form khác, vì thế chiếm được sự truy cập đến tất cả *Buttons*, *TextBoxes*, và *Menus*. Nó là một đặc trưng rất mạnh trong .NET khi sử dụng để giảm số lượng mã yêu cầu cho việc tạo các cửa sổ và màn hình giống nhau. Khái niệm này gọi **visual inheritance**.

Một Form luôn luôn thừa kế từ *System.Windows.Forms*. Có nghĩa là nó có thể truy cập đến tất cả thành phần dữ liệu và các phương thức của lớp *Form* cơ bản. Việc thực thi sự thừa kế yêu cầu một nhà phát triển thừa hưởng đối tượng Form từ một lớp *Form* tuỳ biến thay cho *System.Windows.Forms*. Đó là nguyên nhân tất cả control và thuộc tính trong lớp Form tuỳ biến truyền qua các lớp *Form* được tạo mới.

Tuy nhiên, có vài điều quan trọng phải nhớ. Cấp truy cập của các control khác nhau phải được hiểu, giống như cấp truy cập của các thừa kế chuẩn. Một thành phần dữ liệu *private* thì không thể được truy cập bởi bất kỳ đối tượng nào bên ngoài đối tượng ban đầu. Vì thế, nếu một control không được đánh dấu là *protected* hay *public*, lớp thừa hưởng sẽ không tham khảo đến control hay override bất kỳ phương thức của control.

Sử dụng thừa kế trực quan có thể rất có lợi khi thừa kế tạo ra một số lượng lớn màn hình mà phải có một thiết kế giống nhau và/hoặc làm các chức năng như nhau. Một ví dụ điển hình là một màn hình thực thể dữ liệu. Nếu ứng dụng của chúng ta không cần nhập các mẫu tin cá nhân, mà còn thông tin automobie, sử dụng thừa kế trực quan để định nghĩa một kiểu thông thường phải là một sự chọn lựa tốt. Hiển nhiên, chúng ta sẽ muốn một màn hình trông giống nhau, nhưng vài control sẽ thay đổi. Hãy sửa đổi ví dụ trước của chúng ta để sử dụng kỹ thuật này.

Tạo ra một Windows Application mới trongVisual Studio .NET và đặt tên nó là *VisualInheritance*.

Thay đổi các thuộc tính sau của đối tượng *Form1* mặc định. Chúng ta sẽ tạo một cửa sổ menu cung cấp cho người dùng các khả năng nhập các mẫu tin cá nhân hay các mẫu tin automobie.

- FormBorderStyle – FixedDialog
- MaximizeBox – False
- MinimizeBox – False
- Size – 200, 200

- StartPosition – CenterScreen
- Text – Main Menu

Đặt hai control trên Form. Định vị chúng ở giữa cửa sổ, đặt nhãn là *Person* và Automobie, và đặt tên là *btnPerson* và *btnAuto*. Chúng ta sẽ thêm các bộ điều khiển sự kiện vào sau để mở mọi Form thừa hưởng.

Bây giờ chúng ta sẽ thêm lớp Form cơ bản của chúng ta. Form này sẽ không bao giờ hiển thị trực tiếp, nhưng chúng ta sẽ dùng kiểu trực quan của nó trong mọi form thừa hưởng.

Thêm một Form mới vào ứng dụng bằng cách chọn Project | Add Windows Form. Bỏ qua các tên mặc định và chọn OK trong hộp dialog Visual studio.NET. Sửa đổi các thuộc tính sau của Form để tạo một kiểu trực quan duy nhất.

- Name – frmBase
- BackColor – White
- FormBorderStyle – FixedDialog
- MaximizeBox – False
- MinimizeBox – False
- Size – 250, 250
- StartPosition – CenterScreen
- Text – Base Form

Nó sẽ tạo một hộp dialog trắng. Bây giờ thêm hai *Buttons* vào góc phải của form. Chúng sẽ hành động như hai nút *Save* và *Cancel*. Bằng cách thêm chúng vào lớp cơ bản chúng sẽ được hiện trên các form thừa hưởng, vì thế bảo đảm một giao diện người dùng thông thường. Định vị hai nút trong góc phải và cài các thuộc tính:

Button	Name	Anchor	Location	Modifiers	Size	Text
Button1	btnSave	Bottom, Right	159, 151	Protected	75, 23	Save
Button2	btnCancel	Bottom, Right	159, 185	Protected	75, 23	Cancel

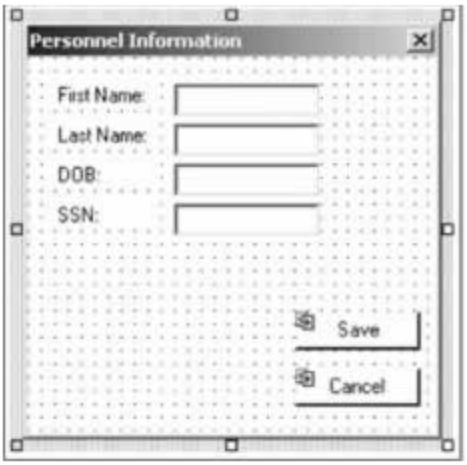
Thuộc tính quan trọng nhất phải chú ý đó là *Modifiers*. Nó có thể cài mức cách ly của lớp *Button* bên trong form. Nó có thể được cài bất kỳ mức : *public, protected, private,* hay *internal*. Sau khi chỉnh sửa thuộc tính trong cửa sổ *Properties*, xem xét đoạn mã để thấy các khai báo của hai đối tượng *Button* đã chỉnh sửa thành *protected*. Nó sẽ rất quan trọng trong việc cho phép các đối tượng Form thừa hưởng truy cập vào *Buttons*.

Các thành phần *protected* chỉ có thể được truy cập bởi các lớp thừa hưởng; chúng không được truy cập bởi bất kỳ đoạn mã nào bên trong. Các Form thừa hưởng không thể truy cập các control khai báo với mức *private* mặc định. Các buttons sẽ được hiển thị trên Forms thừa hưởng, nhưng không có bộ điều khiển sự kiện nào có thể được thêm, như các đối tượng không thể được truy cập từ các lớp thừa hưởng.

Cuối cùng chúng ta sẵn sàn thêm một Form thừa hưởng. Tuy nhiên, Visual studio.NET yêu cầu các lớp Form cơ bản được biên dịch đầu tiên, vì vậy đầu tiên chúng ta phải xây dựng dự án ít nhất một lần. Khi hoàn thành, chọn Project | Add Inherited Form. Bỏ qua tên mặc định của tập tin lớp bằng cách click *Open* trong hộp dialog kết quả. Tiếp đó chọn lớp Form cơ bản đúng để dùng. Một hộp dialog hiển thị các Form có giá trị hiện tại trong dự án, và cho phép bạn thừa hưởng lớp Form mới từ bất kỳ lớp nào của chúng. Chọn lớp frmBase và click OK

Một Form mới sẽ được tạo, nhưng nó sẽ trông giống như lớp *frmBase* ban đầu. Nó có cùng *BackColor* trắng và hai nút *Save* và *Cancel*. Thay đổi thuộc tính *Text* của Form theo các thông tin cá nhân, và thêm bốn control *Label* và bốn control *TextBox*. Thay đổi thuộc tính *Text* của các *Label* là "First Name:," "Last Name:", "DOB:" và "SSN:" Thay đổi thuộc tính Name của các control TextBox thành txtFName, txtLName, txtDOB, và

txtSSN, và để trống các thuộc tính *Text*. Form mới sẽ trông giống như màn hình bên dưới:



Lập lại tiến trình thêm một Form được thừa kế mới vào dự án. Một lần nữa thừa hưởng nó từ lớp *frmBase*. Thời điểm này thay đổi thuộc tính *Text* của Form để thông tin Automobie và thuộc tính Name vào *frmAuto* và thêm bốn Label vào Form với tựa : "Manufacturer:", "Model:", "Year:", và "Color:" Thêm bốn control *TextBox* định vị bên cạnh các *Label*, và thay đổi thuộc tính *Name* thành *txtManufact, txtModel, txtYear, txtColor* cho mọi *TextBox*.

Bây giờ chúng ta sẵn sàng thêm các bộ điều khiển sự kiện vào các Form thừa hưởng của ta. Nhớ rằng trong một ứng dụng chức năng thì nút *Save* sẽ được dùng giống như ADO.NET hay một đối tượng kinh doanh back-end để lưu dữ liệu vào một kho dữ liệu. Trong mẫu nàu thì dữ liệu được đưa vào một tập tin XML nhỏ.

Thêm bộ điều khiển sự kiện *Click* cho Form thông tin cá nhân cả hai nút *Save* và *Cancel*. Những nút này được thừa kế từ một lớp cơ bản, chúng có thể vẫn được thao tác và các sự kiện thêm vào giống như bất kỳ control khác.

```
private void btnSave_Click(object sender, System.EventArgs e)
{
  //Save the values to an XML file
```

```
//Could save to data source, Message Queue, etc.
System.Xml.XmlDocument aDOM = new System.Xml.XmlDocument();
System.Xml.XmlAttribute aAttribute;

aDOM.LoadXml("<PersonnelData/>");

//Add the First Name attribute to XML
aAttribute = aDOM.CreateAttribute("FirstName");
aAttribute.Value = txtFName.Text;
aDOM.DocumentElement.Attributes.Append(aAttribute);
//Add the Last Name attribute to XML
aAttribute = aDOM.CreateAttribute("LastName");
aAttribute.Value = txtLName.Text;
aDOM.DocumentElement.Attributes.Append(aAttribute);
//Add the DOB attribute to XML
aAttribute = aDOM.CreateAttribute("DOB");
aAttribute.Value = txtDOB.Text;
aDOM.DocumentElement.Attributes.Append(aAttribute);
//Add the SSN attribute to XML
aAttribute = aDOM.CreateAttribute("SSN");
aAttribute.Value = txtSSN.Text;
aDOM.DocumentElement.Attributes.Append(aAttribute);
```

```
    //Save file to the file system
    aDOM.Save("PersonnelData.xml");
  }

  private void btnCancel_Click(object sender, System.EventArgs e)
  {
    txtLName.Text = "";
    txtFName.Text = "";
    txtDOB.Text = "";
    txtSSN.Text = "";
  }
```

Đoạn mã này có vẽ quen với bạn, bởi vì nó giống với đoạn mã XML từ ví dụ trước. Ý tưởng cơ bản là xếp thứ tự các nội dung của TextBoxes vào một tập tin XML và lưu nó vào một hệ thống tập tin. Nút *Cancel* để xoá các control *TextBox*.

Đoạn mã cho Automobile Information Form thì càng giống ngoại trừ sự khác nhau về tên TextBox được dùng và một tập tin XML khác được tạo. Khi nó không biểu lộ các tin tức chúng ta sẽ không hiện nó.

Cuối cùng , thêm bộ điều khiển sự kiện sau vào sự kiện Click của nút trong Form1.

```
private void btnPerson_Click (object sender, System.EventArgs e)
{
  Form3 aForm = new Form3();
  aForm.ShowDialog();
}
```

```
private void btnAuto_Click(object sender, System.EventArgs e)
{
  Form4 aForm = new Form4();
  aForm.ShowDialog();
}
```

2.3 Windows Controls

Các ứng dụng Windows Forms bao gồm nhiều contrrol khác nhau. Các control này có thể đơn giản như các control *Button* và *TextBox*, hay chúng có thể tinh vi và phức tạp hơn như các control *Charting* và *TreeView*. .NET framework có nhiều control sẵn sàng kết hợp với các ứng dụng Windows Forms, và có hàng trăm control được dùng trong các phát triển ứng dụng .NET tuỳ biến. Chính vì thế, chúng ta sẽ xem xét cách tất cả control hoạt động và tương tác tại một cấp cao hơn.

Các control trong Windows Forms bao gồm những cái mà một nhà phát triển sẽ muốn tìm trong một thư viện lớp được thiết kế cho các giao diện người dùng đồ hoạ:

- Labels
- Buttons
- Checkboxes
- Menus
- Radio buttons
- Combo boxes
- Listboxes
- Textboxes
- Tabcontrols

- Toolbars
- Tree views

Như chúng ta thấy, Visual Studio .NET có thể thêm các control này vào một Form cho bạn. Các bước xảy ra khi một control được thêm vào một Form như sau:

1. Một biến của kiểu control yêu cầu được khai báo như một đối tượng riêng trong lớp Form.
2. Trong phương thức *InitializeComponent()*, đối tượng control được tạo và gán vào một biến riêng.
3. Các thuộc tính của control, như là Location, Size, và Color được cài bên trong phương thức *InitializeComponent()*.
4. Control được thêm vào tập hợp control trên form.
5. Cuối cùng, các bộ điều khiển sự kiện được thực thi khi nhà phát triển thêm chúng vào thông qua IDE

Mọi control thừa kế từ *System.Windows.Forms.Control*. Lớp cơ bản này chứa các phương thức và các thuộc tính cơ bản được dùng bởi bất kỳ control nào cung cấp một giao diện người dùng cho người sử dụng. Control này quản lý chức năng cơ bản được yêu cầu để chiếm bàn phím và chuột như là định nghĩa kích cở của nó và vị trí trên cha mẹ của nó.

2.3.1 Dynamic Controls

Kể từ khi tất cả control khả thị thừa kế từ lớp *Control*, chúng ta có thể thấy những thuận lợi của đa hình khi làm việc với các tập hợp control. Tất cả control chứa một thuộc tính *Controls* hoạt động như một tập hợp control chứa đựng. Nó cho phép bạn viết mã lặp qua các tập hợp Controls và vận dụng hay yêu cầu mọi control riêng lẽ sử dụng các thuộc tính và phương thức của lớp *Control*.

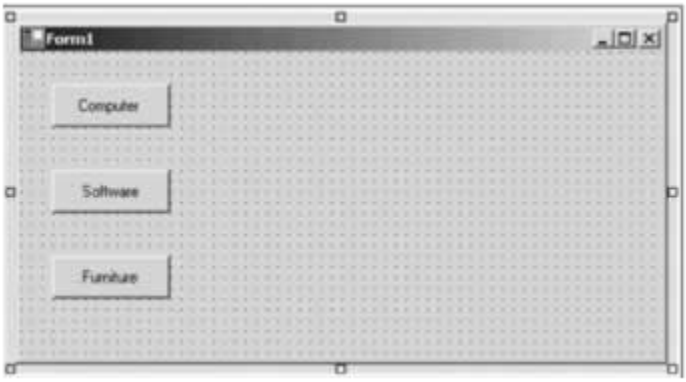
Thuộc tính *Control* này thì động và có thể được dùng để tuỳ chỉnh hình thức của giao diện người dùng tại thời gian chạy bằng cách thêm và xoá các control vào một *Form*

hay *Control*. Giống như tất cả tập hợp, tập hợp control có các phương thức cho phép thêm và huỹ các đối tượng, đó là nguyên nhân các control đượ thêm vào và huỹ từ các giao diện người dùng. Nó có thể là một kỹ thuật mạnh để thiết kế giao diện người dùng tuỳ biến. Trên thực tế, nếu bạn xem xét phương thức *InitializeComponent()* tạo bởi Visual studio.NET bạn sẽ thấy chính xác cách đoạn mã thêm các control vào các *Form*, và nó cũng là cách mà chúng ta thêm control *Button* vào ứng dụng Windows Forms đầu tiên của chúng ta ở đầu chương. Hãy tạo một ứng dụng có các thuận lợi về khả năng này để tuỳ chỉnh giao diện người dùng tại thời gian chạy.

Một ứng dụng thông thường yêu cầu các màn hình khác nhau cho mọi đối tượng khác nhau. Một ví dụ là một hệ thống quản lý hàng tồn, hệ thống này phải quản lý các máy tính, phần mềm, và trang bị. Mọi đối tượng này có các thuộc tính duy nhất; tuy nhiên chúng chia sẽ một vài đặc tính chung. Phụ thuộc vào thiết kế giao diện, nó rất hữu ích để tuỳ chỉnh giao diện người dùng dựa vào kiểu đối tượng đang được thao tác trên hệ thống và chỉ hiển thị các trường đó. Tuỳ chỉnh động của giao diện người dùng này có thể được hoàn hảo bởi việc vận dụng thuộc tính *Control* tại thời gian chạy.

Mở Visual Studio .NET và tạo một ứng dụng Window C# mới với tiêu đề *DynamicUI*.

Thêm ba control Button ở đâu đó trên bên trái của Form. Gán các nút với thuộc tính *Text* là Computer, Software, và Furniture. *Form* nên giống như bên dưới:



Khi một ngươi dùng chọn nút thích hợp thì giao diện người dùng sẽ tự động tuỳ chỉnh để nhập kiểu đối tượng đó. Một ứng dụng xí nghiệp sẽ dùng các cài đặt này từ một

nguồn dữ liệu back-end hay tập tin định hình; tuy nhiên, chúng ta sẽ nhấn mạnh tính logic của hiển thị một cách trực tiếp trong ứng dụng. Nó không là một ví dụ tốt, và nếu ví dụ này đực mở rộng thì bước đầu tiên là tạo một phương thức tượng trưng cho giao diện người dùng.

Có một số thuộc tính chúng ta sẽ cài đặt cho mọi control chúng ta thêm vào Form. Nó bao gồm các thuộc tính *Size* và *Location* khi các control được định vị ở đâu đó. Chúng ta cũng thường cài các thuộc tính *Text* và *Name*. Khi thêm một số lượng lớn control vào một form thì tiến trình này có thể nhanh chóng dẫn đến dư thừa và lặp lại mã, do đó để tránh chúng ta sẽ tạo một phương thức tiện ích để cài các thuộc tính này một lần. Chúng ta có thể gọi phương thức này cho mọi control mà chúng ta thêm vào Form. Thêm các phương thức riêng vào lớp *Form1*.

```
private void AddControl(Control aControl, Point Location, Size Size,
                        String strText, int TabIndex, string strName)
{
  aControl.Location = Location;
  aControl.Size = Size;
  aControl.Text = strText;
  aControl.TabIndex = TabIndex;
  aControl.Name = strName;
  this.Controls.Add(aControl);
}
```

Phương thức này chấp nhận một đối tượng *Control* và cài các thuộc tính public trên nó. Chú ý rằng khi chúng tra gọi phương thức này chúng ta sẽ truyền vào một lớp thừa hưởng, giống như một *Label* hay *TextBox*. Việc này có thể làm được thông qua đa

hình, bởi vì lớp *Control* cơ bản định nghĩa các thuộc tính được truy cập bởi các thuộc tính này.

Bây giờ, chúng ta thêm các bộ điều khiển sự kiện cho các *Button*. Thêm một bộ điều khiển sự kiện *Click* cho mọi nút và thêm đoạn mã sau. Tên của các bộ điều khiển sự kiện sẽ khác nhau phụ thuộc vạo tên của các *Button*:

```
private void btnComp_Click(object sender, System.EventArgs e)
{
  Controls.Clear();
  InitializeComponent();

  AddControl(new Label(),new Point(125,24),new Size(45,20),"ID:",0,"");
  AddControl(new TextBox(),new Point(174,21),new Size(125, 20),
        "",0,"txtID");
  AddControl(new Label(),new Point(125,54),new Size(45,20),"OS:",0,"");
  AddControl(new TextBox(),new Point(174,50),new Size(125,20),
        "",1,"txtOS");
  AddControl(new Label(),new Point(125,84),new Size(45,20),
        "Speed:",0,"");
  AddControl(new TextBox(),new Point(174,78),new Size(125,20),
        "",2,"txtSpeed");
}

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

```
{
   Controls.Clear();
   InitializeComponent();

   AddControl(new Label(),new Point(125,24),new Size(45,20),
            "ID:",0,"");
   AddControl(new TextBox(),new Point(174,21),new Size(125, 20),
            "",0,"txtID");
   AddControl(new Label(),new Point(125,54),new Size(45,20),
            "Vendor:",0,"");
   AddControl(new TextBox(),new Point(174,50),new Size(125, 20),
            "",1,"txtVendor");
   AddControl(new Label(),new Point(125,84),new Size(45,20),
            "Name:",0,"");
   AddControl(new TextBox(),new Point(174,78),new Size(125, 20),
            "",2,"txtName");
}

private void btnFurn_Click(object sender, System.EventArgs e)
{
   Controls.Clear();
   InitializeComponent();
```

```
    AddControl(new Label(),new Point(125,24),new Size(45,20),"ID:",0,"");
    AddControl(new TextBox(),new Point(174,21),new Size(125, 20),
            "",0,"txtID");
    AddControl(new Label(),new Point(125,54),new Size(45,20),
            "Color:",0,"");
    AddControl(new TextBox(),new Point(174,50),new Size(125, 20),
            "",1,"txtColor");
    AddControl(new Label(),new Point(125,84),new Size(45,20),
            "Type:",0,"");
    ComboBox aCombo = new ComboBox();
    aCombo.Items.AddRange(new Object[] {"Desk","Chair","Whiteboard"});
    AddControl(aCombo,new Point(174,78),new Size(125, 20),"",2,"");
}
```

Chạy ứng dụng và chọn các tuỳ chọn khác nhau thông qua các nút. Bạn sẽ thấy giao diện người dùng tự tuỳ chỉnh dựa vào kiểu nhập vào từ người dùng.

2.4 Các Control tuỳ biến(Custom control)

Custom controls là khía cạnh rất quan trọng của việc phát triển Windows Forms. Các loại control này được gọi là *ActiveX* control hay *UserControls* trong Visual Basic. Khái niệm cơ bản là để cho phép một nhà phát triển tạo ra chức năng mới và/hoặc gộp các control đang tồn tại vào một control chung, control này có thể dễ dàng sử dụng lại ở bên kia ứng dụng hay trong các đa ứng dụng.

Custom controls cho phép các nhà phát triển gói chức năng và bản trình bày vào một gói lớp đơn mà có thể sử dụng lại dễ dàng suốt một ứng dụng. Giao diện người dùng, các sự kiện, các thuộc tính và các phương thức có thể được cài và định hình bởi nhà phát

triển. Nhóm chức năng này có thể đước chèn vào dự án nếu cần thiết một cách dễ dàng. Nó cũng có khả năng sử dụng lại các custom control, và toàn bộ thư viện control có thể được phát triển để sử dụng bởi các nhà phát trển trong một tổ chức.

Thêm vào đó, nếu một custom control rất hữu ích và thể hiện chức năng thì các nhà phát triển khác bên ngoài một tổ chức đơn sẽ rất muốn có nó, và hoàn toàn có thể bán control theo cách thương mại. Một nền công nghiệp toàn phần cung cấp các custom control cho các nhà phát triển ứng dụng. Các công ty này tập trung thời gian và tài nguyên của họ vào việc phát triển một chức năng của control một cách rộng rãi; vì thế nó luôn luôn rẽ hơn để mua một trong số các control này thay vì tự tạo ra nó.

Tóm lại, một custom control là một lớp tuỳ biến mà liên kết chức năng logic doanh nghiệp và/hoặc các đa control và tuỳ biến hiển thị logic trong một gói đơn. Gói này có thể được sử dụng lại nhiều lần trong một Form, giống như một *Button* chuẩn hay control *Label*. Một ví dụ cho vấn đề này trong một ứng dụng doanh nghiệp là một *TextBox*.

2.4.1 Lớp UserControl

Mọi custom controls nên thừa kế từ lớp *System.Windows.Forms.UserControl*. Nó đặt plumbing thích hợp vào nơi mà được yêu cầu với các host *Control*, quản lý thanh cuộn, và cung cấp một bề mặt thiết kế cho nhà phát triển. Lớp này hành động giống như lớp Form cơ bản trong đó nó cung cấp một sự thực thi cơ bản và các lớp thừa hưởng tuỳ biến cung cấp chức năng doanh nghiệp. Một control không cần đoạn mã pluming này bởi vì nhà phát triển sẽ viết tất cả cho nó hay nó sẽ không hiển thị một giao diện người dùng có thể đơn giản thừa kế trực tiếp từ *System.Windows.Control*.

Trên thực tế, xây dựng một custom control thì rất giống với xây dựng một *Form*. Một *UserControl* chứa một vùng client trong đó các control Windows Forms có thể được vận dụng. Các bộ điều khiển sự kiện có thể gắn vào các control này để tương tác với người dùng. Sự khác nhau là control đó phải luôn luôn nghĩ về cách các nhà phát triển khác sẽ sử dụng custom control này trong một ứng dụng hosting. Custom control hành

động như một cấp đơn giản hoá cho các nhà phát triển tầng thứ ba, những người mà không cần hiểu cách control làm việc. Control sẽ không tồn tại độc lập mà nó sẽ luôn luôn thực thi bên trong một ứng dụng hosting.

2.4.1.1 Các Control constituent

Các controls mà trang điểm cho giao diện người của một custom control được gọi là *constituent control*. Các control này được sở hữu bởi đối tượng *UserControl*. Chúng rất hữu ích cho việc xây dựng có thể dùng lại các thành phần giao diện ngươi dùng mà có thể được vận dụng và định vị như một nhóm. Ví dụ, giả sử mỗi cửa sổ trong một ứng dụng chứa cùng nhóm control RadioButton. Để thay vào việc sao chép và dán các control này vào mỗi Form, một sự chọn lựa sẽ tạo một custom control mà chứa nhóm control *RadioButton* chung này. Một nhà phát triển có thể thêm custom control này vào bất kỳ *Form* nào, định vị nó vào vị trí thích hợp, và các control constituent *RadioButton* trong *User Control* sẽ hiển thị và cập nhật chính xác mà không có bất kỳ sự tương tác nào với các ứng dụng hosting.

Tuy nhiên, các *constituent control* không thể được truy cập từ bên ngoài đối tượng *User Control*, khi chúng được khai báo là các biến private như mặc định. Mặc dù một nhà phát triển có thể thay đổi bằng tay đặc tính này để các *constituent control* là public, tuy nhiên nó xem như một sự vi phạm. Phương thức đúng của việc này là yêu cầu thông tin để định nghĩa các thuộc tính, các phương thức và các sự kiện là public trong lớp custom *UserControl* và sắp xếp các yêu cầu này vào các thuộc tính, phương thức, sự kiện khác nhau của *constituent control*. Nó cho phép *User Control* có thể chỉnh sửa bên trong Visual Studio .NET IDE, và cung cấp một môi trường thiết kế phù hợp cho nhà phát triển client.

2.4.1.2 Các Control mở rộng

Custom controls không luôn luôn cần được tạo ra bởi liên kết các *constituent control*. Mở rộng và tuỳ chỉnh chức năng của một control Window Forms đơn có thể rất hữu ích bởi vì nó có thể ẩn sự tuỳ biến bên dưới một control chuẩn cho người dùng. E-mail validation TextBox là một ví dụ điển hình. Để thay cho việc viết lại một control

TextBox và thêm các chức năng yêu cầu. E-mail validation *TextBox* có thể được sử dụng lại trong các ứng dụng như một sự thay thế cho một *TextBox* chuẩn và người dùng sẽ không biết sự khác nhau. Nó có thể thực hiện trong .NET bằng cách sử dụng thừa kế và có thể là một cách hiệu quả để tạo một custom control.

Để mở rộng một control đang tồn tại thì sự khai báo cho một *User Control* phải được thay đổi để thừa kế từ control đang tồn tại để thay cho *System.Windows.Forms.UserControl*. Bởi vì sức mạnh của việc thừa kế, bạn có thể truy cập đến tất cả các thuộc tính, phương thức, sự kiện của lớp cơ sở.

2.4.1.3 Các Event exposing

Các sự kiện là khía cạnh quan trọng khác của sự phát triển control, và cung cấp các sự kiện hữu ích cho các nhà phát triển cho phép họ phản ứng lại các hoạt động xảy ra bên trong custom control và phản ứng lại trong các ứng dụng của họ.

Khi thiết kế các custom control, một vài tuỳ chọn có hiệu lực cho các exposing events. Một hay nhiều sự kiện của các *constituent control* nằm trong chính *User Control* của nó có thể được phô ra. Nhắc lại, các constituent control được khai báo private và không được truy cập bởi các nhà phát triển client. Ví dụ, giả sử một nhà phát triển tạo một custom control chứa một control *ListBox* và một control *Button*, và control *ListBox* được phổ biến trước với các đoạn mã *States* và *State*. Custom control này có thể được dùng để cho phép một người dùng chọn một *State* từ một danh sách và click vào nút khi họ chọn đúng. Nhà phát triển client cần được viết mã mà được xử lý khi *Button* bên trong *User Control* được click, giống như một nhà phát triển sẽ viết mã bên dưới mọt sự kiện click của Button chuẩn.

Để biểu diễn sự kỳ diệu này, *User Control* phải gọi sự kiện tới ứng dụng hosting. Nó được thực hiện bởi việc thêm một bộ điều khiển sự kiện bên dưới sự kiện *constituent control* mà sẽ được theo dõi. Nó sẽ được thực hiện trong *User Control*.

2.4.1.4 Các Event tuỳ biến

Chú ý rằng tiến trình này chỉ làm việc cho các sự kiện mà có hiệu lực đối với lớp *UserControl*. Chúng ta không tạo bất kỳ sự kiện mới nào; chúng ta chỉ rỏ khi nào các sự kiện này được kích cho một ứng dụng client. Vài điều thường được yêu cầu trong một *User Control* là phô ra một sự kiện tuỳ biến mà có thể được sử lý bởi các ứng dụng hosting.

Một *User Control* phải chứa hay sử dụng các nội dung sau để thể hiện một sự kiện tuỳ biến:

- Một lớp nắm dữ liệu trạng thái sự kiện. Nó phải thừa hưởng từ *System.EventArgs*. Nó là cách dữ liệu được truyền qua tự một sự kiện đến client. Nó phải được định nghĩa ben ngoài lớp *UserControl*.
- Một đại diện cho sự kiện, được định nghĩa lại bên ngoài lớp *UserControl*.
- *User Control* phải chứa khai báo sự kiện sử dụng các kiểu đại diện và một phương thức để kích sự kiện.