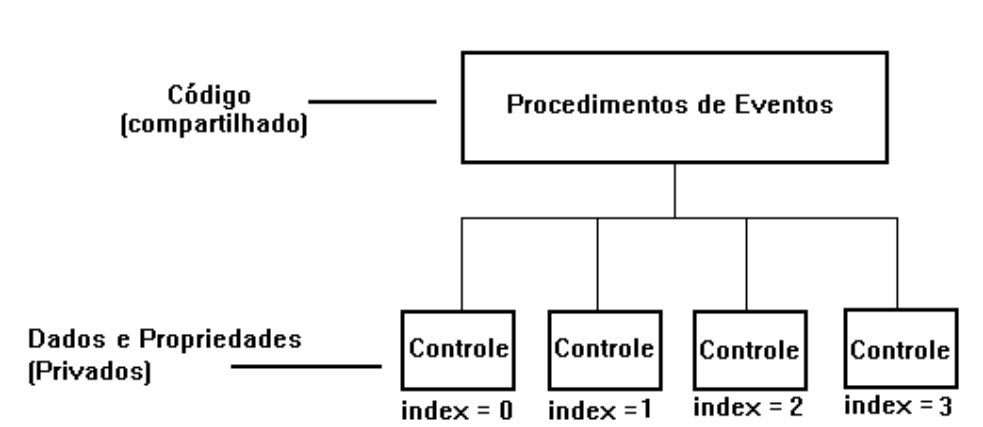
Controles e Menus Dinâmicos

Objetivo

Apresentar controles e menus dinâmicos, que são controles/menus que são criados em tempo de execução. Eles são, basicamente, utilizados quando necessita-se ter vários controles de um mesmo tipo ou menus que compartilhem os mesmos procedimentos.

Control Arrays



Control arrays é um grupo de controles que compartilham o mesmo nome, tipo e procedimentos de eventos. Entretanto cada controle é fisicamente separado do outro, e possui propriedades com valores próprios.

Cada controle possui um índice (propriedade *Index* de cada controle), e é referenciado no código da aplicação como *nome_controle(índice)*. O índice do controle á uma de suas propriedades, que é especificada quando você o cria.

Control arrays podem ser criados estaticamente ou dinamicamente. Estaticamente, quando são criados em tempo de projeto, e seu número de instâncias é fixo; dinamicamente, quando apenas uma instância é criada em tempo de projeto, e as outras apenas em tempo de execução, através de comandos do próprio Visual Basic.

Criando um Control Array Estaticamente

Um Control array estático é útil quando o programador sabe exatamente quantas instâncias de um objeto ele vai necessitar (ou seja ele vai desenhá-las todas no formulário em tempo de projeto), porém ele gostaria de reaproveitar código.

Veja o seguinte exemplo:

Um programador está criando uma tela de seleção de fornecedores. O usuário poderá selecionar por país e por estado, e conforme sua seleção, a query vai sendo montada em português em uma Text Box.

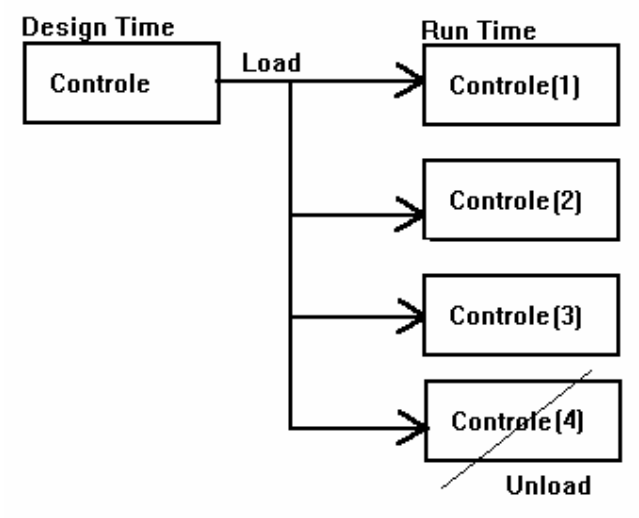
Existem dois painéis sobrepostos, um para seleção de estado e outro para seleção de país. Em cada painel existe um botão de "Sair". Ao ser pressionado, a query em português e em inglês (interna), é montada.

O código associado ao botão de "Sair" (código que monta a query) da seleção de Estado e da seleção de País é o mesmo, ao invés de duplicá-lo, um bom programador faria do botão "Sair" um Control Array.

Você pode criar um *Control Array* de três formas :

- Especificar a propriedade Index na janela de propriedades
- Dar o mesmo nome a dois controles
- Cortar/Copiar e Colar um controle

Criando Controles Dinamicamente



Uma vez criado o Control Array, você pode dinamicamente adicionar controles ao seu formulário através do comando *Load*. O comando *Load* cria uma nova instância do controle, copiando todas as propriedades do último controle criado, exceto *Index*, *Visible* e *TabIndex*.

O controle criado, tem inicialmente a propriedade Visible com valor *False*. Para torná-lo visível, atribuir o valor True à sua propriedade *Visible*.

Menus Dinâmicos

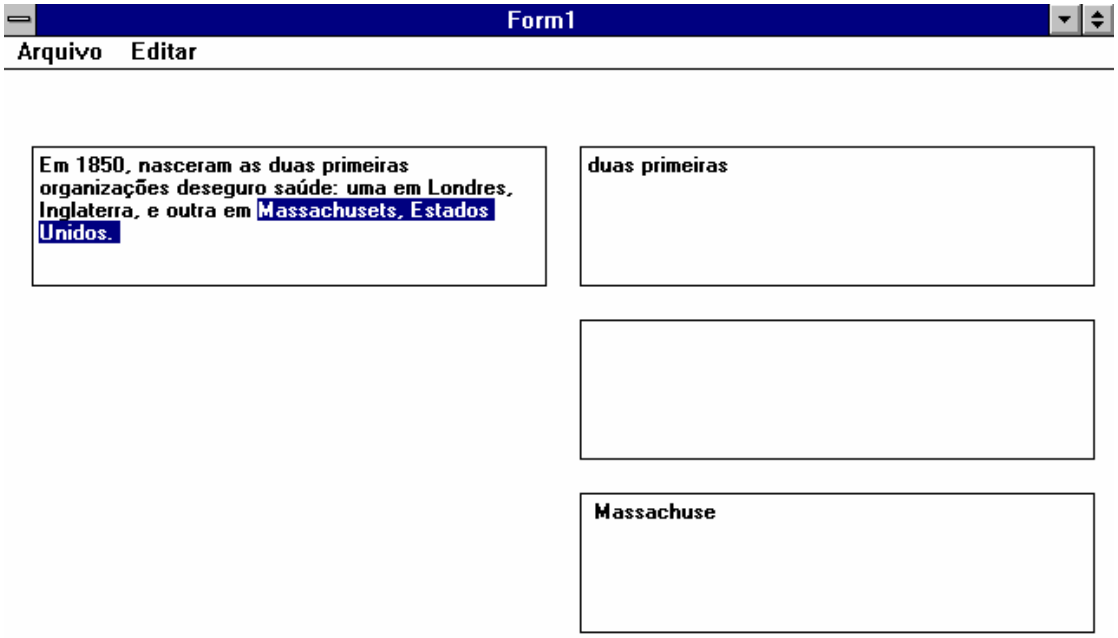
Menus dinâmicos seguem a mesma filosofia de controles dinâmicos. Você cria o Menu Array e, em tempo de execução, os ítens do menu são criados, à medida que vão se tornando necessários.

Assim como em controles dinâmicos, devemos atribuir valores a algumas propriedades, como enabled, visible e caption, no momento de sua criação.

Para carregar um menu, utiliza-se o comando *Load*, e para descarregar o comando *Unload*.

Exemplo - Controles e Menus Dinâmicos

Para exemplificar a utilização de menus e controles dinâmicos, criaremos uma aplicação que crie Text Boxes, e você consiga cortar e colar textos entre eles.



1. Inicializar o Visual Basic.

2. Criar duas TextBoxes, uma ao lado da outra, na parte superior do formulário (mas não colado à borda superior). Atribua as seguintes propriedades, respectivamente :

Name	TXBOrigem	TXBBuffer
Text	""	""
Visible	True	False
Index	""	0
Multiline	True	True

3. Criar um menu, como aprendido anteriormente, com a seguinte estrutura :

Caption	Name	Index	Visible	Enabled
Arquivo	MNUArquivo		True	True
Criar Buffer	MNUCriarBuffer		True	True
Apagar Buffer	MNUApagarBuffer		True	True
Sair	MNUSair		True	True
Editar	MNUEditar		True	False
Copiar	MNUCopiar	0	True	True
Cortar	MNUCortar	0	True	True
Colar	MNUColar	0	True	True

4. Criar uma variável *Integer* de nível de formulário, como aprendido anteriormente, com o nome de *Índice*.

5. Adicionar as seguintes linhas de código ao click de *MNUCriarBuffer* :

```
Sub MNUCriar_buffer_Click ()

    If indice < 3 Then

        If indice > = 1 Then
            Load TXBBuffer(indice)
            TXBBuffer(indice).Top = TXBBuffer(indice - 1).Top + 1500
            Load MNUCopiar(indice)
            Load MNUCortar(indice)
            Load MNUColar(indice)
        End If

        TXBBuffer(indice).Visible = True

        MNUCopiar(indice).Caption = "Copiar #" + Str(indice + 1)
        MNUCortar(indice).Caption = "Cortar #" + Str(indice + 1)
        MNUColar(indice).Caption = "Colar #" + Str(indice + 1)

        MNUCopiar(indice).Visible = True
        MNUCortar(indice).Visible = True
        MNUColar(indice).Visible = True

        MNUEditar.Enabled = True

        indice = indice + 1

    End If

End Sub
```

6. Adicionar as seguintes linhas de código ao click de *MNUApagarBuffer* :

```
Sub MNUApagarBuffer_Click ()

    If indice > 1 Then
        Unload TXBBuffer(indice - 1)

        Unload MNUCopiar(indice - 1)
        Unload MNUCortar(indice - 1)
        Unload MNUColar(indice - 1)

        indice = indice - 1
    Else
      If indice = 1 then
        TXBBuffer(indice - 1).visible = False
        indice = indice - 1
      Endif
    End If

End Sub
```

7. Adicionar as seguintes linhas de código ao click de *MNUCopiar* :

```
Sub MNUCopiar_Click (index As Integer)

    TXBBuffer(index).SelText = TXBOrigem.SelText
    TXBOrigem.SelLength = 0

End Sub
```

8. Adicionar as seguintes linhas de código ao click de *MNUCortar* :

```
Sub MNUCortar_Click (index As Integer)

    TXBBuffer(index).SelText = TXBOrigem.SelText
    TXBOrigem.SelText = ""

End Sub
```

9. Adicionar as seguintes linhas de código ao click de *MNUColar* :

```
Sub MnuColar_Click (index As Integer)

    TXBOrigem.SelText = TXBBuffer(index).Text
    TXBBuffer(index).Text = ""

End Sub
```

Executando a Aplicação

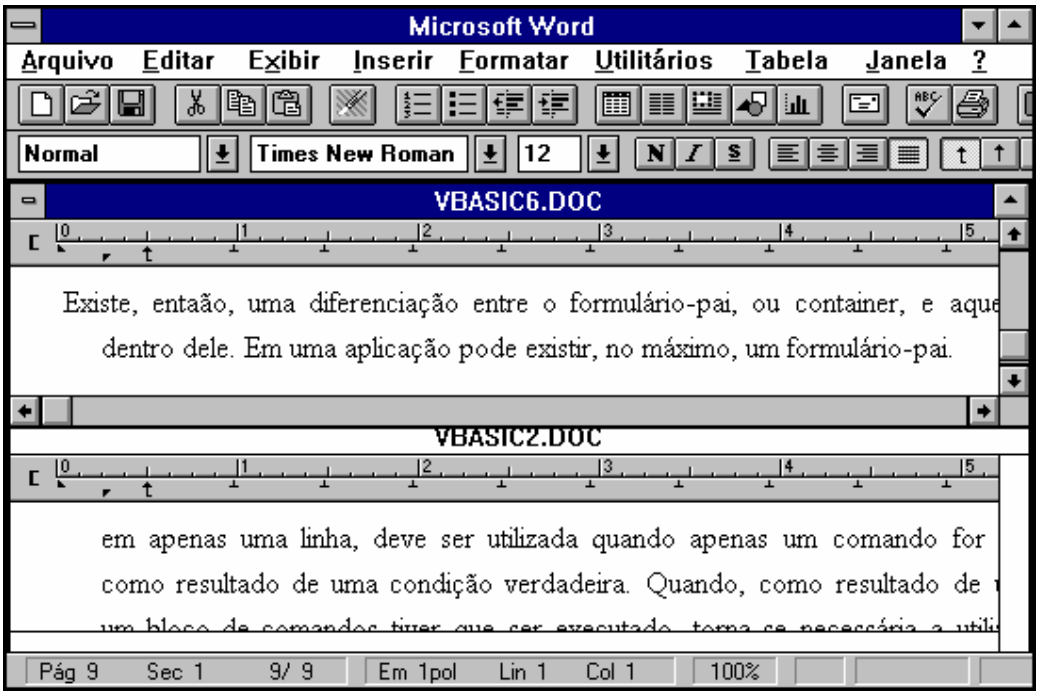
10. Digite texto na primeira Text Box. Crie Text Boxes dinamicamente e corte e cole textos entre elas. Quando estiver familiarizado com a aplicação, tente fazer com que a caixa de origem seja também um Control Array.

Multiple Document Interface (MDI)

Objetivo

Apresentar esta interface, tão comum em ambiente Windows. MDI permite que um formulário contenha vários outros formulários dentro dele.

.Aplicações MDI



O MDI (Multiple Document Interface) permite que uma aplicação possua várias janelas dentro de um único formulário.

Em uma aplicação MDI, existe um formulário, que pode ser denominado de *pai* ou *container*, que possui vários "filhos". Os controles e respectivos procedimentos, são compartilhados entre os filhos.

Uma aplicação pode conter, no máximo, um formulário-pai, porém vários tipos de filhos (Formulários MDI-filho diferentes). Formulários MDI e outros formulários podem coexistir dentro de uma mesma aplicação.

A maioria dos aplicativos do Windows têm interface MDI. Ex.: Microsoft Word, Microsoft Excel, etc. O Visual Basic é um exemplo de aplicação não MDI.

Criando e Fechando Formulários

Coerentemente à estrutura da linguagem Visual Basic, um formulário MDI é um objeto. Quando um novo formulário é criado, o que acontece, na realidade, é a criação de uma instância do objeto *formulário-filho*. Assim sendo, existe apenas um formulário-filho.

Para criar uma nova instância de um formulário-filho, usa-se palavra reservada **New**, no momento de da declaração da variável. Depois, basta mostrar o formulário.

```
Sub MNUNovoFilho_Click ( )
   Dim novo_filho as New nome_do_formulário
         novo_filho.show
End Sub
```

Palavra-Chave ME

Para ter acesso a uma instância de um formulário, usa-se a palavra reservada **Me**.

Veja o exemplo abaixo, que utiliza a palavra reservada Me para alterar o caption de cada formulário-filho (considera-se que existe uma variável global que é incrementada sempre que um novo formulário é carregado) :

```
Sub Formulário_filho_Load ()
   contador = contador + 1
   Me.Caption = "Formulário Filho"+ str( contador )
End Sub
```

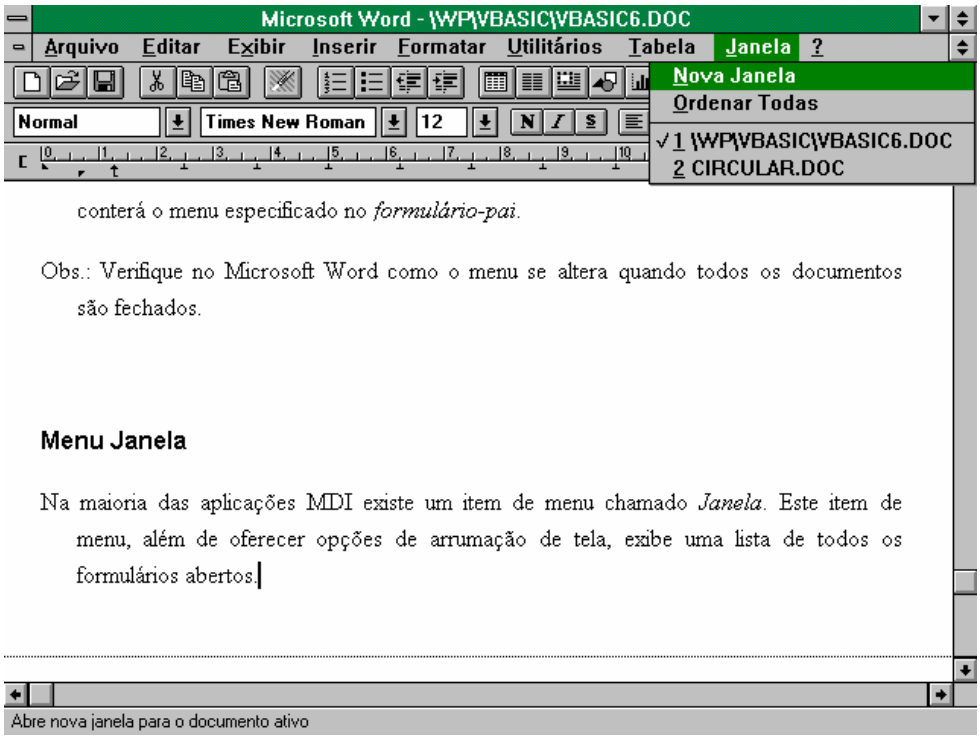
Menus em Aplicações MDI

Em aplicações MDI, se houver algum formulário MDI-filho aberto, a barra de menu conterá o menu especificado no *formulário-filho*. Caso contrário, a barra de menu conterá o menu especificado no *formulário-pai*.

Obs.: Verifique no Microsoft Word como o menu se altera quando todos os documentos são fechados.

Menu Janela

Na maioria das aplicações MDI existe um item de menu chamado *Janela*. Este item de menu, além de oferecer opções de arrumação de tela, exibe uma lista de todos os formulários abertos.



Para criar um item de menu de tipo Janela, é necessário:

1. Chamar a Menu Design Window a partir do formulário onde o menu deverá aparecer.
2. Adicionar um item de menu chamado "Janela".
3. Marcar a check-box Window List.

O Método Arrange

O método *Arrange*, quando aplicado a um formulário MDI_filho, arruma as instâncias deste relatório na tela.

Dependendo dos valores assumidos, as instâncias de formulário serão arrumados da seguinte maneira :

Arrumação	Parâmetro
Em cascata	0
Em blocos horizontais	1
Em blocos verticais	2

Barras de Ferramentas em Aplicações MDI

Como vimos antes, para se criar uma barra de ferramentas, utiliza-se o *Panel3D* e *Group Push Buttons*.

Em aplicações MDI, a barra de ferramentas é sempre criada no formulário-pai, para que seja comum a todas instâncias de formulário-filho. Isto apresenta dois problemas :

1. Os clicks de botões da barra de ferramentas, estando no formulário-pai, não podem chamar procedimentos dos formulários-filhos.

2. As variáveis necessárias nos procedimentos de cada formulário-filho, também não podem ser enxergados pelo formulário-pai.

A solução é :

- Fazer com que todos os procedimentos que necessitem ser chamados, tanto do formulário-pai como dos filhos, estejam codificados em um módulo (.BAS).

- Criar um vetor , em que cada item contenha o conjunto de váriaveis necessárias por instância de formulário (Fazer um vetor de tipos, onde o tipo será o conjunto de todas as variáveis de uma instância de formulário-filho). A associação de cada item deste vetor com cada instância de formulário filho deverá ser feita através da propriedade *tag* de cada formulário-filho, que deverá ter sido "setada" quando a instância do formulário foi criada.

Exemplo - Criando uma Aplicação MDI simples

1. Inicialize o Visual Basic.

2. Atribua o valor *True* à propriedade *MDIChild* do Form1.

3. Crie um Command Button e uma Picture Box no formulário.

4. Adicione as seguintes linhas de código ao click do botão :

```
Sub Command1_Click
      Picture1.picture = LoadPicture( c:\windows\leaves.bmp")
End Sub
```

5. Do menu File, selecione a opção *New MDI Form*.

6. Crie um menu no formulário-pai com a seguinte estrutura :

Caption	Name	Window List
&Arquivo	MNUArquivo	
&Novo Filho	MNUNovoFilho	
&Janela	MNUJanela	√
&Bloco	MNUBloco	

7. Adicionar as seguintes linhas de código ao click de MNUNovoFilho :

```
Sub MNUNovoFilho_Click()
   Dim filho as New Form1
         Filho.show
End Sub
```

8. Adicionar as seguintes linhas de código ao click de MNUBloco :

```
Sub MNUBloco
   MDIForm1.Arrange 2
EndSub
```

9. Executar a aplicação.

Tratamento de Erros

Objetivo

Apresentar uma maneira de tratar os erros de run-time ocorridos em qualquer aplicação Visual Basic. Se não houver rotinas de tratamento de erro, a aplicação aborta.

Tratamento de Erros no Visual Basic

Para criar um sistema de tratamento de erros para uma aplicação Visual Basic, o operador deve seguir basicamente três passos :

- Ativar o Tratamento de Erros do Visual Basic;
- Escrever a rotina de tratamento de erros. Esta rotina deverá incluir o "display" de mensagens para o usuário assim como, ações corretivas para o problema;
- Indicar ao sistema para onde a aplicação deverá retornar após o tratamento de erro.

Ativando o Tratamento de Erro

A ativação do tratamento de erros no Visual Basic se faz através da cláusula *On Error* . Nesta cláusula o programador indica para onde a aplicação deve pular em caso de erro, que pode ser um label ou um número de linha.

A cláusula On Error deve ser especificada no início de uma rotina qualquer e o label que indica o início da rotina de tratamento de erro deverá estar dentro da mesma rotina.

Sintaxe :

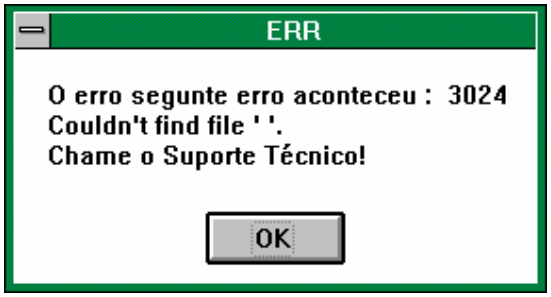
On Error Goto < Label >

Enviando Mensagens de Erro e Tratando o Erro

Quando ocorre um erro, o Visual Basic gera um número referente ao erro ocorrido. Este número pode ser obtido através da função *ERR*. A mensagem de texto relativa ao erro ocorrido pode ser fornecido através da função *ERROR*.

Exemplo :

```
MsgBox "O erro seguinte erro aconteceu : " + Str(Err) + Chr(10) +
       " + Error(Err) + Chr(10) + "Chame o Suporte Técnico!"
```



Quando um erro ocorre, o programador pode, além de apenas mostrar o erro ocorrido e finalizar a aplicação, pode também mostrar uma mensagem para usuário pedindo que este tome uma ação corretiva, dependendo do erro. Isso pode ser implementado se dentro da rotina de tratamento de erros, o programador testar o tipo do erro ocorrido (através da função *ERR*), e para cada tipo de erro tomar uma atitude diferente.

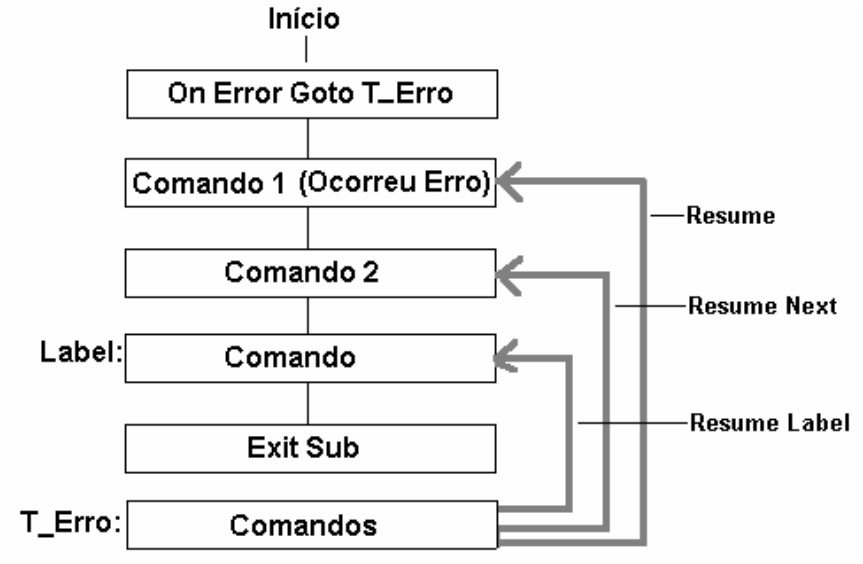
Veja o exemplo abaixo, onde a aplicação está tentando copiar um arquivo de um disquete que não está no drive. Neste caso existe uma ação corretiva e a aplicação não necessita ser abortada.

Exemplo :

```
Sub Command3D1_Click ()
On Error GoTo T_Erro
    FileCopy "a:arq1.txt", "b:arq2.txt"
    Exit Sub
T_Erro:
    Select Case Err
        Case 71
            MsgBox "Não existe disquete no drive. Coloque um e tente
                    novamente!"
        Case 55
            MsgBox "Não se pode copiar um arquivo aberto. Feche-o e
                    tente novamente"
        Case Else
            MsgBox "Um erro aconteceu. Chame o suporte Técnico!"
    End Select
    Resume Next
End Sub
```

Retorno de Rotinas de Tratamento de Erro

Quando ocorre um erro em uma aplicação, se o programador tiver usado a cláusula *On Error*, o fluxo do programa é desviado para uma rotina de tratamento de erros. Depois deste desvio, se corrigido o erro, de onde o programa deverá prosseguir ? Isto quem define é o programador, através um entre três comandos : *Resume*, *Resume Next*, *Resume* < *label* >. Existe ainda um quarto comando que pode ser utilizado para sair de uma rotina de erro, o *End*, que serve para finalizar uma aplicação. Sem um destes quatro comandos dentro de uma rotina de tratamento de erro, uma erro de compilação ocorre.



Resume O comando *resume* faz com que a aplicação tente continuar seu fluxo a partir de comando onde ocorreu o erro. Se o erro não tiver sido corrigido, novo erro ocorre. Se ele continuar não sendo corrigido, a aplicação entra em "loop".

Resume Next O comando *Resume Next* faz com que a aplicação tente continuar seu fluxo a partir do comando seguinte ao comando onde ocorreu o erro. Se a partir deste ponto existam comandos cujo funcionamento dependiam do sucesso da operação onde o erro havia acontecido, novos erros acontecerão.

Resume < Label > Este comando faz com que a aplicação tente continuar seu fluxo de execução a partir de um label definido pelo operador dentro da mesma rotina. Funciona como um "goto".

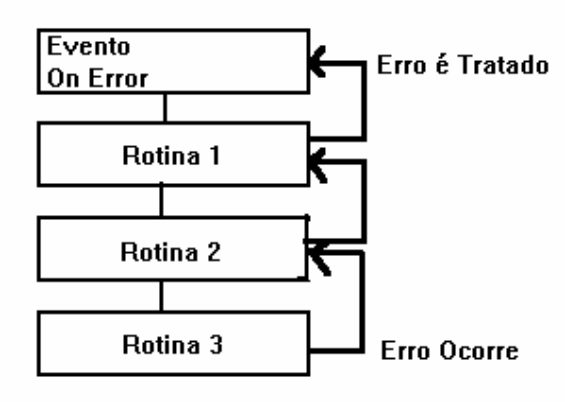
End Finaliza a aplicação. Cuidado para não esquecer de fechar bancos de dados abertos.

Onde deve ser especificada a cláusula On Error

O usuário pode especificar uma cláusula On Error em cada módulo da aplicação (sub-rotina, função ou evento do sistema) e codificar rotinas de tratamento de erro para cada uma delas. No entanto, isto não é necessário.

Sempre que ocorre um erro dentro de uma rotina, o sistema verifica se foi especificado tratamento de erro para aquela rotina. Em caso positivo, o erro é

tratado dentro daquela rotina e a execução do sistema prossegue dali, dependendo do que foi especificado pelo usuário. Em caso negativo, o sistema verifica se foi especificado tratamento de erro na rotina chamadora daquela função ou rotina. Caso exista, o tratamento é efetuado, caso contrário a procura por tratamento de erro prossegue na árvore de rotinas chamadoras, até que esta acabe. Se nenhum tratamento de erro tiver sido especificado para aquela árvore de chamadas, um erro de execução ocorre.



Quando um erro é passado para rotinas chamadoras para que seja tratado, a execução da aplicação prossegue a partir da rotina onde o erro foi tratado, não necessariamente a rotina onde o erro ocorreu.