Formulários, Controles, e Menus

O primeiro passo para criar uma aplicação com o Visual Básico é criar a interface, a parte visual da aplicação com que o usuário interagirá. Forms e controles são os blocos básicos do edifício usados para criar a interface; eles são os objetos que você trabalhará para construir sua aplicação.

Forms são objetos que expõem propriedades que definem a sua aparência, métodos que definem o seu comportamento, e eventos que definem sua interação com o usuário. Fixando as propriedades da form e escrevendo o código Visual Basic para responder a seus eventos, você personaliza o objeto para satisfazer as exigências de sua aplicação.

Controles são objetos contidos dentro de objetos de form. Cada tipo de controle tem seu próprio conjunto de propriedades, métodos e eventos que os fazem satisfatório para um propósito particular. Alguns dos controles que você pode usar em suas aplicações são bons para entrar ou exibir texto. Outros controles lhe deixaram ter acesso outras aplicações e dados de processo como se a aplicação distante fosse parte de seu código.

Este capítulo introduz os conceitos básicos de funcionamento de forms e controles, e as propriedades, métodos, e eventos associadas à eles. Muitos dos controles padrões são discutidos, como também itens específicos da form, como menus e caixas de diálogo.

Compreendendo Propriedades, Métodos e Eventos Uma introdução aos objetos e suas propriedades, métodos, e eventos associados.

Forms e controles Visual Basic são objetos que expõem as suas próprias propriedades, métodos e eventos. Propriedades podem ser pensadas como atributos de um objeto, métodos como suas ações, e eventos como suas respostas.

Um objeto cotidiano como o balão de hélio de uma criança também tem propriedades, métodos e eventos. As propriedades de um balão incluem atributos visíveis como sua altura, diâmetro e cor. Outras propriedades descrevem seu estado (inchou ou não inchou), ou atributos que não são visíveis como sua idade. Por definição, todos os balões têm estas propriedades; as colocações destas propriedades podem diferir de um balão a outro.

Um balão também tem métodos inerentes ou ações que poderia executar. Tem um inche método (a ação de recheio isto com hélio), um esvazie método (expelindo seus conteúdos) e um método de subida (se você fosse deixar ir). Novamente, todos os balões são capazes destes métodos.

Balões também têm respostas predefinidas a certos eventos externos. Por exemplo, um balão responderia ao evento de ser perfurado se esvaziando, ou para o evento de ser lançado subindo no ar.

Figure 3.1 Objetos têm propriedades, respondem a eventos, e executam métodos

Se você pudesse programar um balão, o código Básico Visual poderia se parecer o seguinte. Fixar as propriedades do balão:

```
Balloon.Color = Red
Balloon.Diameter = 10
Balloon.Inflated = True
```

Note a sintaxe do código - o objeto (Balão) seguiu pela propriedade (.Color) seguiu pela tarefa do valor (Vermelho). Você poderia mudar a cor do balão de código repetindo esta declaração e substituindo um valor diferente. Também podem ser fixadas propriedades na janela de Propriedades enquanto você está projetando sua aplicação.

Os métodos de um balão são invocados assim:

```
Balloon.Inflate
```

```
Balloon.Deflate
Balloon.Rise 5
```

A sintaxe é semelhante à propriedade - o objeto (um substantivo) seguiu pelo método (um verbo). No terceiro exemplo, há um artigo adicional, chamou um argumento que denota a distância para subir. Alguns métodos terão um ou mais argumentos para avançar descrever a ação a ser executada.

O balão poderia responder a um evento como segue:

```
Sub Balloon_Puncture()
     Balloon.Deflate
     Balloon.MakeNoise "Bang"
     Balloon.Inflated = False
     Balloon.Diameter = 1
End Sub
```

Neste caso, o código descreve o comportamento do balão quando um evento de perfuração acontece: invoque o Esvazie método, então invoque o método de MakeNoise com um argumento de " Bang " (o tipo de ruído para fazer). Desde que o balão já não é inchado, a propriedade Inchada é fixada para Falso e a propriedade de Diâmetro é fixada a um valor novo.

Enquanto você não pode programar um balão de fato, você pode programar uma form ou controle Visual Basic. Como o programador, você tem o controle. Você decide quais propriedades deveriam ser mudadas, métodos invocar ou que eventos responder para alcançar o aparecimento desejado e comportamento.

Projetando uma Form Os fundamentos de funcionamento com as propriedades de uma form, métodos, e eventos.

Objetos de form são os blocos básicos do edifício uma aplicação Visual Basic, as janelas atuais com que um usuário interage quando eles executam a aplicação. Forms têm suas próprias propriedades, eventos, e métodos com que você pode controlar o seu aparecimento e comportamento.

Figure 3.2 Forms e controles têm suas próprias propriedades, eventos, e métodos

O primeiro passo para projetar uma form é ajuste suas propriedades. Você pode fixar as propriedades de uma forma em tempo de desenvolvimento na janela de Propriedades, ou em tempo de execução escrevendo no código.

Nota Você trabalha com forms e controles, ajuste as propriedades deles, e escreve o código em tempo de desenvolvimento para os seus eventos a qualquer hora que você estiver construindo uma aplicação no ambiente Visual Basic. Tempo de Execução é qualquer hora você está executando a aplicação de fato e está interagindo com a aplicação como o usuário iria.

Ajustando as Propriedades da Form

Muitas das propriedades de uma form afetam seu aparência física. A propriedade Caption determina o texto que é mostrado na barra de título da form; a propriedade de Ícone fixa o ícone que é exibido quando uma form é minimizada. As propriedades de MaxButton e MinButton determinam se a form pode ser maximizada ou pode ser minimizada. Mudando a propriedade de BorderStyle, você pode controlar o comportamento de redimensionamento da form.
As propriedades Height e Width determinam o tamanho inicial de uma form; As propriedades Left e Top determinam a localização da forma em relação ao canto à esquerda superior da tela. A propriedade de WindowState pode ser fixada para inicializar a form maximizada, minimizada, ou em estado normal.

A propriedade de Name fixa o nome pelo qual você se referirá à form em código. Por padrão, quando uma form é adicionada primeiro a um projeto, seu nome é fixado a Form1, Form2, e assim sucessivamente. É uma boa idéia fixar a propriedade Name para algo mais significante, como " frmEntry " para uma form de entrada de ordem.
O melhor modo para se familiarizar com as muitas propriedades de form é experimentar. Mude algumas das propriedades de uma forma na janela de Propriedades (Figura 3.3), então execute a aplicação para ver o efeito delas. Você pode aprender mais sobre cada propriedade selecionando-a e apertando F1 para ver a Ajuda de contexto-sensível.
Figure 3.3 A janela de Propriedades

Eventos e Métodos da Form
Como objetos, forms podem executar métodos e podem responder a eventos.
O evento Resize de uma form é ativado sempre que uma forma é redimensionada, ou através de interação de usuário ou por código. Isto lhe permite executar ações como mover ou redimensionar controles em uma form quando suas dimensões mudarem.
O evento Activate acontece sempre que uma form se torna a form ativa; o evento Deactivate acontece quando outra forma ou aplicação fica ativa. Estes eventos são convenientes para inicializar ou finalizar o comportamento da form. Por exemplo, no evento Activate você poderia escrever o código para destacar o texto em uma caixa de texto particular; no evento Deactivate você poderia salvar as alterações para um arquivo ou banco de dados.
Para fazer uma form visível, você invocaria o método Show:
Form2.Show

Invocando o método Show tem o mesmo efeito como ajustado na propriedade Visible de uma form para True.
Muitos dos métodos de uma form envolvem texto ou gráficos. Os métodos Print, Line, Circle, and Refresh são úteis para imprimir ou desenhar diretamente sobre a superfície de uma form. Estes métodos e mais são discutidos em " Trabalhando com Texto e Gráficos ".
Para Mais Informação Para informação adicional sobre forms, veja " Mais Sobre Forms " " Criando uma Interface " de Usuário.

Clicando Botões para Executar Ações Uma introdução para o controle Botão de Comando.

O modo mais fácil para permitir ao usuário interagir com uma aplicação é colocar um botão para clicar. Você pode usar o controle botão de comando existente no Visual Basic, ou você pode criar seu próprio " botão " usando um controle de imagem que contém um gráfico, como um ícone.

Usando Botões de Comando
A maioria da aplicações Visual Basic têm botões de comando que permitem ao usuário os clicar para executar ações. Quando o usuário escolhe o botão, não só leva a cabo a ação apropriada, ele também vê se está sendo pressionado ou libertado. Sempre que o usuário clica um botão, o procedimento de evento Click é invocado. Você coloca código no procedimento de evento Click para executar alguma ação que você escolhe.
Há muitos modos para escolher um botão de comando em tempo de execução:

- Usar o mouse para clicar no botão.
- Mover o foco para o botão pressionando a tecla Tab, e então escolher o botão pressionando a Barra de espaço ou Enter. (Veja "Compreendendo o Foco" posteriormente neste capítulo.)
- Pressionar uma tecla de acesso (ALT+ a letra sublinhada) para um botão de Comando.
- Configurar a propriedade Value do botão de comando para True no código:

```
cmdClose.Value = True
```

- Invocar o Evento Click do botão de comando no código:

```
cmdClose_Click
```

- Se o botão de comando é o botão de comando padrão (*default*) para a form, pressionar ENTER escolherá este botão, até mesmo se você mudar o foco de um controle diferente para um botão de comando. Em tempo de desenvolvimento, você especifica um botão de comando padrão configurando a propriedade Padrão daquele botão para True.
- Se o botão de comando é o botão de Cancelamento padrão para a form, então pressionar ESC escolherá o botão, até mesmo se você mudar foco a outro controle. Em tempo de desenvolvimento, você especifica um botão de Cancelamento padrão configurando a propriedade Cancel daquele botão para True.

Todas estas ações fazem o Visual Basic invocar o procedimento de evento Click.

A aplicação Testar Botões

Você usa a propriedade Caption para exibir texto no botão para dizer ao usuário o que o botão faz. Na Figura 3.4, o exempo Testar Botões, contém um botão de comando com sua propriedade Caption configurada para " Mudar Sinal ". (Para trabalhar com uma versão deste exemplo, veja Button.frm na aplicação Controls.vbp.)

Note que ' S' é a tecla de acesso para este botão, denotada por um sublinhe. Inserindo um "e comercial" (&) no texto da propriedade Caption faz com que essaa tecla acesse aquele botão (por exemplo, Mudar &Signal).

Figura 3.4 botão de Comando com uma legenda

Quando um usuário clica o botão de comando, o código no procedimento de evento Click do botão de comando é executado. No exemplo, um ícone de semáforo diferente é exibido cada vez que o botão é clicado.

Para Mais Informação Para informação sobre propriedades adicionais do botão de comando, veja " Usando Controles Padrões do Visual Basic.

Controles para Exibir e Entar Texto Uma introdução para os controles Rótulo e Caixa de Texto.

Label and text box controls are used to display or enter text. Use labels when you want your application to display text on a form, and text boxes when you want to allow the user to enter text. Labels contain text that can only be read, while text boxes contain text that can be edited.

Para obter esta característica	**Use este controle**
Texto que pode ser editado pelo usuário, por exemplo um campo de entrada ordenado ou uma caixa de senha	Caixa de Texto
Texto que é somento exibido, por exemplo para identificar um campo em uma form ou exibir instruções para o usuário	Rótulo

Rótulos e Caixas de Texto são discutidas nas próximas seções:

- **Usando Rótulos para Exibir Texto** Os fundamentos para usar o controle Rótulo.

Um controle rótulo exibe texto que o usuário não pode mudar diretamente. Você pode usar rótulos para identificar controles, como caixas de texto e barras de rolagem que não têm a própria propriedade Caption deles. O texto atual exibido em um rótulo é controlado pela propriedade Caption que pode ser configurada em tempo de desenvolvimento na janela de Propriedades ou em tempo de execução nomeando-o em código.

Por padrão, a legenda é a única parte visível do controle rótulo. Porém, se você configurasse a propriedade BorderStyle para 1 (o qual você pode fazer em tempo de desenvolvimento), o rótulo aparece com uma borda - dando a ele uma aparência semelhante a uma caixa de texto. Você também pode mudar o aparecimento do rótulo configurando as propriedades BackColor, BackStyle, ForeColor, e Font.

Ajustando o tamanho do Rótulo para acomodar Seus Conteúdos

Podem ser especificadas legendas para o rótulo em tempo de desenvolvimento na janela de Propriedades. Mas e se você quer entrar em uma legenda mais longa, ou uma legenda que mudará em tempo de execução? Rótulos têm duas propriedades que o ajudam alterar seu tamanho para ajustar legendas maiores ou menores: AutoSize e WordWrap.
A propriedade de AutoSize determina se um controle deveria ser automaticamente redimensionado para ajustar seus conteúdos. Se configurar True, o rótulo cresce horizontalmente para ajustar seus conteúdos, como mostrado em Figura 3.5.
Figure 3.5 Exemplo de Auto Ajuste

A propriedade WordWrap causa o aumento vertical do rótulo para ajustar seus conteúdos, enquanto retendo a mesma largura, como mostrado em Figura 3.6. Para trabalhar com uma versão deste exemplo, veja Wordwrap.frm na aplicação Controls.vbp.
Figure 3.6 Exemplo WordWrap

Note Se você executar o exemplo de AutoSize do Controls.vbp, você notará que para trabalhar de fato, devem ser selecionadas ambas as caixas de verificação para o exemplo de WordWrap. Isto porque, para a propriedade WordWrap da etiqueta ter efeito, AutoSize deve ser fixado para True. A largura da etiqueta só é aumentada se a largura de uma única palavra excede a largura atual do controle.

Para Mais Informação Para informação adicional sobre as propriedades do controle rótulo, veja " Usando Controles " Padrões do Visual Basico.

- **Trabalhado com Caixas de Texto** Os fundamentos para usar o controle Caixa de texto.

  Caixas de texto são controles versáteis que podem ser usados para obter texto introduzido pelo usuário ou exibir texto. Não deveriam ser usadas caixas de texto para exibir texto que você não quer que o usuário mude, a menos que você configurasse a propriedade Locked para True.

  O texto atual exibido em uma caixa de texto é controlado pela propriedade Text. Pode ser configurado de três modos diferentes: em tempo de desenvolvimento na janela de Propriedade, em tempo de execução através de código, ou pela entrada do usuário em tempo de execução. Podem ser recobrados os conteúdos atuais de uma caixa de texto em tempo de execução lendo a propriedade Text.

Linhas Múltiplas-e Word Warp em Caixas de Texto

Por padrão, uma caixa de texto exibe uma única linha de texto e não exibe barras de rolagem. Se o texto é mais longo que o espaço disponível, só uma parte do texto será visível. O visual e o comportamento de uma caixa de texto podem ser mudados configurando duas propriedades, MultiLine e ScrollBars que só estão disponível em tempo de desenvolvento.
Nota A propriedade ScrollBars não deveria ser confundida com controles de barra de rolagem que não são prendidos a caixas de texto e têm o seu próprio conjunto de propriedades.

Configurar MultiLine para True permite uma caixa de texto a aceitar ou exibir linhas múltiplas de texto em tempo de execução. Uma caixa de texto de linhas múltiplas administra o envolvimento da palavra automaticamente contanto que não haja nenhuma barra de rolagem horizontal. A propriedade ScrollBars é configurada para 0-None por padrão. Palavra envolvida automática ajuda o usuário com o problema de inserir quebra linha ao término de linhas. Quando uma linha de texto é mais longa do que pode ser exibido em uma linha, a caixa de texto empurra o texto para a próxima linha.
Quebras de linhas não podem ser entradas na janela de Propriedades em tempo de desenvolvimento. Dentro de um procedimento, você cria uma quebra de linha inserindo um caracter de retorno seguido por um linefeed (caracter ANSI 13 e 10). Você também pode usar a constante vbCrLf para inserir uma combinação return/linefeed. Por exemplo, o procedimento de evento seguinte põe duas linhas de texto em uma caixa de texto de linhas múltiplas (Text1) quando a form é carregada:

```
Sub Form_Load ()
      Text1.Text = "Here are two lines" _
      & vbCrLf & "in a text box"
End Sub
```

Trabalhando com Texto em uma Caixa de Texto

Você pode controlar o ponto de inserção e comportamento de seleção em uma caixa de texto com as propriedades SelStart, SelLength e SelText. Estas propriedades estão só disponíveis em tempo de desenvolvimento.

Quando uma caixa de texto recebe o foco, por padrão o ponto de inserção ou a posição do cursor dentro da caixa de texto é à esquerda de qualquer texto existente. Pode ser movido pelo usuário com o teclado ou com o mouse. Se a caixa de texto perde e então recupera o foco, o ponto de inserção será onde quer que o usuário o tenha colocado por último.

Em alguns casos, este comportamento pode estar desconcertado ao usuário. Em uma aplicação de processamento de texto, o usuário poderia esperar que novos caracteres aparecessem depois de qualquer texto existente. Em uma aplicação de entrada de dados, o usuário poderia esperar que sua digitação substituisse qualquer entrada existente. As propriedades SelStart e SelLength lhe permitem modificar o comportamento para adaptar seu propósito.

A propriedade SelStart é um número que indica o ponto de inserção dentro de uma sequência de texto, com 0 sendo a posição no canto esquerdo. Se a propriedade SelStart é configurada com um valor igual ou maior que o número de caracter na caixa de texto, o ponto de inserção será colocado depois do último caracter, como mostrado na Figura 3.7. Para uma trabalhar com uam versão deste exemplo, veja Text.frm na aplicação Controls.vbp.

Figure 3.7 Insertion point example

A propriedade de SelLength é um valor numérico que fixa a largura do ponto de inserção. Fixando o SelLength para um número maior que 0 causas a numeração de caráter a ser selecionado e destacado, a partir do ponto de inserção atual. Figure 3.8 mostra o comportamento de seleção.

Figure 3.8 Selection example

Se o usuário inicia a digitação enquanto um bloco de texto está selecionado, o texto selecionado será substituído. Em alguns casos, você poderia querer substituir uma seleção de texto com um novo texto usando o comando colar. A propriedade de SelText é uma seqüência de texto que você pode nomear em tempo de execução para substituir a seleção atual. Se nenhum texto é selecionado, SelText inserirá seu texto ao ponto de inserção atual.

Para Mais Informação Para informação adicional sobre as propriedades do controle de caixa de texto, veja " Usando Controles Padrões do Visual Basic".

Controles que Apresentam Escolhas para o Usuários Uma introdução à caixa de verificação, botão de opção, caixa de lista, combo box, e controles de barra de rolagem.

A maioria das aplicações necessitam apresentar escolhas para seus usuários, variando de um simples yes/no (sim/não), opção de selecionar através de uma lista que contém centenas de possibilidades. Visual Basic inclui vários controles padrão que são útil para apresentar escolhas. A tabela seguinte resume estes controles e os usos apropriados deles.

Para obter esta característica	**Use este controle**
Um conjunto pequeno de escolhas das quais um usuário pode escolher uma ou mais opções.	Check boxes
Um conjunto pequeno de opções das quais um usuário pode escolher apenas uma.	Option buttons (use frames se grupos adicionais forem necessários)
Uma lista rolável de escolhas que o usuário pode escolher.	List box

Uma lista rolável de escolhas junto com um campo de edição de texto. O usuário pode escolher da lista ou pode digitar uma escolha no campo de edição.	Combo box

Check boxes, option buttons, list boxes, and combo boxes serão discutidos nas seções seguintes:

- **Selecionando Opções Individuais com Check Boxes** Os fundamentos para usar o controle caixa de verificação (check box).

Uma caixa de verificação indica se uma condição particular esta ligada ou desligada. Você usa caixas de verificação em uma aplicação para dar opções Verdadeiro/Falso (true/false) ou opções de Sim/Não (yes/no) para usuários. Por cada caixa de verificação trabalhar independentemente uma da outra, um usuário pode selecionar qualquer número de caixas de verificação ao mesmo tempo. Por exemplo em Figura 3.9, Tipo negrito e Itálico podem ser ambos verificados.

Figura 3.9 Check boxes

A Aplicação Caixa de Verificação

O exemplo Caixa de Verificação usa uma caixa de verificação para determinar se o texto é exibido em fonte regular ou itálico. Para uma versão de funcionamento deste exemplo, veja Check.frm no Controls.vbp exemplo de aplicação.
A aplicação tem uma caixa de texto, uma etiqueta, um botão de comando, e duas caixas de verificação, como mostrado em Figura 3.10.

Figura 3.10 Check box example

A tabela seguinte lista as colocações de propriedade para os objetos na aplicação.

Object	Property	Setting
Form	Name	
	Caption	frmCheck
		Check Box Example
Text box	Name	
	Text	txtDisplay
		Some sample text
First Check box	Name	
	Caption	chkBold
		&Bold
Second Check box	Name	
	Caption	chkItalic
		&Italic
Command button	Name	
	Caption	cmdClose
		&Close

Quando você assinala Tipo negrito ou Itálico, a propriedade Valor da caixa de verificação é fixada a 1; quando desmarcado, sua propriedade Valor é fixada a 0. O Valor padrão é 0, assim a menos que você mude Valor, a caixa de verificação será desmarcada quando a primeira vez que for

exibido. Você pode usar o vbChecked de constantes e vbUnchecked para representar os valores 1 e 0.

Eventos na Aplicação Caixa de Verificação

O evento Click para a caixa de verificação acontece assim que você clica a caixa. Este procedimento de evento testa para ver se a caixa de verificação foi selecionada (quer dizer, se seu Valor = vbChecked). nesse caso, o texto é convertido a tipo negrito ou itálico fixando as propriedades negrito ou Itálica do objeto Fonte devolvidas pela propriedade Fonte da caixa de texto.

```
Private Sub chkBold_Click ()
      If ChkBold.Value = vbChecked Then   ' If checked.
            txtDisplay.Font.Bold = True
      Else                                                    ' If not
checked.
            txtDisplay.Font.Bold = False
      End If
End Sub

Private Sub chkItalic_Click ()
      If ChkItalic.Value = vbChecked Then ' If checked.
            txtDisplay.Font.Italic = True
      Else                                                    ' If not
checked.
            txtDisplay.Font.Italic = False
      End If
End Sub
```

- **Agrupando Opções com Botões de Opção** Os fundamentos para usar o controle de botão de opção.

Botões de Opção presentes em um conjunto de dois ou mais escolhas para o usuário. Ao contrário das caixas de verificação, os botões de opção sempre deveriam trabalhar como parte de um grupo; um botão de opção selecionando limpa todos os outros botões imediatamente no grupo. Definindo um grupo de botão de opção fala para o usuário, " Aqui é um conjunto de escolhas das quais você pode escolher apenas uma. "

Por exemplo, no grupo de botão de opção mostrado em Figura 3.11, o usuário pode selecionar um de três botões de opção.

Figura 3.11 Selecting an option button

Criando um Grupo de Botão de Opção

Todos os botões de opção colocados diretamente em um form (quer dizer, não em uma frame ou picture box) constitua um grupo. Se você quer criar grupos de botão de opção adicionais, você tem que colocar alguns deles dentro de Frames ou Picture Box.

Todos os botões de opção dentro de qualquer frame ou constitui um grupo separado, como faz todo botão de opção dentro de uma Picture Box. Quando você cria um grupo separado deste modo, sempre desenhe a Frame ou Picture Box primeiro, e então desenhe os botões de opção em cima dela. Figura 3.12 mostra um form com dois grupos de botão de opção.

Figura 3.12 Option button groups

Um usuário pode selecionar só um botão de opção no grupo quando você desenha botões de opção em uma Frame.

Para agrupar controles em uma frame

1. Selecione o controle Frame da caixa de ferramentas e desenhe no form.
2. Selecione o controle botão de opção da caixa de ferramentas e o desenhe dentro da frame.
3. Repita o passo 2 para cada botão de opção adicional que você deseja somar à frame.

Desenhando a frame primeiro e então desenhando cada controle na frameo lhe permite mover a frame e os controles juntos. Se você tentar mover os controles existentes sobre uma frame, os controles não moverão com a frame.
Nota Se você tem controles existentes que você quer agrupar em uma frame, você pode selecionar todos os controles recortar e pode colar em uma frame ou picture box.
Repita o passo 2 para cada botão de opção adicional que você deseja somar à frame.

Recipientes para Controles

Enquanto controles são objetos independentes, um certo relacionamento de pai e filho existe entre forms e controles. Figura 3.12 demonstra como podem ser contidos botões de opção dentro de um form ou dentro de um controle frame.
Para entender o conceito de recipientes, você precisa entender que todos os controles são os filhos do form na qual eles são desenhados. De fato, a maioria dos controles suporta a propriedade de Pai somente de leitura que devolve o form na qual um controle está localizado. Sendo um filho afeta a colocação de um controle no form pai. As propriedades Left e Top de um controle são relativas ao form pai, e não podem ser movidos controles fora dos limites do pai. Movendo um recipiente move os controles também, e a posição do controle relativo às propriedades Left e Top do recipiente não mudam porque o controle move com o recipiente.

Selecionando ou Desabiliitando Botões de Opção

Um botão de opção pode ser selecionado por:

- Clicando-o em tempo de execução com o mouse.
- Usando Tab para ir ao grupo de botões de opção e usando as setas para selecionar um botão de opção dentro do grupo.
- Atribuindo sua propriedade Valor para True em código:

```
optChoice.Value = True
```

- Usando uma tecla de atalho especificada na legenda de uma etiqueta.

Fazer um botão default em um grupo de botão de opção, fixe sua propriedade Valor para True em tempo de desenvolvimento. Permanecerá selecionado até o usuário selecionar um botão de opção diferente ou mudá-lo por código.
Desabilitar um botão de opção, fixe sua propriedade Enabled para False. Quando o programa é executado aparecerá escurecido e significa que está indisponível.

A Aplicação Opções

O form mostrado na Figura 3.13 usa botões de opção para determinar o tipo de processador e sistema operacional para um computador imaginário. Quando o usuário seleciona um botão de opção qualquer no grupo, a legenda da etiqueta é mudada para refletir as escolhas atuais. Para uma versão de funcionamento deste exemplo, veja Options.frm na aplicação Controls.vbp.

Figura 3.13 Option button example

A tabela seguinte lista as colocações de propriedade para os objetos na aplicação.

Object	Property	Setting
Label	Name	
	Caption	lblDisplay
	(Empty)	
Command button	Name	
	Caption	cmdClose
	&Close	

First option button Name
Caption opt486
&486
Second option button Name
CaptionValue opt586
&PentiumTrue
Third option button Name
Caption opt686
P&entium Pro
Frame Name
Caption fraSystem
&Operating System
Fourth option button Name
Caption optWin95
Windows 95
Fifth option button Name
CaptionValue optWinNT
Windows NTTrue

Eventos na Aplicação Opções

A aplicação Opções responde a eventos como segue:

- Os eventos click para os primeiros três botões de opção nomeiam uma descrição correspondente a uma variável de texto a nível do form, strComputer.
- Os eventos click para os últimos dois botões de opção nomeiam uma descrição correspondente a uma segunda variável a nível do form, strSystem.

A chave para esta aproximação é o uso destas duas variáveis a nível do form, strComputer e strSystem. Estas variáveis contêm valores de texto diferentes, dependendo no qual foram selecionados os botões de opção por último.
Cada vez é selecionado um botão de opção novo, o código em seu evento click atualiza a variável apropriada:

```
Private Sub opt586_Click()
      strComputer = "Pentium"
      Call DisplayCaption
End Sub
```

Chama um sub procedimento, chamado DisplayCaption que concatena as duas variáveis e atualiza a propriedade Legenda da etiqueta, então:

```
Sub DisplayCaption()
      lblDisplay.Caption = "You selected a " & _
      strComputer & " running " & strSystem
End Sub
```

Um sub procedimento é usado porque o procedimento atualizar a propriedade Legenda é essencialmente o mesmo para todos os cinco botões de opção, só o valor das variáveis muda de um instante para outro. Isto o salva de ter que repetir o mesmo código em cada dos eventos click.
Para Mais Informação Variáveis e sub procedimentos são discutidos em detalhes em " Fundamentos de Programação ".

- **Usando List Boxes e Combo Boxes** Uma introdução aos controles list box e combo box

  Caixas de lista e Caixa de Combinação apresenta uma lista de escolhas para o usuário. Por default, as escolhas são exibidas verticalmente em uma única coluna, embora você possa montar colunas múltiplas também. Se o número de artigos excede o que pode ser exibido na caixa de combinação ou caixa de lista, barras de rolagem aparecem automaticamente no

controle. O usuário pode rolar então de cima para abaixo ou da esquerda para direita pela lista. Figura 3.14 mostra uma caixa de lista de coluna única.

Figura 3.14 Single-column list box

Um controle caixa de combinação combina as características de uma caixa de texto e uma caixa de lista. Este controle permite ao usuário ou selecionar digitando texto na caixa de combinação ou selecionando um artigo de sua lista. Figura 3.15 mostra uma caixa de combinação.

Figure 3.15 Combo box

Em contraste com um pouco de outros controles que contêm um único valor; por exemplo a propriedade Legenda da etiqueta ou a propriedade Texto da caixa de texto, caixas de lista e caixas de combinação contêm valores múltiplos ou uma coleção de valores. Eles têm métodos embutidos para adicionar, remover e recuperar valores das coleções deles em tempo de execução. Para adicionar vários artigos em uma caixa de lista nomeada List1, o código se pareceria com este:

```
List1.AddItem "Paris"
List1.AddItem "New York"
List1.AddItem "San Francisco"
```

Caixas de lista e caixas de combinação são um modo efetivo para apresentar um número grande de escolhas para o usuário em uma quantia limitada de espaço.

Para Mais Informação Para informação adicional sobre os controles caixa de lista e caixa de combinação, veja " Usando Controles Padrões do Visual Basic ".

· **Usando Barras de rolagem como Dispositivos de Entrada** UMA introdução breve para o controle barra de rolagem.

Embora as barras de rolagem estejam freqüentemente amarradas a caixas de texto ou janelas, você às vezes as verá seno usadas como dispositivos de entrada. Porque estes controles podem indicar a posição atual em uma escala, o controle barra de rolagem pode ser usado para controlar a entrada no programa individualmente - por exemplo, controlar o volume de som ou ajustar as cores em uma figura. O HScrollBar (horizontal) e VScrollBar (vertical) controles operam independentemente de outros controles e têm o seu próprio conjunto de eventos, propriedades, e métodos. Controles de barra de rolagem não são iguais às barras de rolagem embutidas que são prendidas a caixas de texto, caixas de lista, a caixas de combinação, ou Form MDI (caixas de texto e forms MDI têm uma propriedade ScrollBars para adicionar ou remover barras de rolagem que são prendidas ao controle).

A interface do Windows sugere usar controles slider como entrada em dispositivos em vez de barras de rolagem. Podem ser vistos exemplos de controles slider no Windows 95 no painel de controle. Um estilo do controle slider do Windows 95 está incluso nas edições Profissional e Enterprise do Visual Basic.

Para Mais Informação Para informação adicional sobre controle barra de rolagem, veja " Usando Controles Padrões do Visual Basic ".

Controles que exibem Figuras e Gráficos Uma introdução aos controles picture box, image, shape, e line.

Por causa do Windows ser uma interface gráfica de usuário, é importante ter um modo para exibir imagens gráficas na interface de sua aplicação. Visual Basic inclui quatro controles que torna fácil trabalhar com gráficos: o controle caixa de figura, o controle de imagem, o controle forma, e o controle linha.

Os controles imagem, forma e linha às vezes são referenciados por " controles gráficos de peso leve ". Eles requerem menos recursos de sistemas e por consequência exibem um pouco mais rápido que o controle caixa de figura; eles contêm um subconjunto das propriedades, métodos e eventos disponível na caixa de figura. Cada um serve melhor para um propósito particular.

Para obter essa característica	**Use este controle**
Um recipiente para outros controles.	Picture box
Imprimindo ou métodos de gráficos.	Picture box
Exibindo uma figura.	Image control or picture box
Exibindo um elemento gráfico simples	Shape or line control

Picture boxes, image controls, shape controls, e line serão discutidos nas seções seguintes:

- **Trabalhando com Controles Picture Box** Os fundamentos para usar o controle picture box.

O uso primário do controle de caixa de figura é exibir uma figura ao usuário. A figura atual que é exibida é determinado pela propriedade Picture. A propriedade Picture contém o nome do arquivo (e o caminho opcional) para o arquivo de figura que você deseja exibir.

Objetos Forms também têm uma propriedade Picture que pode ser fixada para exibir uma Figura diretamente no fundo do form.

Para Exibir ou substituir uma Figura em tempo de execução, você pode usar a função LoadPicture para alterer a propriedade Picture. Você informa o nome (e caminho opcional) para a figura e a função LoadPicture cuida dos detalhes para carregá-la e exibi-la:

```
picMain.Picture = LoadPicture("VANGOGH.BMP")
```

The picture box control has an AutoSize property that, when set to True, causes the picture box to resize automatically to match the dimensions of its contents. Take extra care in designing your form if you plan on using a picture box with the AutoSize enabled. The picture will resize without regard to other controls on the form, possibly causing unexpected results, such as covering up other controls. It's a good idea to test this by loading each of the pictures at design time.

Using the Picture Box as a Container

The picture box control can also be used as a container for other controls. Like the frame control, you can draw other controls on top of the picture box. The contained controls move with the picture box and their Top and Left properties will be relative to the picture box rather than the form.
A common use for the picture box container is as a toolbar or status bar. You can place image controls on it to act as buttons, or add labels to display status messages. By setting the Align property to Top, Bottom, Left, or Right, the picture box will "stick" to the edge of the form. Figure 3.16 shows a picture box with its Align property set to Bottom. It contains two label controls which could be used to display status messages.
Figure 3.16 Picture box used as a status bar

Other Uses for the Picture Box

The picture box control has several methods that make it useful for other purposes. Think of the picture box as a blank canvas upon which you can paint, draw or print. A single control can be used to display text, graphics or even simple animation.
The Print method allows you to output text to the picture box control just as you would to a printer. Several font properties are available to control the characteristics of text output by the Print method; the Cls method can be used to erase the output.
Circle, Line, Point and Pset methods may be used to draw graphics on the picture box. Properties such as DrawWidth, FillColor, and FillStyle allow you to customize the appearance of the graphics. Animation can be created using the PaintPicture method by moving images within the picture control and rapidly changing between several different images.
For More Information For additional information on the picture box control, see "Using Visual Basic's Standard Controls."

- **Lightweight Graphical Controls** The basics of using the image, shape, and line controls.

The image, shape and line controls are considered to be lightweight controls; that is, they support only a subset of the properties, methods, and events found in the picture box. Because of this, they typically require less system resources and load faster than the picture box control.

Using Image Controls Instead of Picture Boxes

The image control is similar to the picture box control but is used only for displaying pictures. It doesn't have the ability to act as a container for other controls, and it doesn't support the advanced methods of the picture box.
Pictures are loaded into the image control just as they are in the picture box: at design time, set the Picture property to a file name and path; at run time, use the LoadPicture function.
The sizing behavior of the image control differs from that of the picture box. It has a Stretch property while the picture box has an AutoSize property. Setting the AutoSize property to True causes a picture box to resize to the dimensions of the picture; setting it to False causes the picture to be cropped (only a portion of the picture is visible). When set to False (the default), the Stretch property of the image control causes it to resize to the dimensions of the picture. Setting the Stretch property to True causes the picture to resize to the size of the image control, which may cause the picture to appear distorted.
For More Information For additional information on the image control, see "Using Visual Basic's Standard Controls."

Using an Image Control to Create Your Own Buttons

An image control also recognizes the Click event, so you can use this control anywhere you'd use a command button. This is a convenient way to create a button with a picture instead of a caption. Grouping several image controls together horizontally across the top of the screen — usually within a picture box — allows you to create a toolbar in your application.
For instance, the Test Buttons example shows an image control that users can choose like they choose a command button. When the form is first displayed, the control displays one of three traffic icons from the Icon Library included with Visual Basic. Each time the image control is clicked, a different icon is displayed. (For a working version of this example, see Button.frm in the Controls.vbp sample application.)
If you inspect the form at design time, you will see that it actually contains all three icons "stacked" on top of each other. By changing the Visible property of the top image control to False, you allow the next image (with its Visible property set to True) to appear on top.
Figure 3.17 shows the image control with one of the traffic icons (Trffc10a.ico).
Figure 3.17 Image control with a traffic icon

To create a border around the image control, set the BorderStyle property to 1-Fixed Single.
Note Unlike command buttons, image controls do not appear pushed in when clicked. This means that unless you change the bitmap in the MouseDown event, there is no visual cue to the user that the "button" is being pushed.

For More Information For information on displaying a graphic image in an image control, see "Using Visual Basic's Standard Controls."

Using Shape and Line Controls

Shape and line controls are useful for drawing graphical elements on the surface of a form. These controls don't support any events; they are strictly for decorative purposes.

Several properties are provided to control the appearance of the shape control. By setting the Shape property, it can be displayed as a rectangle, square, oval, circle, rounded rectangle, or rounded square. The BorderColor and FillColor properties can be set to change the color; the BorderStyle, BorderWidth, FillStyle, and DrawMode properties control how the shape is drawn.

The line control is similar to the shape control but can only be used to draw straight lines.

For More Information For additional information on the shape and line controls, see "Using Visual Basic's Standard Controls."

- **The Images Application** An example of using the graphical controls.

  The form shown in Figure 3.18 uses four image controls, a shape control, a picture box, and a command button. When the user selects a playing card symbol, the shape control highlights the symbol and a description is displayed in the picture box. For a working version of this example, see Images.frm in the Controls.vbp sample application.

Figure 3.18 Image and shape control example

The following table lists the property settings for the objects in the application.

Object	Property	Setting
Picture box	Name	
	Align	picStatus
	Bottom	
First image control	Name	
	Picture	imgClub
	Spade.ico	
Second image control	Name	
	Picture	imgDiamond
	Diamond.ico	
Third image control	Name	
	Picture	imgHeart
	Heart.ico	
Fourth image control	Name	
	Caption	imgSpade
	Spade.ico	
Shape control	Name	
	Shape	
	BorderWidth	
	Height	
	Width	shpCard
	4 - Rounded Rectangle	
	2735	
	495	
Command button	Name	
	Caption	cmdClose
	&Close	

Events in the Images Application

The Images application responds to events as follows:

- The Click event in each of the image controls sets the Left property of the shape control equal to its own Left property, moving the shape on top of the image.
- The Cls method of the picture box is invoked, clearing the current caption from the status bar.
- The Print method of the picture box is invoked, printing the new caption on the status bar.

The code in the image control Click event looks like this:

```
Private Sub imgHeart_Click()
      shpCard.Left = imgClub.Left
      picStatus.Cls
      picStatus.Print "Selected: Club"
      shpCard.Visible = True
End Sub
```

Note that the first line in the Click event code assigns a value (the Left property of the image control) to the Left property of the shape control using the = operator. The next two lines invoke methods, so no operator is needed. In the third line, the value ("Selected: Club") is an argument to the Print method.
There is one more line of code in the application that is of interest; it is in the Form Load event.

```
shpCard.Visible = False
```

By setting the Visible property of the shape control to False, the shape control is hidden until the first image is clicked. The Visible property is set to True as the last step in the image control Click event.
For More Information For additional information on properties, methods, and events see "Programming Fundamentals."

Additional Controls An introduction to the other standard Visual Basic controls.

Several other standard controls are included in the Visual Basic toolbox. Some controls are useful for working with large amounts of data contained in an external database. Other controls can be used to access the Windows file system. Still other controls defy categorization, but are useful nonetheless.

You can also use ActiveX controls, previously called custom or OLE controls, in a Visual Basic application in the same way that you use the standard controls. The Professional and Enterprise editions of Visual Basic include several ActiveX controls as well as the capability to build your own controls. Additional ActiveX controls for just about any purpose imaginable are available for purchase from numerous vendors.

For More Information For additional information on using ActiveX controls, see "Managing Projects."

Data Access Controls

In today's business, most information is stored in one or more central databases. Visual Basic includes several data access controls for accessing most popular databases, including Microsoft Access and SQL Server.

- The data control is used to connect to a database. Think of it as a pipeline between the database and the other controls on your form. Its properties, methods, and events allow you to navigate and manipulate external data from within your own application.
- The DBList control is similar to the list box control. When used in conjunction with a data control, it can be automatically filled with a list of data from a field in an external database.
- The DBCombo control is like a combination of the DBList and a text box. The selected text in the text box portion can be edited, with the changes appearing in the underlying database.

- The DBGrid control displays data in a grid or table. When used in conjunction with a data control, it presents fully editable data from multiple fields in an external database.
- The MSFlexGrid control is a unique control for presenting multiple views of data. Think of it as a combination of a grid and a tree or outline control. At run time, the user can rearrange columns and rows to provide different views of the data.

For More Information For additional information on data access controls, see "Using Visual Basic's Standard Controls." For more information on working with external data, see "Accessing Data."

File System Controls

Visual Basic includes three controls for adding file handling capabilities to your application. These controls are normally used together to provide a view of drives, directories and files; they have special properties and events that tie them together.

- The DriveListBox control looks like a combo box. It provides a drop-down list of drives from which the user can select.
- The DirListBox is similar to a list box control, but with the built-in capability of displaying a list of directories in the currently selected drive.
- The FileListBox control also looks like a list box with a list of file names in a selected directory.

Note These controls are provided primarily for backward compatibility with applications created in earlier versions of Visual Basic. The common dialog control provides an easier method of working with file access. For more information on common dialog control, see "Miscellaneous Controls" later in this chapter.

Miscellaneous Controls

Several other standard controls are included in Visual Basic. Each serves a unique purpose.

- The timer control can be used to create an event in your application at a recurring interval. This is useful for executing code without the need for user interaction.
- The OLE container control is an easy way to add capabilities like linking and embedding to your application. Through the OLE container control, you can provide access to the functionality of any OLE-enabled application such as Microsoft Excel, Word and many others.
- The common dialog control adds built-in dialog boxes to your application for the selection of files, colors, fonts, and printing functions.

For More Information For additional information on any of the standard controls, see "Using Visual Basic's Standard Controls."

Understanding Focus A brief discussion of focus as it applies to controls.

Focus is the ability to receive user input through the mouse or keyboard. When an object has the focus, it can receive input from a user. In the Microsoft Windows interface, several applications can be running at any time, but only the application with the focus will have an active title bar and can receive user input. On a Visual Basic form with several text boxes, only the text box with the focus will display text entered by means of the keyboard.

The GotFocus and LostFocus events occur when an object receives or loses focus. Forms and most controls support these events.

Event	Description
GotFocus	Occurs when an object receives focus.
LostFocus	Occurs when an object loses focus. A LostFocus event procedure is primarily used for verification and validation updates, or for reversing or changing conditions you set up in the object's GotFocus procedure.

You can give focus to an object by:

- Selecting the object at run time.
- Using an access key to select the object at run time.
- Using the SetFocus method in code.

You can see when some objects have the focus. For example, when command buttons have the focus, they appear with a highlighted border around the caption (see Figure 3.19).

Figure 3.19 A command button showing focus

An object can receive focus only if its Enabled and Visible properties are set to True. The Enabled property allows the object to respond to user-generated events such as keyboard and mouse events. The Visible property determines whether an object is visible on the screen.

Note Frames, labels, menus, lines, shapes, images, and timers cannot receive focus. A form can receive focus only if it doesn't contain any controls that can receive the focus.

Setting the Tab Order An introduction to the concepts of tab order within a form.

The *tab order* is the order in which a user moves from one control to another by pressing the TAB key. Each form has its own tab order. Usually, the tab order is the same as the order in which you created the controls.

For example, assume you create two text boxes, Text1 and Text2, and then a command button, Command1. When the application starts, Text1 has the focus. Pressing TAB moves the focus between controls in the order they were created, as shown in Figure 3.20.

Figure 3.20 Tab example

To change the tab order for a control, set the TabIndex property. The TabIndex property of a control determines where it is positioned in the tab order. By default, the first control drawn has a TabIndex value of 0, the second has a TabIndex of 1, and so on. When you change a control's tab order position, Visual Basic automatically renumbers the tab order positions of the other controls to reflect insertions and deletions. For example, if you make Command1 first in the tab order, the TabIndex values for the other controls are automatically adjusted upward, as shown in the following table.

Control	TabIndex before it is changed	TabIndex after it is changed
Text1	0	1
Text2	1	2
Command1	2	0

The highest TabIndex setting is always one less than the number of controls in the tab order (because numbering starts at 0). Even if you set the TabIndex property to a number higher than the number of controls, Visual Basic converts the value back to the number of controls minus 1.

Note Controls that cannot get the focus, as well as disabled and invisible controls, don't have a TabIndex property and are not included in the tab order. As a user presses the TAB key, these controls are skipped.

Removing a Control from the Tab Order

Usually, pressing TAB at run time selects each control in the tab order. You can remove a control from the tab order by setting its TabStop property to False (0).

A control whose TabStop property has been set to False still maintains its position in the actual tab order, even though the control is skipped when you cycle through the controls with the TAB key.

Note An option button group has a single tab stop. The selected button (that is, the button with its Value set to True) has its TabStop property automatically set to True, while the other buttons have their TabStop property set to False.

Menu Basics An introduction to menu controls and the menu editor.

If you want your application to provide a set of commands to users, menus offer a convenient and consistent way to group commands and an easy way for users to access them.

Figure 3.21 Illustrates the elements of a menu interface on an untitled form.

Figure 3.21 The elements of a menu interface on a Visual Basic form

The *menu bar* appears immediately below the *title bar* on the form and contains one or more *menu titles*. When you click a menu title (such as File), a menu containing a list of menu items drops down. Menu items can include commands (such as New and Exit), separator bars, and submenu titles. Each menu item the user sees corresponds to a menu control you define in the Menu Editor (described later in this chapter).

To make your application easier to use, you should group menu items according to their function. In Figure 3.21, for example, the file-related commands New, Open, and Save As… are all found on the File menu.

Some menu items perform an action directly; for example, the Exit menu item on the File menu closes the application. Other menu items display a *dialog box* — a window that requires the user to supply information needed by the application to perform the action. These menu items should be followed by an ellipsis (…). For example, when you choose Save As… from the File menu, the Save File As dialog box appears.

A menu control is an object; like other objects it has properties that can be used to define its appearance and behavior. You can set the Caption property, the Enabled and Visible properties, the Checked property, and others at design time or at run time. Menu controls contain only one event, the Click event, which is invoked when the menu control is selected with the mouse or using the keyboard.

For More Information For additional information on menu controls, see "Creating Menus with the Menu Editor" in "Creating a User Interface."

Pop-up Menus

A *pop-up menu* is a floating menu that is displayed over a form, independent of the menu bar, as shown in Figure 3.22. The items displayed on the pop-up menu depend on the location of the pointer when the right mouse button is pressed; therefore, pop-up menus are also called *context menus*. (In Windows 95, you activate context menus by clicking the right mouse button.) You should use pop-up menus to provide an efficient method for accessing common, contextual commands. For example, if you click a text box with the right mouse button, a contextual menu would appear, as shown in Figure 3.22.

Figure 3.22 A pop-up menu

Any menu that has at least one menu item can be displayed at run time as a pop-up menu. To display a pop-up menu, use the PopupMenu method.
For More Information For additional information on creating pop-up menus, see "Creating Menus" in "Creating a User Interface."

Using the Menu Editor

With the Menu Editor, you can add new commands to existing menus, replace existing menu commands with your own commands, create new menus and menu bars, and change and delete existing menus and menu bars. The main advantage of the Menu Editor is its ease of use. You can customize menus in a completely interactive manner that involves very little programming.

To display the Menu Editor

- From the **Tools** menu, choose **Menu Editor**.

This opens the Menu Editor, shown in Figure 3.23
Figure 3.23 The Menu Editor

While most menu control properties can be set using the Menu Editor; all menu properties are also available in the Properties window. You would normally create a menu in the Menu Editor; however, to quickly change a single property, you could use the Properties window.
For More Information For additional information on creating menus and using the Menu Editor, see "Creating Menus" in "Creating a User Interface."

Prompting the User with Dialog Boxes An introduction to dialog boxes.

In Windows-based applications, dialog boxes are used to prompt the user for data needed by the application to continue or to display information to the user. Dialog boxes are a specialized type of form object that can be created in one of three ways:

- *Predefined* dialog boxes can be created from code using the MsgBox or InputBox functions.
- *Customized* dialog boxes can be created using a standard form or by customizing an existing dialog box.
- *Standard* dialog boxes, such as Print and File Open, can be created using the common dialog control.

Figure 3.24 shows an example of a predefined dialog box created using the MsgBox function.

Figure 3.24 A predefined dialog box

This dialog is displayed when you invoke the MsgBox function in code. The code for displaying the dialog box shown in Figure 3.24 looks like this:

```
MsgBox "Error encountered while trying to open file," & vbCrLf &
"please retry.", bExclamation, "Text Editor"
```

You supply three pieces of information, or arguments, to the MsgBox function: the message text, a constant (numeric value) to determine the style of the dialog box, and a title. Styles are available with various combinations of buttons and icons to make creating dialog boxes easy.

Because most dialog boxes require user interaction, they are usually displayed as modal dialog boxes. A *modal* dialog box must be closed (hidden or unloaded) before you can continue working with the rest of the application. For example, a dialog box is modal if it requires you to click OK or Cancel before you can switch to another form or dialog box.

Modeless dialog boxes let you shift the focus between the dialog box and another form without having to close the dialog box. You can continue to work elsewhere in the current application

while the dialog box is displayed. Modeless dialog boxes are rare; you will usually display a dialog because a response is needed before the application can continue. From the Edit menu, the Find dialog box in Visual Basic is an example of a modeless dialog box. Use modeless dialog boxes to display frequently used commands or information.

For More Information For additional information on creating dialog boxes, see "Creating a User Interface."

Sample application

Controls.vbp The code examples in this chapter are taken from the Controls.vbp sample application. If you installed the sample applications, you will find them in the \Controls subdirectory of the Visual Basic samples directory (\Vb\Samples\PGuide\Controls).